ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Sindopoly: Διαδικτυακό παιχνίδι με θέμα το Πανεπιστήμιο χρησιμοποιώντας προηγμένες τεχνικές προγραμματισμού

Του φοιτητή
Παπαδόπουλου Νικόλαου
Αρ. Μητρώου: 2019128

Επιβλέπων
Σιδηρόπουλος Αντώνης
Αναπληρωτής Καθηγητής

Θεσσαλονίκη 31-05-2025

Τίτλος Δ.Ε.: Διαδικτυακό παιχνίδι με θέμα το Πανεπιστήμιο χρησιμοποιώντας προηγμένες τεχνικές προγραμματισμού

Κωδικός Δ.Ε. 24138

Ονοματεπώνυμο φοιτητή: ΠΑΠΑΔΟΠΟΥΛΟΣ ΝΙΚΟΛΑΟΣ

Ονοματεπώνυμο εισηγητή: ΣΙΔΗΡΟΠΟΥΛΟΣ ΑΝΤΩΝΗΣ

Ημερομηνία ανάληψης Δ.Ε. 05-03-2024

Ημερομηνία περάτωσης Δ.Ε. 31-05-2024

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Αφιερώνεται στον Αλέξανδρο Παπαδόπουλο»

Πρόλογος

Η επιλογή μου να αναπτύξω τη διπλωματική εργασία με τίτλο «Sindopoly: Διαδικτυακό παιχνίδι με θέμα το Πανεπιστήμιο χρησιμοποιώντας προηγμένες τεχνικές προγραμματισμού» πάρθηκε εξαιτίας του ενδιαφέροντός μου να δημιουργήσω ένα παιχνίδι που συνδυάζει την αγάπη μου για την τεχνολογία με το περιβάλλον του Πανεπιστημίου, στο οποίο φοίτησα. Στο παιχνίδι παρουσιάζονται χαρακτηριστικά κτίρια και σημεία της Πανεπιστημιούπολης του ΔΙΠΑΕ, τα οποία στην προκειμένη περίπτωση αποτελούν «ιδιοκτησίες» και ακολουθούν τις κλασικές αρχές του διάσημου παιχνιδιού «Monopoly». Ήταν μία ευκαιρία να εφαρμόσω τις γνώσεις που απέκτησα ως φοιτητής για την ανάπτυξη ενός διαδραστικού διαδικτυακού παιχνιδιού, ενώ ταυτόχρονα παρείχε έναν τρόπο να τιμήσω και να αναδείξω το περιβάλλον που με υποστήριξε κατά τη διάρκεια των σπουδών μου.

Περίληψη

Το θέμα της Δ.Ε. μου είναι η ανάπτυξη του «Sindopoly: Διαδικτυακό παιχνίδι με θέμα το Πανεπιστήμιο χρησιμοποιώντας προηγμένες τεχνικές προγραμματισμού», δηλαδή ενός διαδικτυακού παιχνιδιού που βασίζεται πάνω στο κλασικό παιχνίδι «Monopoly», αλλά με την ιδιαιτερότητα ότι οι «ιδιοκτησίες» στο παιχνίδι αποτελούνται από κτίρια και διάφορους χώρους ενός και εκτός του πανεπιστημιακού κάμπους του ΔΙΠΑΕ. Οι παίκτες μπορούν να αγοράζουν τα ακίνητα μαθαίνοντας έτσι καλύτερα το πανεπιστημιακό περιβάλλον.

Το παιχνίδι αναπτύχθηκε χρησιμοποιώντας προηγμένες τεχνικές προγραμματισμού και σύγχρονες web τεχνολογίες. Προσφέρει μια ευκαιρία για τους παίκτες να έρθουν σε επαφή με το πανεπιστήμιο μας κάνοντας μια προσομοίωση ένα οικείο, αγαπημένο παιχνίδι. Αυτό επιτυγχάνεται επίσης με πολλές αναφορές σε παλαιότερα συμβάντα ή συχνά φαινόμενα εντός της Πανεπιστημιούπολης.

Τα αποτελέσματα της εργασίας αποδεικνύουν πως η υλοποίηση διαδραστικών παιχνιδιών είναι δυνατό να προσφέρει ουσιαστική γνώση για τον προγραμματιστή και δημιουργό. Η εμπειρία που λήφθηκε βοήθησε να κατανοηθούν καλύτερα οι δυνατότητες της σύγχρονης αρχιτεκτονικής του Web. Ειδικότερα βοήθησε σε μια ολιστική προσέγγιση των κλάδων του Web Development, από Databases και Business Logic μέχρι και APIs και UI/UX Design.

Sindopoly: A University-themed online game using advanced programming techniques

Papadopoulos Nikolaos

Abstract

The subject of my thesis is the development of “Sindopoly: A University-themed online game using advanced programming techniques”, i.e. an online game based on the classic “Monopoly” game, but with the peculiarity that the "properties" in the game consist of buildings and various spaces on and off the campus of the International Hellenic University. Players can buy the properties, thus learning more about the university environment.

The game was developed using advanced programming techniques and modern web technologies. It offers an opportunity for players to get in touch with our university by simulating a familiar, beloved game. This is also achieved with many references to past events or frequent phenomena within the University Campus.

The results of the work prove that the implementation of interactive games can provide substantial knowledge for the programmer and creator. The experience gained helped me to better understand the possibilities of modern Web architecture. In particular, it helped in a holistic approach to the disciplines of Web Development, from Databases and Business Logic to APIs and UI/UX Design.

Ευχαριστίες

Με την ολοκλήρωση της παρούσας Διπλωματικής Εργασίας, θα ήθελα να προσφέρω τις ευχαριστίες μου προς όλους όσους συνέβαλαν στην επιτυχή ολοκλήρωσή της. Αρχικά, ευχαριστώ θερμά τους γονείς μου, την οικογένειά μου, και όλους τους ανθρώπους γενικότερα που με υποστήριξαν, ο καθένας με τον δικό του τρόπο. Επιπλέον, χρωστάω ένα μεγάλο ευχαριστώ στον επιβλέποντα καθηγητή μου, τον κύριο Σιδηρόπουλο Αντώνη, ο οποίος αποτέλεσε σύμβουλος και σημαντικός παράγοντας στη διατριβή μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κατάλογος Κωδίκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
DTO	Data Transfer Object
CRUD	Create Read Update Delete

Κεφάλαιο 1ο: Εισαγωγή

1.1 Ιστορικό υπόβαθρο

Η μετάβαση των επιτραπέζιων παιχνιδιών σε ψηφιακές πλατφόρμες αποτελεί ένα εξαιρετικά ενδιαφέρον φαινόμενο, το οποίο ξεκίνησε να γίνεται εμφανές ήδη από την εποχή που υπολογιστές με βασικές γραφικές διεπαφές ανέλαβαν την προσομοίωση κλασικών παιχνιδιών. Παραδοσιακά επιτραπέζια όπως το «Σκάκι», η «Τρίλιζα» και αργότερα το «Monopoly» βρήκαν τον δρόμο τους σε ψηφιακές εκδοχές που προσέφεραν τη δυνατότητα παιχνιδιού εξ αποστάσεως, τη βελτίωση των γραφικών, αλλά και την ενσωμάτωση σύγχρονων τεχνολογιών. Η ραγδαία ανάπτυξη του διαδικτύου και η διάδοση των φορητών συσκευών (smartphones, tablets) οδήγησε σε ακόμα μεγαλύτερη δημοφιλία των ψηφιακών εκδόσεων επιτραπέζιων παιχνιδιών, καθώς οι χρήστες απέκτησαν τη δυνατότητα να παίζουν οπουδήποτε και οποτεδήποτε χωρίς περιορισμούς χώρου ή χρόνου.

Σε τεχνικό επίπεδο, η υλοποίηση ενός ψηφιακού επιτραπέζιου παιχνιδιού απαιτεί την ανάπτυξη αξιόπιστων μηχανισμών τυχαίων γεγονότων (π.χ. για τη ρίψη των ζαριών), αποτελεσματικής διαχείρισης κατάστασης παικτών (π.χ. χρήματα, ιδιοκτησίες), καθώς και σύγχρονων τεχνικών για την εξασφάλιση της ακεραιότητας της πληροφορίας σε πραγματικό χρόνο. Οι διακομιστές (servers) που φιλοξενούν τέτοια παιχνίδια πρέπει να ανταποκρίνονται σε αιτήσεις παικτών, να ενημερώνουν ταυτόχρονα πολλαπλές συνδέσεις για τις κινήσεις που εκτελούνται και να διατηρούν ένα συνεπές «state» σε ενδεχόμενες διακοπές. Επιπλέον, η επικοινωνία μεταξύ πελάτη (client) και διακομιστή (server) έχει διευκολυνθεί από τη χρήση τεχνολογιών όπως τα WebSockets ή τα RESTful APIs, ενώ βάσεις δεδομένων όπως SQL ή NoSQL όπως και τα in-memory συστήματα (π.χ. το Redis) έχουν αναλάβει την αποθήκευση και την άμεση ανάκτηση των δεδομένων του παιχνιδιού, πιο συγκεκριμένα σε πραγματικό χρόνο, το οποίο παίζει κρίσιμο ρόλο στα ψηφιακά παιχνίδια.

Παράλληλα, τα θεματικά παιχνίδια παίζουν όλο και πιο σημαντικό ρόλο στο να κεντρίσουν το ενδιαφέρον των χρηστών και να ενθαρρύνουν την ενασχόληση των παικτών. Όταν ένα κλασικό επιτραπέζιο αποκτά συγκεκριμένη θεματολογία, για παράδειγμα μια πόλη, χώρα ή ένα πανεπιστήμιο, οι παίκτες αναπτύσσουν έναν πιο στενό δέσιμο με το περιβάλλον του παιχνιδιού. Οι εμβληματικοί χώροι, τα γνωστά κτίρια και οι αναφορές σε οικείες τοποθεσίες δημιουργούν μια αίσθηση οικειότητας και ενισχύουν την αφοσίωση των χρηστών. Στο πλαίσιο ενός πανεπιστημίου στην προκειμένη περίπτωση, η απεικόνιση διάφορων φοιτητικών κτιρίων, αιθουσών ή συχνών σημείων συνάντησης των φοιτητών μπορεί να τραβήξει την προσοχή και των ενεργών φοιτητών και των απόφοιτων εφόσον τους επιτρέπει να ξαναζήσουν στιγμές της ακαδημαϊκής τους ζωής.

1.2 Αιτιολόγηση του έργου

Το «Sindopoly» αποτελεί μια απόπειρα προσαρμογής του κλασικού παιχνιδιού «Monopoly» στο περιβάλλον και την κουλτούρα του ΔΙΠΑΕ. Αντί για τις συνηθισμένες «ιδιοκτησίες» που βλέπουμε στην παραδοσιακή εκδοχή του «Monopoly», εδώ πρωταγωνιστούν κτίρια, χώροι, σημεία συνάντησης και υπηρεσίες της Πανεπιστημιούπολης. Η συγκεκριμένη επιλογή εξυπηρετεί έναν διττό σκοπό: αφενός προσθέτει ένα στοιχείο οικειότητας και σύνδεσης με τον χώρο όπου οι φοιτητές και το εκπαιδευτικό προσωπικό περνούν καθημερινά τον χρόνο τους, και αφετέρου ενισχύει τη διασκεδαστική διάσταση του παιχνιδιού χάρη στη χρήση πραγματικών τοποθεσιών.

Με αυτόν τον τρόπο το «Sindopoly» συνδυάζει την αίσθηση νοσταλγίας που προσφέρει το «Monopoly» με την πολιτισμική ή συναισθηματική αξία που έχει το ΔΙΠΑΕ. Ο παίκτης δεν καλείται απλώς να

αγοράζει και να πουλάει ακίνητα, αλλά να διαχειρίζεται και να αναπτύσσει πραγματικούς χώρους του ΔΙΠΑΕ, όπως το κτίριο της Ηλεκτρονικής, τη Λέσχη ή το Γυμναστήριο, το οποίο στοιχείο επιτρέπει τους συμμετέχοντες να νιώθουν ότι «επενδύουν» σε σημεία του πανεπιστημίου που πιθανώς χρησιμοποιούν στην καθημερινή τους ζωή. Επιπλέον, η χρήση κτιρίων και χώρων του ΔΙΠΑΕ ως εικονικές «ιδιοκτησίες» προσφέρει μια εκπαιδευτική συνιστώσα, αν και έμμεσα: οι παίκτες εξοικειώνονται με τη δομή της Πανεπιστημιούπολης και ενδεχομένως μαθαίνουν για χώρους που δεν είχαν εξερευνήσει.

1.3 Καθορισμός του προβλήματος

Η ανάπτυξη του «Sindopoly» προέκυψε από την ανάγκη αντιμετώπισης ενός κενού στην προβολή του ΔΙΠΑΕ στον έξω κόσμο. Παρά το γεγονός ότι το ΔΙΠΑΕ διαθέτει ποικίλα κτίρια, εγκαταστάσεις και χώρους αναψυχής, οι περισσότεροι φοιτητές και επισκέπτες γνωρίζουν μόνο ένα μικρό μέρος από αυτά, συνήθως μόνο όσα σχετίζονται άμεσα με τις σπουδές τους ή την καθημερινότητά τους. Επιπλέον δεν υπάρχει κάποια ψηφιακή εφαρμογή που να συνδυάζει πληροφορίες για τις κτιριακές υποδομές με τη ψυχαγωγία, με αποτέλεσμα να κεντρίζεται το ενδιαφέρον των φοιτητών στο να εξερευνήσουν σε βάθος τον χώρο όπου σπουδάζουν.

Το «Sindopoly» φιλοδοξεί να καλύψει αυτό το κενό, προτείνοντας ένα παιχνίδι στρατηγικής που αξιοποιεί τα πραγματικά κτίρια και τις τοποθεσίες του πανεπιστημίου ως «ιδιοκτησίες». Έτσι, το κεντρικό πρόβλημα που επιχειρεί να επιλύσει το παιχνίδι είναι η έλλειψη ενός ελκυστικού μέσου για την ανάδειξη του πανεπιστημιακού περιβάλλοντος.

1.4 Παραδοτέα

Το πρώτο και πιο σημαντικό παραδοτέο του έργου είναι ένα ολοκληρωμένο ψηφιακό παιχνίδι εν ονόματι «Sindopoly», το οποίο θα μπορεί να εκτελεστεί άμεσα μέσα από οποιονδήποτε σύγχρονο διακομιστή (web browser). Στο πλαίσιο αυτό έχει δοθεί ιδιαίτερη έμφαση στην ανάπτυξη ενός διαδραστικού περιβάλλοντος που προσομοιώνει ένα μέρος των κανόνων και των μηχανισμών του κλασικού παιχνιδιού «Monopoly». Ο πηγαίος κώδικας του ολοκληρωμένου έργου «Sindopoly» μπορεί να βρεθεί σε αποθετήριο «GitHub» (GitHub repository) του συγγραφέα αυτής της Δ.Ε. [1].

Το δεύτερο βασικό παραδοτέο αφορά την αναλυτική τεκμηρίωση (Documentation) της διαδικασίας ανάπτυξης του παιχνιδιού. Σε αυτή τη Δ.Ε. αποτυπώνεται η γενική αρχιτεκτονική του συστήματος, οι τεχνολογίες που αξιοποιήθηκαν (τόσο στο κομμάτι του backend όσο και του frontend) καθώς και οι μεθοδολογίες που επιλέχθηκαν για τον σχεδιασμό και την υλοποίηση του κώδικα. Η τεκμηρίωση περιλαμβάνει επίσης διαγράμματα ροής, UML μοντέλα, περιπτώσεις χρήσης καθώς και περιγραφές των βασικών δομικών μονάδων.

Τέλος, το τρίτο παραδοτέο αφορά τις οδηγίες χρήσης (user manual) για τους παίκτες και τους διαχειριστές της εφαρμογής. Εδώ περιγράφονται με ακρίβεια τα βήματα που πρέπει να ακολουθήσουν οι παίκτες για τη δημιουργία λογαριασμού, την έναρξη νέας παρτίδας καθώς και τη διαχείριση των κινήσεών τους εντός του παιχνιδιού.

1.5 Επίλογος

Η ολοκλήρωση του πρώτου κεφαλαίου προσφέρει μια σαφή επισκόπηση του θεωρητικού υπόβαθρου, της αναγκαιότητας και των επιδιώξεων που διέπουν την ανάπτυξη του διαδικτυακού παιχνιδιού «Sindopoly». Αρχικά παρουσιάστηκε το ιστορικό πλαίσιο που περιγράφει πώς τα επιτραπέζια παιχνίδια εξελίχθηκαν σε ψηφιακές πλατφόρμες.

Στη συνέχεια η αιτιολόγηση του έργου ανέδειξε τα μοναδικά χαρακτηριστικά του πανεπιστημιακού περιβάλλοντος και τον τρόπο με τον οποίο η ενσωμάτωσή του στο γνωστό πλαίσιο του παιχνιδιού «Monopoly» προσφέρει μια προστιθέμενη αξία για την πανεπιστημιακή κοινότητα. Παράλληλα, ο καθορισμός του προβλήματος τόνισε την απουσία ενός ψυχαγωγικού αλλά ταυτόχρονα ενημερωτικού μέσου προβολής των χώρων του ΔΙΠΑΕ. Σημαντικό ρόλο διαδραματίζουν οι παραδοχές που σχετίζονται με την πολυπλοκότητα της υλοποίησης ενός τέτοιου έργου και τη μελλοντική εξέλιξη του παιχνιδιού.

Τέλος, η αναφορά στα παραδοτέα έδωσε μια σαφή εικόνα του τι πρόκειται να υλοποιηθεί και να παραδοθεί στο πλαίσιο αυτής της διπλωματικής εργασίας. Στα επόμενα κεφάλαια θα παρουσιαστούν αναλυτικά οι επιμέρους πτυχές της υλοποίησης, από το θεωρητικό υπόβαθρο έως τις τεχνικές λεπτομέρειες και τις δοκιμές λειτουργίας.

Κεφάλαιο 2ο: Διερεύνηση & Επιλογή Τεχνολογιών

2.1 Εισαγωγή

Το δεύτερο κεφάλαιο έχει ως στόχο να διερευνήσει το ευρύτερο πλαίσιο τεχνολογιών που θα μπορούσαν να αξιοποιηθούν για την ανάπτυξη του «Sindopoly», αλλά και για οποιουδήποτε άλλου διαδικτυακού παιχνιδιού που έχει την ανάγκη ανταπόκρισης σε αιτήματα των παικτών σε πραγματικό χρόνο. Με αυτόν τον τρόπο το κεφάλαιο εισάγει τον αναγνώστη σε μια συστηματική παρουσίαση των πιθανών τεχνολογικών επιλογών όπως είναι οι διαφορετικές πλατφόρμες Frontend, τα διαθέσιμα Backend frameworks και οι βάσεις δεδομένων που υποστηρίζουν τις απαιτήσεις ενός παιχνιδιού αυτής της φύσης.

Οι απαιτήσεις για τα διαδικτυακά παιχνίδια που λειτουργούν σε πραγματικό χρόνο είναι ιδιαίτερα αυξημένες. Για αυτό θα μελετηθούν τεχνολογίες που εξασφαλίζουν χαμηλή καθυστέρηση (latency) στις κινήσεις των παικτών και στον συγχρονισμό της κατάστασης του παιχνιδιού για όλους τους παίκτες.

Στο τέλος του κεφαλαίου, θα γίνει μια σύντομη αναφορά στην καταληκτική επιλογή τεχνολογιών βάσει των αναγκών του «Sindopoly».

2.2 Υπάρχουσες υλοποιήσεις του «Monopoly»

Η ταχεία ανάπτυξη της τεχνολογίας έχει επιτρέψει την ψηφιακή μετάβαση πολλών επιτραπέζιων παιχνιδιών με πρωταρχικό παράδειγμα τη «Μονόπολη», η οποία διαθέτει επίσημες εκδόσεις για ποικίλες πλατφόρμες. Η εταιρεία «Ubisoft» έχει δημιουργήσει διάφορες εκδοχές του «Monopoly», την απλή «Monopoly» (standard edition) [2], την «Monopoly Plus» για υπολογιστές και κονσόλες όπως «PlayStation» και «Xbox» [3], η οποία προσφέρει κινούμενα γραφικά και online multiplayer, και αρκετές ακόμα εκδοχές της. Παράλληλα στον χώρο των κινητών συσκευών συναντάμε την επίσημη εφαρμογή «Monopoly Mobile» από την εταιρεία «Marmalade Game Studio» [4].

Πέρα όμως από τις εμπορικές εκδόσεις υπάρχουν και αμέτρητα παραδείγματα ανοιχτού κώδικα (open-source) στο «GitHub» που επιχειρούν να αναδημιουργήσουν τη «Monopoly» ή να προτείνουν νέες παραλλαγές της. Ορισμένα από αυτά ακολουθούν πιστά τους κανόνες του κλασικού παιχνιδιού, ενώ άλλα πειραματίζονται με νέους μηχανισμούς και εμπειρίες. Για παράδειγμα, μπορεί κανείς να εντοπίσει αποθετήρια (repositories) όπως το «monopoly» από τον χρήστη «geeeeeeeeek» [5], από το έργο μάλιστα πάρθηκε έμπνευση για μερικά κομμάτια του «Sindopoly» όσον αφορά τα γραφικά της, ή το «monopoly» του χρήστη «intrepidcoder» [6].

Όλα τα παραπάνω παραδείγματα υλοποίησης του «Monopoly» πετυχαίνουν σε πολύ μεγάλο, αν όχι στο μέγιστο, βαθμό αυτό που επιχειρούν να επιτύχουν, δηλαδή την υλοποίηση μέρους ή ολόκληρου του επιτραπέζιου παιχνιδιού «Monopoly». Η διαφορά του «Sindopoly» όμως σε σχέση με όλα τα παραπάνω παραδείγματα είναι το θεματικό του στοιχείο. Υπάρχουν πολλές εκδοχές του «Monopoly», και σε επιτραπέζια φυσική μορφή και σε ψηφιακή μορφή, με θέμα διάφορες χώρες, ιστορίες βγαλμένες από βιβλία ή ταινίες, κλπ., ακόμα και για διάφορα Πανεπιστήμια και Κολλέγια από όλο τον κόσμο, αλλά μια εκδοχή με θέμα το ΔΙΠΑΕ δεν υπήρξε ως τώρα.

2.3 Τεχνολογίες που διερευνήθηκαν

2.3.1 Τεχνολογίες Front-end

Οι περισσότεροι προγραμματιστές ταυτίζουν την ανάπτυξη διεπαφών με εργαλεία όπως η React ή η Angular. Ωστόσο, όταν πρόκειται για 3D γραφικά και πιο σύνθετες οπτικοποιήσεις η πρόκληση αυξάνεται σημαντικά.

Σε τέτοιες περιπτώσεις η React JS σε συνδυασμό με τη βιβλιοθήκη ThreeJS προσφέρει μια ευέλικτη λύση. Από την άλλη η Unity, μια μηχανή παιχνιδιών (game engine) αποτελεί ένα από τα πιο δημοφιλή και ολοκληρωμένα περιβάλλοντα ανάπτυξης παιχνιδιών με τη δυνατότητα δημιουργίας builds σε WebGL, και αποτελεί και αυτή μια αξιόπιστη λύση.

Το έργο δεν θα μπορούσε να υλοποιηθεί το ίδιο εύκολα και γρήγορα χωρίς την χρήση βιβλιοθηκών και frameworks. Η επιλογή της native JavaScript δεν ήταν δυνατή, καθώς δεν προσφέρει όλα τα πλεονεκτήματα των Front-end frameworks, και θα καθυστερούσε εξαιρετικά το έργο.

Τα JavaScript frameworks και οι βιβλιοθήκες που τα συνοδεύουν προσφέρουν πολυάριθμα components, τα οποία μπορούν να αξιοποιήσουν οι προγραμματιστές, αποφεύγοντας έτσι να γράψουν από την αρχή όλο τους τον κώδικα. Ταυτόχρονα εγγυώνται μια βέλτιστη απόδοση των διαδικτυακών εφαρμογών με διάφορες σημαντικές λειτουργίες, κάποιες από τις οποίες θα αναλυθούν σε αυτή την ενότητα. Επομένως τα JavaScript frameworks, ή ακόμα και οι μηχανές παιχνιδιών (game engine) όπως η Unity στη συγκεκριμένη περίπτωση, είναι η προτιμότερη λύση από τη native JavaScript.

Στην ενότητα αυτή αναλύονται τόσο τα βασικά χαρακτηριστικά και τα πλεονεκτήματα και μειονεκτήματα των τριών παραπάνω τεχνολογιών όσο και οι δυνατότητές τους.

2.3.1.1 ReactJS & ThreeJS

Η React αποτελεί μία βιβλιοθήκη ανοιχτού κώδικα γραμμένη σε JavaScript. Αυτή η βιβλιοθήκη επιτρέπει την ανάπτυξη δυναμικών και επαναχρησιμοποιήσιμων διεπαφών χρήστη και χαρακτηρίζεται από την αρχιτεκτονική της βασισμένη σε components, τον Δηλωτικό τρόπο προγραμματισμού (declarative programming) και τη χρήση του Virtual DOM. Η εταιρεία «Facebook» (αργότερα «Meta») τη δημιούργησε το 2011 και τον Μάιο του 2013 έγινε επίσημα διαθέσιμη σε ολόκληρη την κοινότητα προγραμματιστών. Η ευρεία κοινότητα προγραμματιστών γύρω από τη React έχει δημιουργήσει έναν τεράστιο όγκο βιβλιοθηκών, εργαλείων και παραδειγμάτων, κάτι που βοηθάει στον γρήγορο εντοπισμό λύσεων σε προβλήματα και αυξημένη δυνατότητα προσαρμογής [7].

Η ιδέα πίσω από το React JS είναι να χωρίσουμε τον ιστότοπό μας σε πολλά ξεχωριστά μικρά μέρη, τα οποία αναφέρουμε ως στοιχεία (components). Κάθε στοιχείο ουσιαστικά λειτουργεί ακριβώς όπως μια συνάρτηση, δηλαδή λαμβάνει ορισμένες εισόδους γνωστές ως props, κάνει εσωτερικούς υπολογισμούς και επιστρέφει ένα στοιχείο DOM [8].

Κάτι ακόμα που προσφέρει η React JS, όπως και πολλά άλλα JavaScript Frameworks, είναι ο διαχωρισμός κώδικα ή η πράξη φόρτωσης κώδικα JavaScript σε πολλά μικρά κομμάτια και όχι όλα ταυτόχρονα. Αυτή η τεχνική, γνωστή και ως «Lazy Loading», επιτρέπει στο πρόγραμμα περιήγησης να φορτώνει μόνο το περιεχόμενο που χρειάζεται εκείνη τη στιγμή για να λειτουργήσει σωστά [8].

Επίσης ένα ακόμα κρίσιμο συστατικό της React JS είναι το Virtual DOM, πίσω από το οποίο υπάρχει μια κινητήρια δύναμη, και αυτή είναι η βελτίωση της αποτελεσματικότητας των τροποποιήσεων διεπαφής χρήστη σε εφαρμογές web. Μια εικονική αναπαράσταση του τυπικού DOM ονομάζεται Virtual DOM. Επειδή είναι πιο γρήγορο, το Virtual DOM ενημερώνεται πρώτα κάθε φορά που αλλάζει

η διεπαφή χρήστη, και ύστερα, το Virtual DOM συγκρίνεται με το κανονικό DOM για να γίνουν οι λιγότερες εφικτές αλλαγές στο κανονικό DOM. Αυτή η διαδικασία πραγματοποιείται επειδή, σε αντίθεση με το Virtual DOM, το οποίο κάνει όλες τις αλλαγές εικονικά στη μνήμη του προγράμματος περιήγησης, οι παραδοσιακές τροποποιήσεις DOM είναι πολύ δαπανηρές διότι κάθε στοιχείο πρέπει να γίνει ξανά render στην οθόνη [8].

Η ThreeJS από την άλλη είναι μία από τις πιο γνωστές βιβλιοθήκες για την απεικόνιση τρισδιάστατων γραφικών στο περιβάλλον των προγραμμάτων περιήγησης, επειδή αξιοποιεί το WebGL API [9]. Κυκλοφόρησε για πρώτη φορά από τον «Ricardo Cabello» στο «GitHub» τον Απρίλιο του 2010 [10]. Προσφέρει μια ανώτερη βαθμίδα αφαίρεσης (degree of abstraction) σε σχέση με το αμιγές WebGL, το οποίο επιτρέπει τους προγραμματιστές να σχεδιάζουν και να χειρίζονται μοντέλα, υλικά, φώτα, κάμερες και άλλα τρισδιάστατα αντικείμενα με σχετικά απλό κώδικα.

Τα πλεονεκτήματα αυτών των δυο βιβλιοθηκών είναι:

- Η React έχει μια μεγάλη κοινότητα και οικοσύστημα με τεράστιο αριθμό χρηστών, tutorial, npm πακέτων και έτοιμων components που επιταχύνουν τη διαδικασία ανάπτυξης λογισμικού. Η ThreeJS επίσης βρίσκεται πολλά χρόνια σε εξέλιξη και έχει συγκεντρώσει αξιόλογη υποστήριξη από την κοινότητά της.
- Η ύπαρξη εξειδικευμένων βιβλιοθηκών όπως το React Three Fiber διευκολύνει τον χειρισμό σύνθετων 3D στοιχείων εντός ενός React project.
- Οι εφαρμογές μπορούν να τρέχουν σε browser περιβάλλον με λειτουργίες caching. Αυτό επιτρέπει την offline λειτουργία ή γρήγορη φόρτωση σε επόμενες επισκέψεις.

Τα μειονεκτήματα αυτών των δυο βιβλιοθηκών είναι:

- Ενώ η React είναι ιδανική για μοντέρνες ιστοσελίδες και κάνει τη διαδικασία ανάπτυξής τους ευκολότερη, η χρήση της ThreeJS απαιτεί επιπλέον γνώσεις γύρω από τρισδιάστατες γραφικές έννοιες (κάμερες, υλικά, μετασχηματισμοί, κ.λπ.) με αποτέλεσμα να μεγαλώνει δραστικά η καμπύλη μάθησης των τεχνολογιών του έργου.
- Εάν μια σκηνή στη ThreeJS γίνει πολύ σύνθετη μπορεί να υπάρξουν προβλήματα στην απόδοση (frame rate drops) σε πιο αδύναμα συστήματα, επομένως ο έλεγχος βέλτιστων πρακτικών (π.χ. μείωση πολυγώνων, σωστή χρήση lights/shadows) είναι απαραίτητος. Επίσης, αν και στην σημερινή εποχή έχει βελτιωθεί πολύ η κατάσταση, ακόμα υπάρχουν συστήματα που δεν υποστηρίζουν το WebGL ή και να το υποστηρίζουν, δεν έχουν επαρκείς υπολογιστικούς πόρους για να υποστηρίξουν σύνθετες σκηνές.

Παρόλο που η ThreeJS καλύπτει τις περισσότερες ανάγκες, σε ακραίες περιπτώσεις βελτιστοποίησης ίσως χρειαστεί ο προγραμματιστής να γράψει απευθείας WebGL shader code. Άλλος ένα λόγος που το έργο γίνεται ακόμα πιο περίπλοκο και επομένως μεγαλώνει ξανά η καμπύλη μάθησης.

2.3.1.2 Unity Real-Time Development Platform

Η Unity είναι μία κορυφαία πλατφόρμα για την ανάπτυξη παιχνιδιών. Προσφέρει ένα ολοκληρωμένο περιβάλλον IDE εν ονόματι Unity Editor, το οποίο περιλαμβάνει γραφικά εργαλεία, components φυσικής (physics engine), animation controllers, σύστημα prefab κ.ά. [11]. Ενώ ξεκίνησε ως μια πλατφόρμα για ανάπτυξη παιχνιδιών σε υπολογιστές και κονσόλες εξελίχθηκε γρήγορα ώστε να υποστηρίζει mobile, VR/AR και WebGL. Το γεγονός ότι χρησιμοποιεί τη γλώσσα C# για τον ορισμό

της λογικής του παιχνιδιού την καθιστά προσιτή σε προγραμματιστές που έχουν ήδη εμπειρία με το .NET οικοσύστημα, το οποίο αναφέρεται αργότερα σε αυτό το κεφάλαιο.

Ένα από τα βασικά πλεονεκτήματα της Unity είναι η πληθώρα resources, tutorials και επίσημων πακέτων (assets) που μπορούν να καλύψουν σχεδόν κάθε πτυχή ενός παιχνιδιού. Το Unity Store έχει χιλιάδες πακέτα (σκηνές, 3D models, συστήματα σωματιδίων). Έτσι επιτρέπει τη γρήγορη συναρμολόγηση ενός παιχνιδιού χωρίς να χρειάζεται ο δημιουργός να αναπτύξει τα πάντα από το μηδέν.

Η Unity προσφέρει τη δυνατότητα δημιουργίας WebGL builds. Ουσιαστικά, με τη χρήση του Unity Editor μπορεί κανείς να μετατρέψει το παιχνίδι που έφτιαξε σε εφαρμογή JavaScript για έναν web browser χρησιμοποιώντας HTML5/JavaScript, WebAssembly, το WebGL API κλπ. [12]. Αυτό σημαίνει ότι ένα 3D παιχνίδι που αναπτύχθηκε στο Unity για τον υπολογιστή ή για κονσόλες μπορεί με μερικές ρυθμίσεις να εξαχθεί και σε μορφή που μπορεί να υπάρξει στο Web. Ωστόσο η διαδικασία του building σε WebGL μπορεί να αυξήσει το μέγεθος του παραγόμενου πακέτου (bundle) και επομένως να επιβαρύνει τους χρόνους φόρτωσης. Σε σύγκριση με τη λύση React + ThreeJS όπου το τελικό payload μπορεί να είναι αισθητά ελαφρύτερο (ανάλογα βέβαια με το πόσα assets και εξωτερικές βιβλιοθήκες χρησιμοποιούνται) το Unity WebGL build συχνά είναι μεγαλύτερο [13], [14].

Σε επίπεδο πολυπλοκότητας η Unity απαιτεί από τον δημιουργό να προσαρμοστεί σε μια συγκεκριμένη ροή εργασίας (Unity Editor, Scenes, GameObjects, Prefabs, κ.λπ.). Για εφαρμογές που δεν είναι αυστηρά παιχνίδια ή που χρειάζονται υψηλό επίπεδο προσαρμογής στον browser (διαδραστικά UI στοιχεία, δυναμικά data bindings) ίσως η React + ThreeJS αποδειχθεί πιο ευέλικτη. Με άλλα λόγια στη Unity υπάρχει μια μεγαλύτερη καμπύλη μάθησης [14] και αναλόγως το project και το μέγεθος του είτε αξίζει να αφιερωθεί χρόνος για την εκπαίδευση του προγραμματιστή είτε όχι. Παρόλα αυτά αν το έργο είναι ένα αμιγώς 3D παιχνίδι με έμφαση σε μηχανισμούς φυσικής και animations τότε η Unity προσφέρει πιο εξειδικευμένα εργαλεία.

Ωστόσο, ένας βασικός περιορισμός αφορά τη φιλοσοφία ανάπτυξης. Πιο συγκεκριμένα η Unity εστιάζει παραπάνω στην παραγωγή ολοκληρωμένων 3D experiences και όχι αναγκαστικά στην ενσωμάτωση σε κάποια υπάρχουσα web πλατφόρμα. Με άλλα λόγια δεν είναι ο πρωταρχικός σκοπός της Unity η δημιουργία παιχνιδιών web και για αυτό μπορεί να υστερεί σε αυτό το κομμάτι συγκριτικά με τη React JS + ThreeJS. Επίσης η επεξεργασία των UI στοιχείων στη Unity (π.χ. φόρμες εγγραφής, dashboard στατιστικών) είναι συχνά λιγότερο εύκολη σε σύγκριση με ένα τυπικό web framework (React, Angular κ.λπ.). Η διαχείριση και η ανταπόκριση σε «κλασικά» HTTP αιτήματα και REST APIs είναι δυνατή αλλά δεν αποτελεί το βασικό πεδίο στο οποίο ειδικεύεται η Unity, η οποία προορίζεται περισσότερο για αμιγή παιχνίδια.

Τέλος, η ενσωμάτωση σε περιβάλλοντα συνεχή ανάπτυξης (Continuous Integration/Continuous Deployment) μπορεί συχνά να είναι πιο περίπλοκη σε σχέση με μια απλή web-based εφαρμογή. Επιπρόσθετα μια Unity WebGL εφαρμογή τρέχει σε μια «sandboxed» διεργασία εντός του browser, δηλαδή σε ένα περιορισμένο και κλειστό περιβάλλον όπου η αλληλεπίδραση με το DOM ή και άλλα στοιχεία είναι πιο περιορισμένη σε αντίθεση με ένα React-based project.

2.3.1.3 Συμπεράσματα & επιλογή React JS + ThreeJS

Από την παραπάνω σύγκριση γίνεται σαφές ότι και οι δύο προσεγγίσεις μπορούν να είναι επιτυχείς ανάλογα με τον τύπο του ψηφιακού παιχνιδιού ή της εφαρμογής που έχουμε ως στόχο.

Ο συνδυασμός React + ThreeJS είναι ιδανικός για έργα που απαιτούν πολλαπλά στοιχεία διεπαφής χρήστη, φόρμες, δυναμικές συνδέσεις δεδομένων και επιλεκτική παρουσίαση γραφικών 3D. Επίσης

προσφέρει ένα οικείο περιβάλλον για προγραμματιστές που κατέχουν προυπάρχουσες γνώσεις πάνω σε τεχνολογίες Web.

Το Unity προσφέρει ένα πλήρες οικοσύστημα για τη δημιουργία 3D, 2D ή μικτών παιχνιδιών. Ωστόσο το Unity μπορεί να περιλαμβάνει σημαντικά μειονεκτήματα όσον αφορά τα μεγέθη των κατασκευών και την ενσωμάτωση web πλατφόρμες.

Στο συγκεκριμένο έργο επιλέχθηκε ο συνδυασμός React JS + ThreeJS. Αυτό συνέβη για πολλαπλούς λόγους. Αρχικά έπαιξε μεγάλο ρόλο η εμπειρία του συγγραφέα της Δ.Ε. και δημιουργού του «Sindopoly» με τη JavaScript, τα frameworks της, και πιο συγκεκριμένα με τη React JS. Η ThreeJS εφόσον είναι μια βιβλιοθήκη της React JS δεν διαφέρει πολύ στον τρόπο χρήσης της και επομένως δεν απαιτεί πολλές επιπλέον γνώσεις. Επίσης η ύπαρξη μιας μεγάλης κοινότητας και εξαιρετικού documentation έπαιξαν κρίσιμο ρόλο. Τέλος οι μεγαλύτερες αποδόσεις για μικρότερα έργα όπως το συγκεκριμένο ήταν εξαιρετικά σημαντικές για την επιλογή της τεχνολογίας.

2.3.2 Τεχνολογίες Back-end

Σε αυτή την ενότητα παρουσιάζεται μια εκτενής ανασκόπηση των πιο δημοφιλών τεχνολογιών backend που μπορούν να αξιοποιηθούν για την ανάπτυξη ενός διαδικτυακού παιχνιδιού όπως το «Sindopoly». Συγκεκριμένα, αναλύονται τρεις κύριες τεχνολογίες οι οποίες αποφασίστηκε εξ' αρχής πως θα διερευνηθούν για τον σκοπό αυτής της Δ.Ε.: το Spring Boot (Java οικοσύστημα), το .NET (C#, F# κ.λπ.) και το Node.js (JavaScript/TypeScript στο server-side). Για κάθε τεχνολογία παρουσιάζεται μια σύντομη ιστορική αναδρομή, τα βασικά χαρακτηριστικά, η κοινότητα και το documentation που τη συνοδεύει. Σκοπός είναι να παρθεί από αυτή την έρευνα μια σφαιρική εικόνα για το ποια πλατφόρμα ταιριάζει καλύτερα στις απαιτήσεις ενός multiplayer παιχνιδιού, ιδίως εάν απαιτείται ταυτόχρονη εξυπηρέτηση πολλών χρηστών σε πραγματικό χρόνο.

2.3.2.1 Spring Boot

Το Spring Boot είναι μέρος του ευρύτερου οικοσυστήματος Spring, το οποίο ξεκίνησε στις αρχές της δεκαετίας του 2000. Το Spring αποτέλεσε μια ελαφριά εναλλακτική προσέγγιση στις τότε πολύπλοκες Java EE εφαρμογές. Ο κύριος στόχος και ιδέα του Spring ήταν η υιοθέτηση και αξιοποίηση σχεδιαστικών προτύπων όπως το Inversion of Control και το Dependency Injection. Με αυτό τον τρόπο επιτεύχθηκε η απλούστερη διαχείριση των εξαρτήσεων και η μεγαλύτερη ευκολία στη συντήρηση κώδικα από τους προγραμματιστές. Σταδιακά η κοινότητα του Spring πρόσθεσε πολυάριθμες βιβλιοθήκες (π.χ. τη Spring Data, τη Spring Security, τη Spring MVC, κ.λπ.) που κάλυπταν ένα ευρύ φάσμα αναγκών των εταιρικών εφαρμογών [15].

Η άφιξη του Spring Boot το 2014 έδωσε βάση στην «out-of-the-box» εμπειρία του προγραμματιστή. Με άλλα λόγια ο προγραμματιστής μπορεί να δημιουργήσει μια πλήρη εφαρμογή με όσο το δυνατόν τις λιγότερες ρυθμίσεις. Παλιότερα οι προγραμματιστές έπρεπε να διαχειρίζονται πολλαπλά αρχεία XML για τη διασύνδεση με τις βάσεις δεδομένων τους. Ως απάντηση σε αυτό το Spring Boot προσέφερε μια σειρά από «starter» πακέτα που συμπεριλαμβάνουν έτοιμες προεπιλογές και έτσι μειώνεται με αυτό τον τρόπο σημαντικά ο χρόνος που απαιτείται για να ξεκινήσει ένα νέο project. Ως αποτέλεσμα το Spring Boot έχει γίνει ιδιαίτερα δημοφιλές στους προγαμματιστές, κυρίως για τις εταιρικές εφαρμογές αλλά όχι μόνο.

Η αρχιτεκτονική REST (Representational State Transfer) έχει επικρατήσει ευρέως ως το πιο κοινό πρότυπο επικοινωνίας μεταξύ του πελάτη (client) και του διακομιστή (server). Το Spring Boot σε συνδυασμό με το Spring MVC ή το Spring WebFlux (εν ονόματι reactive approach) επιτρέπει στον

προγραμματιστή να δημιουργεί REST endpoints εξαιρετικά εύκολα [16]. Μέσα από κάποια απλά annotations τύπου @RequestMapping ορίζονται οι διαδρομές (routes) της εφαρμογής. Ταυτόχρονα τα δεδομένα ανταλλάσσονται συνήθως σε μορφή JSON.

Στα προηγούμενα πρέπει να προστεθεί ότι το Spring Cloud παρέχει επίσης εργαλεία για τη σχεδίαση και την υλοποίηση μικροϋπηρεσιών (microservices) τα οποία μπορεί να φανούν εξαιρετικά χρήσιμα στον προγραμματιστή [17]. Τα microservices προτιμώνται συχνά σε μεγάλης κλίμακας συστήματα και αυτό επειδή κάθε υπηρεσία μπορεί να αναπτύσσεται, να διανέμεται και να συντηρείται ανεξάρτητα. Για παράδειγμα θα μπορούσε να υφίσταται μια υπηρεσία για τη διαχείριση των κινήσεων των παικτών πάνω στο ταμπλό, μια δεύτερη για το σύστημα πληρωμών εντός παιχνιδιού και μια τρίτη για τον έλεγχο ταυτότητας των χρηστών και έπειτα τη διαχείριση των δικαιωμάτων τους.

Ένα ακόμα χρήσιμο στοιχείο του Spring Boot είναι πως υποστηρίζεται από μια τεράστια και ενεργή κοινότητα. Υπάρχουν επίσημα guides και tutorials στην ιστοσελίδα του Spring καθώς και αμέτρητα άρθρα, video tutorials και forum threads στο Διαδίκτυο που λύνουν κάθε λογής πρόβλημα. Επιπλέον συνεχίζουν τακτικά να δημοσιοποιούνται νέες εκδόσεις (releases) που εξασφαλίζουν ότι προστίθενται συνεχώς νέα χαρακτηριστικά και βελτιώνονται οι αποδόσεις [18].

2.3.2.2 .NET

Η πλατφόρμα .NET αρχικά παρουσιάστηκε από τη Microsoft ως .NET Framework στις αρχές της δεκαετίας του 2000 με κύριο γνώμονα την ανάπτυξη εφαρμογών Windows. Με την κυκλοφορία του .NET Core προσφέρεται πλέον πλήρης υποστήριξη για πολλαπλά λειτουργικά συστήματα (Windows, Linux, macOS), ενώ παράλληλα έγινε ανοιχτού κώδικα [19]. Η ύπαρξη της ASP.NET Core επιτρέπει την ταχεία ανάπτυξη web εφαρμογών και APIs, για αύτο και η πλατφόρμα είναι ιδανική για server-side υλοποίηση.

Η γλώσσα C# αναγνωρίζεται ως η κύρια γλώσσα προγραμματισμού στην .NET πλατφόρμα, αλλά υποστηρίζονται επίσης η F# (λειτουργικός προγραμματισμός), η Visual Basic, κ.ά. Το C# είναι ένα αντικειμενοστραφές (OOP) εργαλείο, το οποίο συνδυάζει δομή από γλώσσες όπως C++ και Java. Συγκεκριμένα, χαρακτηρίζεται από στατική τυποποίηση, υψηλή αναγνωσιμότητα και πληθώρα σύγχρονων χαρακτηριστικών (LINQ, async/await, pattern matching).

Η απόδοση του .NET Runtime θεωρείται εξαιρετική, ειδικά για διακομιστές (server) που πρέπει να χειριστούν πολλαπλές ταυτόχρονες (concurrent) αιτήσεις. Η μεταγλώττιση Just-In-Time (JIT) σε συνδυασμό με τη βελτιστοποίηση του κώδικα προσφέρει σε πολλές περιπτώσεις αποτελέσματα συγκρίσιμα με γλώσσες όπως η Java ή και C++ και σε συγκεκριμένα σενάρια η απόδοση μπορεί να είναι ακόμα και καλύτερη [20]. Βεβαίως, η πραγματική απόδοση εξαρτάται από τις αλγοριθμικές επιλογές του προγραμματιστή, τη δομή των βάσεων δεδομένων και τον τρόπο διαχείρισης της μνήμης του συστήματος.

Η καμπύλη εκμάθησης της .NET πλατφόρμας θεωρείται μέτρια προς εύκολη για όσους έχουν ήδη εμπειρία σε αντικειμενοστραφείς γλώσσες (π.χ. C++ ή Java). Το Visual Studio και το Visual Studio Code (διάσημα IDEs - ολοκληρωμένα περιβάλλοντα ανάπτυξης) προσφέρουν ολοκλήρωση κώδικα (IntelliSense), εργαλεία debugging και εντοπισμό σφαλμάτων.

Η κοινότητα γύρω από το .NET είναι μεγάλη και προσφέρει ένα ολοκληρωμένο documentation από τη Microsoft και εκατοντάδες βιβλιοθήκες και frameworks ανοιχτού κώδικα (NuGet packages) για πιο εξειδικευμένες λειτουργίες. Σε αντίθεση με το παρελθόν η στροφή της Microsoft στο open source έχει διευκολύνει τη διάδοση και υιοθέτηση του .NET Core/5+ σε πολλούς οργανισμούς και φορείς.

2.3.2.3 Node.js

Το Node.js εμφανίστηκε γύρω στο 2009 και μετέτρεψε την JavaScript σε γλώσσα ικανή να τρέχει στον διακομιστή και όχι μόνο στον φυλλομετρητή [21]. Η καινοτομία που έφερε το Node.js ήταν ένας μηχανισμός ασύγχρονης διαχείρισης I/O, με τον οποίο διευκολύνθηκε η ανάπτυξη εφαρμογών που απαιτούν υψηλό βαθμό ταυτόχρονης επεξεργασίας (concurrency) με ελάχιστο κόστος σε πόρους. Δεν χρησιμοποιεί threads για κάθε αίτηση, αλλά βασίζεται σε ένα event loop όπου οι κλήσεις που σταματούν την εφαρμογή (blocking calls) αποφεύγονται ή εκτελούνται ασύγχρονα με αποτέλεσμα να διατηρείται η ταχύτητα της εφαρμογής [22].

Ένα από τα μεγαλύτερα πλεονεκτήματα του Node.js είναι η ύπαρξη του npm (Node Package Manager) που φιλοξενεί μία από τις μεγαλύτερες συλλογές βιβλιοθηκών ανοιχτού κώδικα στον κόσμο [22]. Πολύ γνωστές εταιρείες όπως η Netflix, η Uber και η LinkedIn (και όχι μόνο) έχουν δηλώσει ότι χρησιμοποιούν Node.js σε κρίσιμα τμήματα των υποδομών τους [23].

Επιπλέον ένα εξαιρετικά σημαντικό πλεονέκτημα είναι ότι χρησιμοποιείται η γλώσσα JavaScript και εδώ όπως και στο front-end. Χάρη σε αυτή οι ομάδες ανάπτυξης μπορούν να επαναχρησιμοποιούν λογική ή μοντέλα δεδομένων. Για παράδειγμα ο ίδιος κώδικας επαλήθευσης (validation) ή τύπων (types, αν χρησιμοποιηθεί η TypeScript) μπορεί να εκτελείται τόσο στον client όσο και στον server με αποτέλεσμα να μειωθούν τα σφάλματα που εμφανίζονται από ασυμβατότητες.

Ειδικά σε multiplayer παιχνίδια η ανάγκη για επικοινωνία σε πραγματικό χρόνο είναι κρίσιμη. Το Node.js προσφέρει ενσωματωμένους μηχανισμούς (π.χ. το http και net module) για τη δημιουργία TCP/HTTP servers. Ωστόσο το μεγάλο πλεονέκτημα είναι η ύπαρξη εξειδικευμένων βιβλιοθηκών όπως το Socket.io που διαχειρίζεται τις WebSocket συνδέσεις [24].

Το Node.js θεωρείται εξαιρετικά αποδοτικό σε αυτές τις χρήσεις, δεδομένου πως η ασύγχρονη φύση του επιτρέπει την εξυπηρέτηση χιλιάδων ταυτόχρονων συνδέσεων χωρίς να παραμένει σε ένα κλασικό threading μοντέλο. Παρά ταύτα σε CPU-intensive εργασίες όπως επεξεργασία εικόνων ή περίπλοκοι αλγόριθμοι το Node.js δεν είναι τόσο αποδοτικό καθώς ο event loop μπορεί να επιβραδυνθεί. Σε τέτοιες περιπτώσεις γίνεται χρήση workers μέσω των Worker Pool threads, τα οποία διαχειρίζεται μια άλλη βιβλιοθήκη και όχι ο προγραμματιστής [25].

2.3.2.4 Κριτήρια επιλογής

Η υλοποίηση ενός διαδικτυακού παιχνιδιού όπως το «Sindopoly» περιλαμβάνει συγκεκριμένες και σημαντικές απαιτήσεις.

Αρχικά τα πιόνια και οι τοποθεσίες τους πρέπει να ενημερώνονται ακαριαία σε πραγματικό χρόνο προκειμένου όλοι οι παίκτες να βλέπουν την τρέχουσα κατάσταση όπως πρέπει να είναι. Αυτό σημαίνει ότι το back-end καλείται να διαχειριστεί μεγάλο αριθμό χρηστών που ενδεχομένως επιχειρούν να τροποποιήσουν τα ίδια δεδομένα (π.χ. την ίδια ιδιοκτησία) ταυτόχρονα. Οπότε η αρχιτεκτονική του συστήματος πρέπει να προβλέπει μηχανισμούς για την αποφυγή πιθανών αντικρουόμενων συναλλαγών των χρηστών και τη διασφάλιση της ακεραιότητας της πληροφορίας.

Παράλληλα η δυνατότητα κλιμάκωσης (scaling) αρχίζει να αποκτά σημασία σιγά σιγά όταν ο αριθμός των παρτίδων αυξάνεται, αφού πρέπει να μπορούμε να προσθέτουμε περισσότερα instances του server χωρίς να διακόπτεται η παρεχόμενη υπηρεσία. Και εδώ δηλαδή παίζει ρόλο η αρχιτεκτονική του συστήματος και ο τρόπος με τον οποίο έχτισε τον κώδικα ο προγραμματιστής.

Σημαντικό επίσης είναι το ενδεχόμενο συνεχούς επέκτασης του παιχνιδιού καθώς νέες λειτουργίες μπορεί να υιοθετηθούν στο μέλλον. Επομένως η αρχιτεκτονική που θα έχει το παιχνίδι οφείλει να είναι

δομημένη με τέτοιο τρόπο που να επιτρέπει την προσθήκη τέτοιων λειτουργιών και χαρακτηριστικών χωρίς δραστικές αλλαγές.

Τέλος όσον αφορά την ασφάλεια, πέρα από την προστασία των λογαριασμών των παικτών το οποίο είναι από τα πιο βασικά κομμάτια της υλοποίησης ενός συστήματος σε αυτή την εποχή, η επαλήθευση της ταυτότητας ενός χρήστη (authentication) και ο έλεγχος των αδειών πρόσβασης στα δεδομένα πριν από οποιαδήποτε ενέργεια παίζουν πρωτεύοντα ρόλο.

Βάσει όλων των προαναφερθέντων σε αυτή την ενότητα αλλά και σε ολόκληρο το κεφάλαιο, το Spring Boot, το .NET και το Node.js πληρούν σε γενικές γραμμές τις βασικές προϋποθέσεις για την ανάπτυξη ενός ασφαλούς, επεκτάσιμου και αποδοτικού συστήματος back-end για το «Sindopoly».

Η τελική επιλογή ωστόσο εξαρτάται σε μεγάλο βαθμό από τη συνολική εμπειρία και τις προτιμήσεις του προγραμματιστή ή της ομάδας ανάπτυξης, καθώς και από το πώς σκοπεύουν αυτοί να αναπτύξουν το παιχνίδι και αν σκοπεύουν να συνεχίσουν τη διατήρησή του μακροπρόθεσμα.

2.3.2.5 Συμπεράσματα & επιλογή .NET

Λαμβάνοντας υπόψη τις απαιτήσεις ενός διαδικτυακού παιχνιδιού όπως το «Sindopoly» είναι φανερό ότι τόσο το Spring Boot όσο και το .NET και το Node.js αποτελούν εξαιρετικές επιλογές για την ανάπτυξη του back-end του. Καθεμία από αυτές τις τεχνολογίες διαθέτει τα δικά της πλεονεκτήματα και μειονεκτήματα, χωρίς όμως να υπάρχει κάποια πτυχή που να την καθιστά αναντικατάστατη ή να αποκλείει τις υπόλοιπες. Οι τεχνικές δυνατότητες, η απόδοση και η κλιμάκωση είναι σε παρόμοια επίπεδα, ειδικά όταν ο κώδικας αναπτυχθεί σωστά και υπάρξει κατάλληλη αρχιτεκτονική.

Από πρακτική σκοπιά η τελική επιλογή εξαρτάται κυρίως από την εμπειρία και την προτίμηση του δημιουργού ή της ομάδας που θα αναπτύξει το παιχνίδι. Αν για παράδειγμα οι προγραμματιστές έχουν εντρυφήσει στον κόσμο της Java, το Spring Boot θα τους προσφέρει ένα τεράστιο οικοσύστημα βιβλιοθηκών και εξαιρετικά αναλυτικό documentation. Αντίστοιχα σε μια ομάδα με γνώσεις σε C# θα ήταν πιο λογικό να επιλέξει το ASP.NET Core. Έτσι και αν υπάρχει βαριά εμπειρία σε JavaScript όπως είναι το φυσιολογικό στους περισσότερους προγραμματιστές Web που έχουν ασχοληθεί με front-end, το Node.js προσφέρει την ευκαιρία να μην χρειαστεί ο προγραμματιστής να μάθει νέα γλώσσα προγραμματισμού εξ' ολοκλήρου από την αρχή.

Στην παρούσα Δ.Ε. επιλέχθηκε το Spring Boot κυρίως χάρη στην ευρεία αποδοχή του στον χώρο των web APIs και τη μεγάλη διαθεσιμότητα πόρων εκμάθησης. Ταυτόχρονα ταιριάζει στις προτιμήσεις του δημιουργού. Η επιλογή αυτή δεν σημαίνει ότι τα άλλα δύο τεχνολογικά οικοσυστήματα υστερούν για το «Sindopoly», αλλά απλώς ότι το Spring Boot εξυπηρετεί πιο άμεσα τους στόχους και τις γνώσεις που ήδη υπάρχουν στο υπόβαθρο.

2.3.3 Τεχνολογίες Database

Παρόλο που η βάση δεδομένων (database) συνήθως εντάσσεται στο πλαίσιο το back-end, κρίθηκε αναγκαίο να δημιουργηθεί ένα κεφάλαιο εξ' ολοκλήρου για τη διερεύνηση και ανάλυση των τεχνολογιών που την αφορούν.

Οι βάσεις δεδομένων αποτελούν πολλές φορές έναν από τους πιο κρίσιμους παράγοντες επιτυχίας καθώς αποτελούν τα θεμέλια όλης της εφαρμογής. Στο πλαίσιο ενός διαδικτυακού παιχνιδιού όπως το «Sindopoly» η επιλογή τεχνολογιών για τη βάση δεδομένων μπορεί να παίξει σημαντικό ρόλο στην ταχύτητα ανταπόκρισης της εφαρμογής και γενικότερα της ενημέρωσης της κατάστασης, στην

αξιοπιστία του συστήματος όσον αφορά τις συναλλαγές και ενέργειες των παικτών καθώς και στην υποστήριξη ενός τεράστιου αριθμού παικτών συγχρόνως.

Σε αυτή την ενότητα θα εξεταστούν τρεις βασικές τεχνολογίες βάσεων δεδομένων: η MariaDB, η PostgreSQL και το Redis. Θα γίνει αναφορά στο ιστορικό, στα χαρακτηριστικά και στις ιδιαιτερότητες της καθεμίας καθώς και στο γιατί και πώς μπορούν να χρησιμοποιηθούν σε εφαρμογές σαν το «Sindopoly». Τέλος, θα γίνει μνεία στο γεγονός ότι ορισμένες από αυτές τις τεχνολογίες μπορούν να συνδυαστούν μεταξύ τους ώστε να αξιοποιηθούν τα πλεονεκτήματα της καθεμίας ταυτοχρόνως.

2.3.3.1 MariaDB

Η MariaDB αποτελεί μια διακλάδωση (fork) του MySQL, ενός από τα πλέον δημοφιλή Συστήματα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS). Ουσιαστικά η ιστορία της MariaDB ξεκινά όταν η MySQL εξαγοράστηκε από την Oracle. Τότε ο Michael “Monty” Widenius, ένας από τους βασικούς συνιδρυτές της MySQL, αποφάσισε να δημιουργήσει ένα ανοιχτού κώδικα παρακλάδι με σκοπό να προσφέρει στην κοινότητα θα μια εναλλακτική, πλήρως ανοιχτή και συμβατή έκδοση του MySQL [26]. Η αρχική φιλοσοφία ήταν ότι όλα τα θετικά στοιχεία του MySQL θα συνεχίσουν να εξελίσσονται χωρίς τους πιθανούς περιορισμούς που θα έθετε η Oracle. Έκτοτε η MariaDB όχι μόνο διατηρεί υψηλή συμβατότητα με τη MySQL, αλλά έχει προσθέσει και νέες λειτουργίες και χαρακτηριστικά.

Σε επίπεδο χαρακτηριστικών η MariaDB υιοθετεί πλήρως τη δομή των βάσεων δεδομένων τύπου MySQL. Για παράδειγμα, υποστηρίζει και αυτή SQL ερωτήματα, triggers, stored procedures, views κ.λπ.. Παράλληλα ενσωματώνει νέες μηχανές αποθήκευσης (όπως InnoDB, Aria και άλλες) που συνεισφέρουν στην απόδοση και την ανθεκτικότητα στα σφάλματα [27]. Αυτές οι μηχανές διακρίνονται από διαφορετικές φιλοσοφίες, π.χ. η InnoDB στηρίζεται σε συναλλαγές (ACID compliance) με υποστήριξη Foreign Keys και row-level locking, ενώ άλλες μηχανές στοχεύουν σε ταχύτερη ανάγνωση ή σε άλλα εξειδικευμένα σενάρια.

Η απόδοση της MariaDB στις περισσότερες περιπτώσεις είναι συγκρίσιμη ή και ανώτερη από αυτήν της MySQL ιδίως σε αναβαθμισμένες εκδόσεις που εκμεταλλεύονται βελτιώσεις στο replication ή σε συγκεκριμένες μηχανές αποθήκευσης [28]. Για παράδειγμα, μία από τις μεθόδους αντιγραφής δεδομένων (replication) που ξεχωρίζουν είναι η βελτιωμένη δυνατότητα parallel replication χάρη στην οποία μπορεί να υπάρξει ταυτόχρονη μεταφορά δεδομένων σε πολλαπλούς slave servers.

Η MariaDB έχει αναδειχθεί σε μια από τις πιο δημοφιλείς επιλογές για σχεσιακές βάσεις δεδομένων τόσο σε μικρά όσο και σε μεγάλα έργα. Αξιοποιείται από πλήθος οργανισμών, από startups έως μεγάλες εταιρείες στο διαδίκτυο (hosting providers) ή σε εφαρμογές ηλεκτρονικού εμπορίου [29]. Το γεγονός ότι βασίζεται στην ίδια φιλοσοφία με το MySQL επιτρέπει τη χρήση των ίδιων εργαλείων διαχείρισης όπως το phpMyAdmin ή το MySQL Workbench, καθώς και την ενσωμάτωση σε ήδη υπάρχοντα περιβάλλοντα. Επιπλέον, υπάρχουν πολλαπλές διανομές Linux (Ubuntu, Debian, CentOS κ.λπ.) που προσφέρουν πακέτα εγκατάστασης της MariaDB με ελάχιστες προσαρμογές.

Τέλος, το documentation της MariaDB είναι εκτενές τόσο σε επίσημο επίπεδο όσο και από την κοινότητα σε φόρουμ και blogs. Πλέον η ενεργή κοινότητα προγραμματιστών πίσω από τη MariaDB παρέχει συχνές ενημερώσεις, επιδιορθώσεις σφαλμάτων και νέες εκδόσεις.

2.3.3.2 PostgreSQL

Η PostgreSQL αποκαλείται από μερικούς ως «η πιο προηγμένη σχεσιακή βάση δεδομένων ανοιχτού κώδικα» [30]. Η ιστορία της ξεκινά τη δεκαετία του 1980 από την ομάδα του Πανεπιστημίου Berkeley

[31]. Σκοπός της ομάδας ήταν να δημιουργήσει μια νέα γενιά Συστημάτων Διαχείρισης Βάσεων Δεδομένων που θα βασιζόντουσαν στο σχεσιακό μοντέλο και θα παρείχαν τις ευκολίες που απαιτούνται από εφαρμογές μηχανικής για τις οποίες ένα συμβατικό σχεσιακό σύστημα δεν είναι κατάλληλο [32].

Ένα από τα βασικά χαρακτηριστικά που την καθιστούν δελεαστική είναι η ενσωμάτωση προηγμένων δυνατοτήτων όπως το JSONB (βέλτιστη αποθήκευση και αναζήτηση JSON) και τα window functions που επιτρέπουν την εκτέλεση σύνθετων αναλυτικών ερωτημάτων (analytics) σε πραγματικό χρόνο [33], [34]. Χάρη στο JSONB η PostgreSQL μπορεί να χρησιμοποιηθεί ως μια υβριδική βάση δεδομένων ικανή να χειρίζεται τόσο δομημένα (relations) όσο και ημιδομημένα δεδομένα (documents) με εξαιρετική ταχύτητα και ευελιξία. Τα window functions από την άλλη δίνουν τη δυνατότητα για υπολογισμούς, ομαδοποιήσεις και στατιστικά εντός της ίδιας δήλωσης SQL χωρίς να απαιτείται πολλαπλή επεξεργασία σε επίπεδο εφαρμογής.

Επιπλέον, η PostgreSQL είναι γνωστή για την ορθότητα των συναλλαγών της (ACID compliance), την ανθεκτικότητα σε σφάλματα (fault tolerance) μέσω της χρήσης WAL (Write-Ahead Logging) καθώς και τη δυνατότητα εύκολης δημιουργίας αντιγράφων ασφαλείας και cluster. Προσφέρει ακόμα δυνατότητες ιεραρχικών ερωτημάτων (CTE - Common Table Expressions), partitioning σε μεγάλους πίνακες, και ευρεία γκάμα τύπων δεδομένων (π.χ. array, hstore, range types) [35].

Για τις ανάγκες ενός παιχνιδιού όπως το «Sindopoly», όπου μπορεί να υπάρχουν περιπτώσεις που χρειάζονται σύνθετες στατιστικές, ειδικότερα στο μέλλον κατά την κλιμάκωση του έργου, η PostgreSQL προσφέρει ισχυρά εργαλεία. Ωστόσο, σε περιπτώσεις όπου η εφαρμογή βασίζεται σε πιο απλά queries, η απόφαση μπορεί να επηρεαστεί κυρίως από την υπάρχουσα γνώση του προγραμματιστή ή της ομάδας ανάπτυξης.

2.3.3.3 Redis

Το Redis συγκαταλέγεται στις in-memory βάσεις δεδομένων. Χρησιμοποιείται ευρέως για την προσωρινή αποθήκευση (caching) και την υποστήριξη real-time εφαρμογών [36]. Σε αντίθεση με μια κλασική σχεσιακή βάση, το Redis αποθηκεύει τα δεδομένα του βασικά στη μνήμη RAM (αν και υπάρχουν επιλογές που επιτρέπουν τη μετατροπή ή τη συγχρονισμένη εγγραφή σε δίσκο). Αυτή η προσέγγιση καθιστά το Redis ταχύτατο στις λειτουργίες ανάγνωσης και εγγραφής, το οποίο είναι ιδανικό για σενάρια όπου παίζει ρόλο η χαμηλή καθυστέρηση (latency) [37].

Ο τρόπος οργάνωσης δεδομένων στο Redis δεν είναι ο κλασικός πίνακας (table) που συναντάμε στις SQL βάσεις. Αντίθετα, βασίζεται σε δομές δεδομένων όπως τα key-value pairs, τις λίστες (lists), τα σύνολα (sets), τα ταξινομημένα σύνολα (sorted sets) και τα hash. Μέσα από αυτές τις δομές προσφέρεται μεγάλη ευελιξία στην οργάνωση της πληροφορίας [37].

Η χρήση του Redis σε multiplayer εφαρμογές έχει να κάνει με την ανάγκη για αμεσότητα και μικρές καθυστερήσεις στη διεκπεραίωση ενεργειών. Στο «Sindopoly» για παράδειγμα κάθε φορά που ένας παίκτης κινεί το πιόνι του, ρίχνει ζάρια ή γενικότερα πραγματοποιείται οποιαδήποτε άλλη ενέργεια κατά τη διάρκεια του παιχνιδιού, τα δεδομένα μπορούν να αποθηκεύονται προσωρινά στο Redis. Με αυτόν τον τρόπο οι επακόλουθες αιτήσεις των άλλων παικτών μπορούν να αντλούν πληροφορίες απευθείας από τη μνήμη, χωρίς να χρειάζεται να προσπελαστεί η κύρια SQL ή NoSQL βάση, η οποία είναι πιο «βαριά». Και αυτό ώστε να διασφαλιστεί η ταχύτητα των αιτήσεων, και επομένως η ταχύτητα ανταπόκρισης του παιχνιδιού.

Ωστόσο, το Redis έχει και ορισμένους περιορισμούς. Αρχικά η λειτουργία του βασίζεται κυρίως στη μνήμη RAM. Αυτό συνήθως σημαίνει υψηλότερο κόστος σε επίπεδο υποδομής, ιδίως αν ο όγκος

δεδομένων είναι μεγάλος. Επίσης, αν σταματήσει εντελώς η λειτουργία του server για κάποιο χρονικό διάστημα, τα δεδομένα που είναι αποθηκευμένα στο Redis, και επομένως στη μνήμη RAM, θα χαθούν για πάντα. Επιπλέον, παρότι υπάρχουν μέθοδοι παραμονής των δεδομένων (persistence) σε δίσκο, αυτές ενδέχεται να μην είναι τόσο πλήρεις όσο σε μια σχεσιακή ή NoSQL βάση (π.χ. MariaDB, PostgreSQL, MongoDB). Επομένως, το Redis δεν είναι η ενδεδειγμένη επιλογή για την κύρια καταγραφή των κρίσιμων δεδομένων, αλλά για μια καταγραφή των δεδομένων που έχουν τη μεγαλύτερη ζήτηση από τους πελάτες (clients) που θα συνοδεύει μια παραδοσιακή βάση δεδομένων.

Παρ' όλα αυτά αν χρησιμοποιηθεί ορθά στο πλαίσιο που του αρμόζει, το Redis μπορεί να ενισχύσει δραματικά την απόδοση μιας εφαρμογής. Ειδικά σε παιχνίδια όπου η προσωρινή κατάσταση αλλάζει συχνά.

2.3.3.4 Συμπεράσματα και Επιλογή MariaDB + Redis

Η πραγματικότητα είναι ότι οι τεχνολογίες βάσεων δεδομένων σπάνια αξιοποιούνται αυτόνομα σε σύγχρονα έργα, δεδομένου πως κάθε λύση έχει συγκεκριμένα προτερήματα που καλύπτουν διαφορετικές ανάγκες.

Σε διαδικτυακές εφαρμογές με μεγάλες απαιτήσεις, όπως και στη συγκεκριμένη περίπτωση, υπάρχει η ανάγκη για την αντοχή σε μεγάλο φορτίο, την ακεραιότητα των πιο κρίσιμων συναλλαγών, καθώς και την ταχεία ενημέρωση της κατάστασης της εφαρμογής και τη χαμηλή καθυστέρηση. Έτσι, η επιλογή μιας κύριας SQL βάσης δεδομένων σε συνδυασμό με μια in-memory λύση αποτελεί μια ευρέως διαδεδομένη και αποδοτική πρακτική.

Η MariaDB ξεχωρίζει για τη σταθερότητά της και τον εκτενή κύκλο υποστήριξης που διαθέτει. Η υψηλή απόδοσή της την καθιστά κατάλληλη για εφαρμογές που απαιτούν κλασική διαχείριση συναλλαγών (ACID) και σύνθετα ερωτήματα SQL. Από την πλευρά της η PostgreSQL προσφέρει ένα εύρος προηγμένων δυνατοτήτων, όπως και έγινε αναφορά σε αυτό το κεφάλαιο, και διαπρέπει σε σύνθετες υλοποιήσεις με περίπλοκα αναλυτικά ερωτήματα.

Επομένως, μπορεί κανείς να καταλήξει ξανά στο συμπέρασμα πως η επιλογή της τεχνολογίας έχει να κάνει με τις προτιμήσεις του δημιουργού, καθώς και οι δυο τεχνολογίες μπορούν να υποστηρίξουν το συγκεκριμένο έργο. Στην προκειμένη περίπτωση επιλέχθηκε η MariaDB για την απλότητά της.

Ωστόσο, για εφαρμογές που απαιτούν άμεση απόκριση και διαρκή ενημέρωση του περιβάλλοντος, η χρήση μιας in-memory βάσης όπως το Redis καθίσταται καθοριστική. Σε εφαρμογές όπως το «Sindopoly» όπου η κατάσταση του παιχνιδιού αλλάζει διαρκώς, το Redis αναλαμβάνει να αποθηκεύει προσωρινά τα πιο σημαντικά δεδομένα, ώστε να υπάρχει πάντα άμεση ανταπόκριση στους παίκτες. Για αυτό και επιλέχθηκε και το Redis για να χρησιμοποιηθεί στην ανάπτυξη του «Sindopoly».

Αυτή η στρατηγική διαμοιρασμού των αρμοδιοτήτων παρέχει τα καλύτερα από δύο «κόσμους». Η MariaDB διατηρεί με ασφάλεια τα κρίσιμα και «μακροπρόθεσμα» δεδομένα (π.χ. το προφίλ των παικτών, τα ιστορικά των παρτίδων, κλπ.), ενώ το Redis διατηρεί τα «βραχυπρόθεσμα» δεδομένα (π.χ. την κατάσταση του παιχνιδιού, που βρίσκονται οι παίκτες, ποιες ιδιοκτησίες έχουν αγορασμένες, κλπ.). Και σε περίπτωση που σταματήσει η λειτουργία του server και χαθούν τα δεδομένα που είναι αποθηκευμένα στο Redis, απλά γίνεται ο συμβιβασμός πως οι παίκτες, όταν επανέλθει η λειτουργία του server, δε θα συνεχίζουν τις παρτίδες τους, καθώς δε θα έχει σημασία πλέον, αλλά θα ξεκινούν καινούργιες.

2.3.4 Επιπλέον Τεχνολογίες

Παρότι ο βασικός άξονας σύγκρισης στο συγκεκριμένο κεφάλαιο επικεντρώνεται σε back-end, front-end και databases, η επιτυχία ενός σύγχρονου multiplayer παιχνιδιού εξαρτάται και από ένα σύνολο επιπλέον τεχνολογιών που διαπερνούν αυτά τα επίπεδα. Αυτές οι τεχνολογίες υποστηρίζουν είτε τη ροή δεδομένων, είτε τη συνεργατική τεκμηρίωση, είτε και την οπτικοποίηση. Παρακάτω αναλύονται οι επιπλέον τεχνολογίες που χρησιμοποιήθηκαν κατά τη σχεδίαση και υλοποίηση του συστήματος.

2.3.4.1 WebSockets

Τα WebSockets είναι μια τεχνολογία που είναι πολύ σημαντική για εφαρμογές όπως multiplayer παιχνίδια, συστήματα συνομιλίας, πλατφόρμες χρηματοοικονομικών δεδομένων, κλπ., καθώς είναι απαραίτητα στην εποχή μας για την αμφίδρομη επικοινωνία κομματιών μιας εφαρμογής σε πραγματικό χρόνο. Δεν μπορούν να κατηγοριοποιηθούν αναγκαστικά σε μία από τις κατηγορίες Back-end και Front-end, καθώς είναι πιο πολύ μια λειτουργία στο επίπεδο μεταφοράς (Transport Layer) που εκτείνεται και στα δυο επίπεδα. Για αυτό λοιπών και αναλύονται στην συγκεκριμένη υποενότητα [38].

Τα WebSockets διαφέρουν από άλλες τεχνικές, όπως το long-polling ή το SSE (Server-Sent Events), καθώς επιτρέπουν στον διακομιστή να ωθεί δεδομένα με το που αλλάξει κάτι χωρίς να περιμένει αίτημα από τον πελάτη, το οποίο είναι αυτό που συμβαίνει υπό κανονικές συνθήκες. Επομένως, είναι ιδανική τεχνολογία για το «Sindopoly», όπου η κάθε κίνηση κάποιου παίκτη πρέπει να εμφανίζεται άμεσα σε όλους τους άλλους παίκτες [39].

Στη συγκεκριμένη περίπτωση που χρησιμοποιείται η Spring Boot, μπορεί να χρησιμοποιηθεί το SockJS, προκειμένου να αξιοποιηθούν οι μηχανισμοί επανασύνδεσης του παίκτη στο παιχνίδι, το οποίο συμβαίνει συνήθως εξαιτίας της σύνδεσής του στο Internet, αλλά και άλλων μηχανισμών και features.

Ωστόσο, αξίζει να σημειωθεί και να τονιστεί ότι τα WebSockets δεν προορίζονται για μεταφορές πολύ μεγάλων αρχείων ή για διάφορες λειτουργίες που ταιριάζουν καλύτερα σε REST APIs. Αντιθέτως, μπορούν να λειτουργήσουν συμπληρωματικά. Έτσι, στο «Sindopoly», μπορεί να χρησιμοποιηθεί μια REST διαδρομή για να χειρίζεται τη δημιουργία των παιχνιδιών, την εγγραφή των παικτών, τα ιστορικά δεδομένα, και άλλων βασικών λειτουργιών. Και από την άλλη, μπορεί να υπάρχει ένα WebSocket κανάλι που θα μεταδίδει άμεσα τα γεγονότα του ίδιου του παιχνιδιού, ώστε να παραμένουν οι παίκτες πλήρως συγχρονισμένοι μεταξύ τους.

2.3.4.2 Docker

Το Docker είναι μια πλατφόρμα λογισμικού που διευκολύνει την ταχεία ανάπτυξη, δοκιμή και «deployment» προγραμμάτων. Το λογισμικό «συσκευάζεται» από το Docker σε τυποποιημένες μονάδες που ονομάζονται κοντέινερ (containers), τα οποία περιέχουν όλες τις απαραίτητες βιβλιοθήκες, τα εργαλεία συστήματος και τον κώδικα για τη λειτουργία του προγράμματος [40].

Η υιοθέτηση του Docker αποτέλεσε κεντρική τεχνολογική απόφαση, καθώς επέτρεψε στο Sindopoly να διανέμεται και να εκτελείται σε πλήρως ελεγχόμενο περιβάλλον, ανεξάρτητο από βιβλιοθήκες ή ρυθμίσεις του υποκείμενου λειτουργικού συστήματος. Αντί κάθε χρήστης ή κόμβος παραγωγής να εγκαθιστά Java, Node, MariaDB και Redis ξεχωριστά, όλα τα συστατικά «πακετάρονται» σε απομονωμένα containers τα οποία ξεκινούν με μια μόνη εντολή.

Το Docker Compose επεκτείνει αυτήν την ιδέα, επιτρέποντας να περιγραφούν πολλά κοντέινερ, όπως το back-end, το front-end, η βάση δεδομένων MariaDB και το Redis, σε ένα και μόνο αρχείο «docker-compose.yml» [41]. Έτσι, με την εντολή «docker compose up», το Sindopoly εκκινεί ολόκληρο το stack

του σε λίγα δευτερόλεπτα. Με αυτό τον τρόπο, το ίδιο περιβάλλον αναπαράγεται πιστά σε οποιοδήποτε υπολογιστή ή μηχάνημα, απλοποιώντας την εγκατάσταση και ελαχιστοποιώντας τις περιπτώσεις να λειτουργεί σε ένα μηχάνημα και όχι σε άλλο. Επομένως για την εγκατάσταση του Sindopoly σε οποιοδήποτε server θα χρησιμοποιηθεί το Docker Compose.

2.3.4.3 SwaggerUI

Το Swagger UI είναι ένα ανοικτό περιβάλλον τεκμηρίωσης (documentation) που μετατρέπει το πρότυπο OpenAPI (πρώην Swagger) σε διαδραστική σελίδα HTML. Κάθε endpoint εμφανίζεται με την περιγραφή του, τα ζητούμενα πεδία και τα πιθανά σφάλματα, ενώ ο χρήστης μπορεί να δοκιμάζει live κλήσεις χωρίς να εγκαθιστά εξωτερικό εργαλείο [42].

Στο Sindopoly η βιβλιοθήκη springdoc-openapi παράγει αυτόματα το αρχείο προδιαγραφής από τις σημειώσεις του Spring Boot. Με μία μόνο εξάρτηση Maven και χωρίς χειροκίνητη γραφή YAML, εκτίθεται στο /swagger-ui.html μία πλήρης «κονσόλα» όπου ο προγραμματιστής, ο tester ή ακόμη και ο διαχειριστής του server βλέπει:

- τα REST endpoints όλου του συστήματος
- τα σχήματα JSON (DTO) με παραδείγματα request/response
- δυνατότητα αποστολής δοκιμαστικού αιτήματος και λήψης της πραγματικής απάντησης του server.

Η ενσωμάτωση του Swagger UI επιταχύνει τόσο την ανάπτυξη όσο και την υποστήριξη, καθώς οι νέοι συνεργάτες κατανοούν άμεσα το API του συστήματος και τα endpoints του, μπορούν να πειραματιστούν χωρίς curl, ενώ ο διδάσκων επαληθεύει γρήγορα αν ένα endpoint λειτουργεί μετά από την αλλαγή του. Με άλλα λόγια, «λύνει» τα χέρια του προγραμματιστή, παρουσιάζοντας του πολλά κομμάτια του κώδικα, που υπό κανονικές συνθήκες θα έπρεπε να ψάξει χειροκίνητα μέσα σε όλο το codebase.

2.3.4.4 Draw.io

Το draw.io αποτελεί το κύριο εργαλείο σχεδίασης που χρησιμοποιήθηκε για τα διαγράμματα του Sindopoly, όπως τα C4 και το ER διάγραμμα. Λειτουργεί εξολοκλήρου μέσα από τον browser, επιτρέποντας αποθήκευση τόσο τοπικά όσο και σε υπηρεσίες cloud όπως το Google Drive ή το OneDrive, χωρίς να απαιτείται εγκατάσταση στον υπολογιστή (αν και υπάρχει αυτή η δυνατότητα). Η βιβλιοθήκη προτύπων του περιλαμβάνει έτοιμα σχήματα για προσαρμόσιμα UML διαγράμματα. Μοναδικό πλεονέκτημα είναι η δυνατότητα εξαγωγής σε PNG/SVG με ενσωματωμένο XML, ώστε ένα διάγραμμα να εμφανίζεται στο report ως εικόνα και ταυτόχρονα να διατηρεί το «source» του για μελλοντική επεξεργασία [43].

2.3.4.5 Mermaid.js

Η Mermaid.js είναι βιβλιοθήκη που μετατρέπει δηλώσεις κειμένου (ή κώδικα σε γενικές γραμμές) απευθείας σε διαγράμματα. Για το Sindopoly χρησιμοποιείται κυρίως στο διάγραμμα κλάσεων και τις ροές της εφαρμογής. Δέχεται απλό κώδικα τύπου, όπως φαίνεται στον Κώδικας Α.

Κώδικας Α: Mermaid.js Demo

```
graph TD; Player -->|plays| Game
```

Όπως και το draw.io, λειτουργεί κι αυτή εξολοκλήρου μέσα από τον browser, μέσα από ειδικό editor [44].

2.3.5 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε μια εκτενής βιβλιογραφική επισκόπηση και διερεύνηση κάποιων βασικών τεχνολογιών που θα μπορούσαν να αξιοποιηθούν για την ανάπτυξη του «Sindopoly».

Για το Frontend εξετάστηκαν η React σε συνδυασμό με τη βιβλιοθήκη ThreeJS και η Unity. Αποτελούν δύο διαφορετικές προσεγγίσεις για την υλοποίηση 3D γραφικών και διαδραστικών περιβαλλόντων στο Web. Η React σε συνδυασμό με το ThreeJS προσφέρει μια πιο ευέλικτη και ελαφριά λύση για web εφαρμογές. Από την άλλη η Unity αποτελεί μια ολοκληρωμένη πλατφόρμα για τη δημιουργία 3D παιχνιδιών και παρέχει ένα υψηλό επίπεδο εργαλείων δημιουργίας παιχνιδιών αλλά και μεγαλύτερες απαιτήσεις για την ενσωμάτωση τους σε περιβάλλον Web.

Για το Backend εξετάστηκε το Spring Boot από το οικοσύστημα της Java, το .NET και το Node.js. Καθεμία από αυτές τις επιλογές εμφανίζει κάποια πλεονεκτήματα όπως η ευρεία υιοθέτηση του Spring Boot, οι επιδόσεις και το εκτενές οικοσύστημα του .NET, και η ασύγχρονη αρχιτεκτονική καθώς και η χρήση της γλώσσας JavaScript του Node.js. Σε εκείνο το σημείο ξεκαθαρίστηκε ότι η τελική απόφαση εξαρτάται από τις ανάγκες της εκάστοτε ομάδας ανάπτυξης, όπως πάντα.

Για το Database εξετάστηκε η MariaDB, η PostgreSQL και το Redis. Η MariaDB έχει να προσφέρει για την εφαρμογή καλή συμβατότητα, ταχύτητα και σταθερότητα. Η PostgreSQL ενδείκνυται συνήθως για πιο πολύπλοκα συστήματα που απαιτούν τη δυνατότητα επεκτασιμότητάς τους. Το Redis από την άλλη μπορεί να χρησιμοποιηθεί για να αποθηκευτούν προσωρινά δεδομένα που χρειάζονται μονάχα κατά τη διάρκεια του παιχνιδιού. Σε αυτό το σημείο αναφέρθηκε ξανά πως η τελική απόφαση εξαρτάται από τις προτιμήσεις και τις ανάγκες του προγραμματιστή.

Στο τέλος επιλέχθηκαν τα React + ThreeJS για το Frontend, το Spring Boot για το Backend και η συνδυαστική χρήση της MariaDB + Redis για το Database. Επίσης, επιλέχθηκαν επιπλέον τεχνολογίες όπως τα WebSockets, προκειμένου να υποστηρίξουν περαιτέρω την ανάπτυξη και την ορθή λειτουργία του συγκεκριμένου έργου.

Όσον αφορά τα διαγράμματα που δημιουργήθηκαν, και όχι το σύστημα αυτό καθαυτό, χρησιμοποιήθηκαν οι τεχνολογίες draw.io και mermaid.js, η κάθε μία σε διαφορετική περίπτωση, ανάλογα τα πλεονεκτήματα που προσφέρει.

Στο επόμενο κεφάλαιο θα εξεταστεί η αρχιτεκτονική του «Sindopoly» σε βάθος. Θα αναλυθούν οι επιμέρους διεργασίες της υλοποίησης, οι μηχανισμούς επικοινωνίας μεταξύ Frontend και Backend καθώς και οι λεπτομέρειες στη διαχείριση της βάσης δεδομένων.

Κεφάλαιο 3ο: Σχεδιασμός & Αρχιτεκτονική Συστήματος

3.1 Εισαγωγή

Το συγκεκριμένο κεφάλαιο εστιάζει στον σχεδιασμό και την αρχιτεκτονική του «Sindopoly». Πλέον γίνεται η μετάβαση από τη θεωρητική διερεύνηση τεχνολογιών που πραγματοποιήθηκε στο Κεφάλαιο 2, στην ανάλυση της όλης αρχιτεκτονικής του συστήματος που υποστηρίζει το «Sindopoly».

Στο προηγούμενο κεφάλαιο τεκμηριώθηκαν όλοι οι λόγοι, για τους οποίους επιλέχθηκε η Spring Boot, η MariaDB, το Redis, και τα WebSockets, καθώς και οι εναλλακτικές τους, οι οποίες απορρίφθηκαν. Η παρούσα ενότητα αναλαμβάνει να δείξει πώς αυτές οι τεχνολογίες πλαισιώνονται σε μια συνεκτική λογική υποδομή.

3.2 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις (Functional Requirements, ή FR) ορίζουν τι πρέπει να υλοποιεί το «Sindopoly». Η παρούσα ενότητα ταξινομεί τις βασικές λειτουργίες του Sindopoly κατά βαθμό αναγκαιότητας ακολουθώντας τη μέθοδο MoSCoW (Must / Should / Could / Would) [45]. Έτσι διευκρινίζεται ποιες δυνατότητες είναι αδιαπραγμάτευτες για την έναρξη παραγωγικής λειτουργίας και ποιες μπορούν να υλοποιηθούν σταδιακά σε επόμενα sprint. Στον Πίνακας 3.1 παρουσιάζονται οι πιο βασικές λειτουργικές απαιτήσεις του συστήματος.

Πίνακας 3.1: Λειτουργικές Απαιτήσεις τύπου Must Have

Κωδικός	Λειτουργική Απαίτηση
FR-001	Το σύστημα πρέπει να επιτρέπει σε εξουσιοδοτημένο χρήστη τη δημιουργία νέας παρτίδας (lobby) και να τη διατηρεί ενεργή για προκαθορισμένο χρονικό διάστημα.
FR-002	Το σύστημα πρέπει να μπορεί να εκκινεί μια παρτίδα μόνο όταν έχει συμπληρωθεί ο ελάχιστος αριθμός παικτών και να ενημερώνει όλους τους συμμετέχοντες ότι το παιχνίδι ξεκίνησε.
FR-003	Το σύστημα πρέπει να αποτρέπει την εκκίνηση ή διαγραφή παρτίδας από μη εξουσιοδοτημένους χρήστες (όσοι δεν είναι «owners» του lobby), επιστρέφοντας κατάλληλο μήνυμα σφάλματος.
FR-004	Το σύστημα πρέπει να καταγράφει έναν παίκτη όταν συνδέεται ή επανασυνδέεται, χωρίς να δημιουργεί διπλές εγγραφές.
FR-005	Το σύστημα πρέπει να εκτελεί τον γύρο ενός παίκτη, συμπεριλαμβανομένης της ρίψης ζαριών, της μετακίνησης του πιονιού, της εφαρμογής των κανόνων του τετραγώνου και της κοινοποίησης του αποτελέσματος σε όλους τους παίκτες, με ένα και μοναδικό αίτημα χρήστη.
FR-006	Το σύστημα πρέπει να επιτρέπει στον παίκτη να αποκτά ιδιοκτησίες εφόσον διαθέτει επαρκούς πόρους,

	εξασφαλίζοντας ότι η συναλλαγή ολοκληρώνεται ατομικά και χωρίς race conditions (πχ. ταυτόχρονες αγορές).
FR-007	Το σύστημα πρέπει να χειρίζεται τη διαδικασία φυλάκισης και απελευθέρωσης παίκτη, είτε μέσω του περάσματος γύρων είτε μέσω πληρωμής προστίμου.
FR-008	Το σύστημα πρέπει να επισημαίνει παίκτη ως χρεοκοπημένο μόλις το χρηματικό του υπόλοιπο γίνει αρνητικό, αφαιρώντας του τη δυνατότητα να συνεχίζει να παίζει.
FR-009	Το σύστημα πρέπει να τερματίζει αυτόματα την παρτίδα όταν παραμένει ένας ή κανένας ενεργός παίκτης και να αναδεικνύει τον νικητή βάσει προκαθορισμένων κανόνων.

Με τον χαρακτηρισμό «Must Have» υποδηλώνεται πως η συγκεκριμένη λειτουργική απαίτηση είναι αδιαπραγμάτευτη για να θεωρηθεί το σύστημα πλήρως λειτουργικό, διότι χωρίς αυτήν το σύστημα θεωρείται ατελές και πως δεν μπορεί να γίνει χρήση του. Επομένως, οι λειτουργίες που βρίσκονται στον Πίνακας 3.1 έχουν υλοποιηθεί για το «Sindopoly».

Πίνακας 3.2: Λειτουργικές Απαιτήσεις τύπου Should Have

Κωδικός	Λειτουργική Απαίτηση
FR-010	Το σύστημα θα έπρεπε να παρακάμπτει αυτόματα παίκτες που καθυστερούν υπερβολικά στο γύρο τους, ενημερώνοντας όλους τους συμμετέχοντες για το γεγονός.
FR-011	Το σύστημα θα έπρεπε να επιτρέπει σε παίκτη που έχασε προσωρινά τη σύνδεση να επανενταχθεί στην παρτίδα χωρίς την απώλεια της προόδου μες το παιχνίδι.

Με τον χαρακτηρισμό «Should Have» υποδηλώνεται πως η συγκεκριμένη λειτουργική απαίτηση, αν και αποτελεί σημαντικό κομμάτι του συστήματος, δεν είναι απαραίτητο να υλοποιηθεί για την ομαλή λειτουργία του, διότι χωρίς αυτήν το σύστημα μπορεί να λειτουργήσει ομαλά και να πραγματοποιήσει τις επιχειρησιακές ανάγκες του, αλλά δεν μπορεί να θεωρηθεί πρακτική η χρήση του. Επομένως, οι λειτουργίες που βρίσκονται στον Πίνακας 3.2 έχουν υλοποιηθεί για το «Sindopoly».

Πίνακας 3.3: Λειτουργικές Απαιτήσεις τύπου Could Have

Κωδικός	Λειτουργική Απαίτηση

FR-012	Το σύστημα θα μπορούσε να διαθέτει πίνακα κορυφαίων παικτών (leaderboard) βασισμένο σε μετρήσεις όπως νίκες ή συνολικά κέρδη.
FR-013	Το σύστημα θα μπορούσε να διατηρεί επαρκές απόθεμα τυχαίων τιμών για τις ρίψεις ζαριών, αναπληρώνοντάς το προληπτικά ώστε να αποφεύγονται καθυστερήσεις, αντί να δημιουργεί τυχαίες τιμές την στιγμή που τις χρειάζεται.
FR-014	Το σύστημα θα μπορούσε να αξιοποιεί μεταβλητές του περιβάλλοντός του ή εξωτερικά API με σκοπό να πετύχει το «true randomness» στη ρίψη των ζαριών.

Με τον χαρακτηρισμό «Could Have» υποδηλώνεται πως η συγκεκριμένη λειτουργική απαίτηση θα ήταν ιδανικό να υλοποιηθεί, αλλά δεν είναι απαραίτητο να υλοποιηθεί για την ομαλή λειτουργία του συστήματος, διότι χωρίς αυτήν το σύστημα μπορεί να χρησιμοποιηθεί κανονικά, και αυτή απλά βοηθάει απλά άμα υπάρχει. Οι λειτουργίες που βρίσκονται στον Πίνακας 3.3 έχουν επίσης υλοποιηθεί για το «Sindopoly».

Πίνακας 3.4: Λειτουργικές Απαιτήσεις τύπου Would Have

Κωδικός	Λειτουργική Απαίτηση
FR-015	Το σύστημα θα έπρεπε να καταγράφει τον χρόνο ολοκλήρωσης κάθε γύρου ώστε να μπορεί να υπολογίζει στατιστικές απόδοσης (π.χ. P95 latency).
FR-016	Το σύστημα θα μπορούσε να επιτρέπει σε θεατές να παρακολουθούν μια δημόσια παρτίδα σε πραγματικό χρόνο χωρίς δυνατότητα παρέμβασης.
FR-017	Το σύστημα θα μπορούσε να προσφέρει δυνατότητα λήψης αρχείου καταγραφής (log) των ενεργειών μιας ολοκληρωμένης παρτίδας.
FR-018	Το σύστημα θα μπορούσε να επιτρέπει προηγμένα φίλτρα και εναλλακτικά κριτήρια ταξινόμησης στον πίνακα των κορυφαίων παικτών.

Με τον χαρακτηρισμό «Would Have» υποδηλώνεται πως η συγκεκριμένη λειτουργική απαίτηση δεν είναι απαραίτητο να υλοποιηθεί για την ομαλή λειτουργία του συστήματος, ούτε του προσφέρει κάποια σημαντική αξία, και επομένως γίνεται και να μην υλοποιηθεί ή να υλοποιηθεί σε κάποια αργότερη αναβάθμιση του συστήματος στο μέλλον. Επομένως, οι λειτουργίες που βρίσκονται στον Πίνακας 3.4 δεν έχουν υλοποιηθεί για το «Sindopoly», αλλά μπορεί να υλοποιηθούν στο μέλλον.

3.3 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις (Non Functional Requirements, ή NFR) καθορίζουν πόσο καλά πρέπει να εκτελούνται οι λειτουργίες του «Sindopoly», θέτοντας μετρήσιμα όρια σε επιδόσεις, αξιοπιστία,

ασφάλεια και επεκτασιμότητα. Δεν έχουν να κάνουν με κάποια ενέργεια του συστήματος ή του χρήστη, αλλά αφορούν περιορισμούς, τους οποίους πρέπει να τηρεί το σύστημα, προκειμένου να ανταποκριθεί στις επιχειρησιακές ανάγκες του έργου, έτσι όπως έχουν τεθεί από τον «πελάτη». Είναι κι αυτές εξ ίσου σημαντικές και για αυτό απαιτείται προσοχή στην επιλογή τους.

Οι μη λειτουργικές απαιτήσεις αποτελούν τον οδηγό πλοήγησης για τις κρίσιμες τεχνολογικές και αρχιτεκτονικές επιλογές κάθε έργου. Εάν καθοριστούν εσφαλμένα, ή παραμείνουν ασαφείς, το λογισμικό κινδυνεύει να εξελιχθεί σε λύση που, ακόμη και αν «τρέχει», αδυνατεί να ικανοποιήσει τις πραγματικές επιχειρησιακές ανάγκες.

Για αυτό κάθε NFR πρέπει να πηγάζει απευθείας από τους στόχους του συστήματος, να διατυπώνεται με σαφή και μετρήσιμα κριτήρια, και να θέτει ρεαλιστικά όρια. Μόνο έτσι οι ομάδες αρχιτεκτονικής και προγραμματισμού μπορούν να αξιολογήσουν αντικειμενικά εναλλακτικές λύσεις (π.χ. in-memory cache vs SQL replica.

Στην πράξη, για παράδειγμα, η «24/7» διαθεσιμότητα χωρίς προϋποθέσεις είναι περισσότερο ευσεβής πόθος παρά μια ρεαλιστική μη λειτουργική απαίτηση. Αντίθετα, μια απαίτηση όπως «99,5 % uptime ανά μήνα» μεταφράζεται σε συγκεκριμένο όριο downtime (~3,5 ώρες) και είναι κάτι πολύ πιο ρεαλιστικό, έτσι ώστε το τελικό προϊόν δεν θα εκπλήσσει δυσάρεστα κανέναν, ούτε τους χρήστες, ούτε τους προγραμματιστές.

Παρουσιάζονται κατά κατηγορία οι πιο βασικές μη λειτουργικές απαιτήσεις:

- **Επιδόσεις:** Για ένα multiplayer παιχνίδι όπως το «Sindopoly» ο χρόνος από το πάτημα του κουμπιού «Ρίξε τα ζάρια» μέχρι να ενημερωθεί το ταμπλό όλων των παικτών είναι ύψιστης σημασίας. Ουσιαστικά, αυτή η ενέργεια πρέπει να πραγματοποιείται στιγμιαία. Για αυτό λοιπόν υπάρχει η ανάγκη για μια μνήμη που έχει υψηλές ταχύτητες για ενέργειες read και write, όπως η RAM, όπου τα δεδομένα αναγκαστικά αποθηκεύονται προσωρινά. Η βάση όπου τα δεδομένα θα αποθηκεύονται μόνιμα μπορεί να ενημερώνεται ασύγχρονα, πιθανώς και ποτέ, καθώς τα στοιχεία ενός παιχνιδιού δεν έχουν κάποια αξία μετά το τέλος του παιχνιδιού.
- **Ασφάλεια:** Τα περισσότερα δεδομένα που κρατάει το «Sindopoly» δεν είναι αναγκαστικά ευαίσθητα, πχ. το όνομα του παίκτη μες το σύστημα (nickname), οι κινήσεις του σε ένα παιχνίδι, τα στατιστικά του που ούτως ή άλλως είναι δημόσια και ελεύθερα προσβάσιμα μέσω του ίδιου του συστήματος. Από την άλλη, υπάρχουν και δεδομένα που είναι ευαίσθητα, όπως στοιχεία που χρησιμοποιεί ο χρήστης για να συνδεθεί, πχ. κωδικός, email, κλπ.. Ασχέτως όμως της ευαισθησίας των δεδομένων, το σύστημα οφείλει να διασφαλίζει, πρώτα την ακεραιότητά τους αν ανήκουν στην πρώτη κατηγορία, και την ιδιωτικότητά τους μαζί με την ακεραιότητά τους αν ανήκουν στη δεύτερη κατηγορία. Η διαρροή τους ή, ακόμα χειρότερα, η τροποποίησή τους από κάποιον επιτιθέμενο θα έφερνε σε δύσκολη θέση αρχικά τον ίδιο τον χρήστη, και έπειτα θα έπληττε τη φήμη της πλατφόρμας και θα αποθάρρυνε τη χρήση της.
- **Επεκτασιμότητα:** Με την πάροδο του χρόνο η ζήτηση του «Sindopoly» είναι πολύ πιθανό να εκτοξευτεί. Ήδη η ιδέα του «Sindopoly» είναι γνωστή ανάμεσα σε πολλούς φοιτητές του ΔΙΠΑΕ. Η αρχιτεκτονική του συστήματος οφείλει να μπορεί να υποστηρίξει την μελλοντική αύξηση των χρηστών και τη γενικότερη αναβάθμιση του παιχνιδιού, χωρίς την πολυπλοκότητα που συνοδεύεται συνήθως από μια τέτοια κίνηση. Εφόσον ξεκινήσει να αποκτά μεγαλύτερη φήμη το παιχνίδι, και νέους χρήστες από άλλα τμήματα, μερικά αντικείμενα θα χρειαστεί να ενημερωθούν. Για παράδειγμα, οι κάρτες απόφασης και εντολής ως τώρα αφορούν τους φοιτητές του ΜΠΗΣ, αλλά καλό θα ήταν πχ. να υπάρχουν πολλές «τράπουλες» ανάμεσα στις

οποίες μπορεί να επιλέξει ο χρήστης. Για παράδειγμα, ένας φοιτητής του τμήματος Γεωπονίας θα εκτιμούσε αν το περιεχόμενο των καρτών απόφασης και εντολής είχε να κάνουν με το τμήμα του, τους καθηγητές του, συμβάντα που έχει ζήσει, κλπ.. Το σύστημα οφείλει να μπορεί να υποστηρίξει και αυτή την αναβάθμιση, χωρίς να χρειάζεται να γραφτούν κομμάτια του συστήματος, ή και ολόκληρο το σύστημα, ξανά από την αρχή.

3.4 Περιορισμοί

Κάθε έργο λογισμικού υλοποιείται μέσα σε συγκεκριμένα πλαίσια. Οι περιορισμοί αυτοί, αν και συχνά εκλαμβάνονται ως εμπόδια, κατευθύνουν τους αρχιτέκτονες προς πιο συγκεντρωμένες, ρεαλιστικές λύσεις. Παρακάτω παρουσιάζονται οι σημαντικότεροι περιορισμοί που επηρέασαν τον σχεδιασμό και την ανάπτυξη του «Sindopoly», καθώς και ο τρόπος με τον οποίο καθόρισαν τεχνολογικές επιλογές και προτεραιότητες.

3.4.1 Χρονικός ορίζοντας

Για την υλοποίηση του έργου αφιερώθηκε στην ουσία ένα ακαδημαϊκό εξάμηνο. Ως εκ τούτου, έγινε επικέντρωση στις «Must Have» και «Should Have» λειτουργικές απαιτήσεις καθώς και μονάχα σε πολύ βασικούς κανόνες της «Μονόπολη», προκειμένου να υπάρξει κάτι ελαφρώς παραπάνω από ένα MVP (Minimum Viable Product), δηλαδή μια έκδοση ενός προϊόντος με αρκετά χαρακτηριστικά ώστε να είναι χρησιμοποιήσιμη από τους πρώτους πελάτες.

3.4.2 Προϋπολογισμός

Δεν υπήρχε δυνατότητα συνδρομής σε εμπορικές cloud υπηρεσίες ή επί πληρωμή βιβλιοθήκες παιχνιδιών. Για αυτό λοιπόν επιλέχθηκαν ισχυρές δωρεάν τεχνολογίες με μακροχρόνια υποστήριξη (Java/Spring Boot, MariaDB, Redis, React), ώστε να είναι σίγουρο πως τυχόν μελλοντικοί συντηρητές θα μπορούν να βρουν τεκμηρίωση και παραδείγματα χωρίς κόστος για την αδειοδότηση.

3.4.3 Περιορισμένοι Πόροι

Το σύστημα καλείται να τρέχει σε ένα μέρος ενός διαθέσιμου φυσικού server του Τμήματος, για το οποίο δεν ήταν γνωστοί εξ' αρχής οι πόροι. Αυτό οδήγησε στην ανάγκη να υλοποιηθεί η ενδιάμεση αποθήκευση της κατάστασης του παιχνιδιού αποκλειστικά στη μνήμη RAM με τη βοήθεια του εργαλείου Redis, ώστε να περιοριστεί η κίνηση στον σκληρό δίσκο, αλλά και σε όριο μεγέθους ανά παρτίδα για να στηριχθούν έως και εκατοντάδες ταυτόχρονες παρτίδες.

3.4.4 Διαδικτυακές Ιδιαιτερότητες

Οι φοιτητές συχνά βρίσκονται πίσω από firewalls/NAT του Πανεπιστημίου. Επομένως, ήταν αναγκαία η χρήση WebSocket over HTTPS (port 443) με SockJS fallback. Η αρχιτεκτονική απέφυγε custom TCP sockets ή UDP multicast που θα απαιτούσαν να ανοίξουν νέα ports, το οποίο είναι επιπλέον δουλειά χωρίς κάποιο όφελος.

3.4.5 Εξειδίκευση

Ο δημιουργός διέθετε προυπάρχουσα εμπειρία σε Java, JavaScript, στο ReactJS framework, και στη MariaDB. Ως αποτέλεσμα απορρίφθηκε η εναλλακτική ενός πλήρους game-engine (π.χ. Unity Multiplayer) ώστε να αποφευχθεί η απότομη καμπύλη μάθησης που θα υπήρχε αναγκαστικά και δεν ταίριαζε στον διαθέσιμο χρόνο.

3.4.6 Συμπέρασμα

Οι παραπάνω περιορισμοί, αντί να παρεμποδίσουν την πρόοδο, λειτούργησαν ως κατευθυντήριες γραμμές. Δηλαδή, συμπύκνωσαν το εύρος του έργου, επέβαλαν λιτές και δοκιμασμένες τεχνολογίες και ενθάρρυναν αρχιτεκτονικές αποφάσεις που υποστηρίζουν τη μελλοντική επεκτασιμότητα χωρίς εξάρτηση από ακριβές υποδομές.

3.5 Θεμελιώδεις Αρχές

Η παρούσα ενότητα συνοψίζει τις θεμελιώδεις αρχές που κατεύθυναν τον σχεδιασμό και την υλοποίηση του Sindopoly. Στόχος της είναι να προσφέρει σε μελλοντικούς συντηρητές ένα «κοινό λεξιλόγιο» καλών πρακτικών. Έτσι, ακόμη κι όταν απαιτηθούν νέες λειτουργίες, οι αλλαγές θα ευθυγραμμίζονται με το αρχικό πνεύμα του συστήματος. Κάποιες αρχές χαρακτηρίζονται αυστηρές (δεν επιτρέπεται παρέκκλιση), άλλες ως συστάσεις (προτιμώνται, αλλά επιτρέπεται τεκμηριωμένη εξαίρεση).

3.5.1 Διαστρωμάτωση & Separation of Concerns

Η αρχιτεκτονική διαστρωμάτωσης (Layered Architecture) επιβάλλει τον χωρισμό των τεχνικών ευθυνών σε διακριτά επίπεδα, έτσι ώστε καθετί που αλλάζει για διαφορετικό λόγο να βρίσκεται και σε διαφορετικό στρώμα. Το κλασικό μοντέλο τριών επιπέδων περιλαμβάνει (α) τη διεπαφή, (β) την καθαρή επιχειρησιακή λογική και (γ) το υποσύστημα πρόσβασης στα δεδομένα, είτε αυτά βρίσκονται σε αρχεία είτε σε σχεσιακή βάση. Με τον τρόπο αυτό, μια τροποποίηση στο UI δεν «σκεπάζει» τη λογική του παιχνιδιού, ούτε η αλλαγή βάσης δεδομένων αναγκάζει επιδιόρθωση του front-end. Η επικοινωνία επιτρέπεται μόνο προς το αμέσως κατώτερο επίπεδο, αποφεύγοντας έτσι κυκλικές εξαρτήσεις που αργότερα καθιστούν τον κώδικα δύσκαμπτο και δυσανάγνωστο. Στο πλαίσιο του «Sindopoly» ο διαχωρισμός είναι ακόμη πιο αυστηρός. Τα REST και WebSocket controllers περιορίζονται σε μια απλή μετατροπή αιτημάτων σε κλήσεις υπηρεσιών. Το «game engine» φιλοξενεί όλους τους κανόνες (ρίψη ζαριών, ενοίκια, τερματισμό). Και τέλος, η πρόσβαση τόσο στη μνήμη του Redis όσο και στη μόνιμη βάση της MariaDB γίνεται αποκλειστικά μέσω συγκεκριμένων adapter. Έτσι κάθε στρώμα μπορεί να εξελίσσεται ή να αντικαθίσταται χωρίς να «σπάει» τα υπόλοιπα [46], [47].

3.5.2 SOLID & Clean Code

Οι γνωστές αρχές «SOLID», ένα αναγνωρισμένο σύνολο κανόνων καλής σχεδίασης, υιοθετήθηκαν και εφαρμόστηκαν πιστά για να κρατηθεί ο κώδικας ευέλικτος και επεκτάσιμος. Η αρχή «Single Responsibility» ορίζει ότι κάθε κλάση, συνάρτηση ή πακέτο υπηρετεί έναν και μόνο σκοπό. Αν μια οντότητα χρειάζεται να αλλάξει για δεύτερο λόγο, τότε πρέπει να διασπαστεί. Για παράδειγμα, το «DiceService» (κλάση στο Back-end) έχει μόνο την ευθύνη παραγωγής τυχαίων ζαριών και τίποτε άλλο. Με το «Open/Closed» επιδιώκεται κώδικας «κλειστός» σε τροποποιήσεις αλλά «ανοιχτός» σε επεκτάσεις, για παράδειγμα ένα νέο είδος τετραγώνου (π.χ. «Donation Square») προστίθεται ως υποκλάση χωρίς να χρειαστεί να πειραχτεί ο υπάρχων κώδικας. Το «Liskov Substitution» απαιτεί κάθε υποκλάση να μπορεί να υποκαταστήσει πλήρως τη γονική της χωρίς να ανατρέπεται η συμπεριφορά του συστήματος. Για παράδειγμα, οι εναλλακτικές γεννήτριες ψευδοτυχαίων αριθμών (σχεδόν «true randomness») μπορούν να λειτουργήσουν στη θέση της απλής γεννήτριας που υπάρχει ήδη στο Spring Boot χωρίς να σπάει το παιχνίδι. Παράλληλα το «Interface Segregation» εστιάζει στη χρήση πολλών διεπαφών με μία συγκεκριμένη ευθύνη η κάθε μια, αντί για μια γενική διεπαφή. Με άλλα λόγια, οι κλάσεις υλοποιούν μόνο ό,τι πραγματικά χρησιμοποιούν, αποφεύγοντας «φουσκωμένους» μηχανισμούς. Τέλος, το Dependency Inversion ανιστρέφει την εξάρτηση (όπως λέει και το όνομα, «Αρχή Αντίστροφης Εξάρτησης»). Ουσιαστικά, διευκρινίζει πως αντικείμενα υψηλού επιπέδου

στηρίζονται σε αφαιρέσεις παρά σε αντικείμενα χαμηλού επιπέδου, επιτρέποντας π.χ. την αντικατάσταση της γεννήτριας τυχαίων αριθμών χωρίς αλλαγές στο game-engine [48], [49].

3.5.3 Event-Driven & Ασφαλές Concurrency

Σε περιβάλλοντα πολλαπλών παικτών όπως το «Sindopoly» η ταυτόχρονη πρόσβαση στα ίδια δεδομένα συμβαίνει πολύ συχνά, με άλλα λόγια είναι ο κανόνας και όχι η εξαίρεση. Το «Sindopoly» αντιμετωπίζει το ζήτημα με γεγονότα (events) που δεν μεταβάλλονται εκ των υστέρων. Κάθε γύρος παράγει ένα TURN_END, το οποίο διανέμεται σε όλους τους πελάτες και φυλάσσεται ακριβώς όπως δημιουργήθηκε [50].

3.5.4 Design for Failure & Ανάκαμψη

Η εμπειρία έχει δείξει ότι οποιοδήποτε σύστημα μπορεί να αποτύχει. Συνεπώς, θεωρήθηκε σκόπιμο να σχεδιαστεί το σύστημα έτσι ώστε να συνεχίζει να λειτουργεί ακόμη και κάτω από μερική δυσλειτουργία. Όταν, για παράδειγμα, το διαθέσιμο απόθεμα τυχαίων αριθμών, είτε από κάποιον πολύπλοκο αλγόριθμο που εκμεταλλεύεται μεταβλητές του περιβάλλοντος του συστήματος, είτε από κάποιο εξωτερικό API, εξαντληθεί, ο server γυρίζει αμέσως σε τοπική γεννήτρια που προσφέρει το Spring Boot χωρίς να σταματήσει το παιχνίδι [51].

3.5.5 Αποκλεισμός επιχειρησιακής λογικής από τη γραφική διεπαφή

Η συγκεκριμένη σχεδιαστική αρχή απαιτεί να παραμένει όλος ο κώδικας που υλοποιεί κανόνες παιχνιδιού, επαληθεύσεις ή περίπλοκους υπολογισμούς σε ξεχωριστό, ανεξάρτητο στρώμα του back-end. Στο περιβάλλον του πελάτη (browser) εκτελείται μόνο κώδικας που αντλεί δεδομένα από τα API και τα αποδίδει οπτικά [52]. Ο διαχωρισμός αυτός είναι κρίσιμος για δύο λόγους.

Πρώτον, ο κώδικας της γραφικής διεπαφής (User Interface ή UI) φορτώνεται και μπορεί θεωρητικά να αλλοιωθεί από οποιονδήποτε χρήστη. Αν «κρυβόταν» λογική ελέγχου δικαιωμάτων ή υπολογισμός ενοικίων στο front-end, ένας επιτήδειος θα μπορούσε να την παρακάμψει με το «πείραγμα» JavaScript κώδικα τοπικά. Δεύτερον, τα σύνθετα συστήματα απαιτούν εκτεταμένα tests. Όσο περισσότερη λογική παραμένει στον browser, τόσο δυσκολότερη γίνεται η απομόνωση και ο έλεγχος της.

Στο Sindopoly το UI είναι φτιαγμένο με το ReactJS framework και περιορίζεται ρητά σε ρόλο «παρουσίασης». Κάνει κλήσεις στα REST ή WebSocket endpoints, λαμβάνει JSON και αποτυπώνει το αποτέλεσμα στο ταμπλό ή σε κάποιο modal. Κάθε λογιστική πράξη, από την τιμολόγηση μιας αγοράς μέχρι τον προσδιορισμό νικητή, εκτελείται αποκλειστικά στον server.

3.6 Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική ενός συστήματος είναι ένα από τα πιο σημαντικά σημεία, αν όχι το πιο σημαντικό, στην υλοποίηση ενός συστήματος. Στην ουσία μοιάζει με έναν χάρτη, δηλαδή, όσο πιο πολύ μεγεθύνουμε τόσες περισσότερες λεπτομέρειες για την υλοποιήσει θα ανακαλύψουμε. Από την άλλη όταν κάνουμε μερικά βήματα προς τα πίσω, μπορούμε να διακρίνουμε μόνο τα μεγάλα σχήματα και τις σχέσεις μεταξύ τους. Επομένως, ο αρχιτέκτονας λογισμικού επιλέγει λοιπόν το κατάλληλο επίπεδο μεγέθυνσης, ανάλογα με το είδος της πληροφορίας που χρειάζεται τη δεδομένη στιγμή.

Η παραγωγή των αρχιτεκτονικών διαγραμμάτων ξεκινά συνήθως πριν από την κωδικοποίηση, ώστε όλοι οι εμπλεκόμενοι να αποκτήσουν κοινή εικόνα και να οργανωθούν σε ομάδες με σαφείς ευθύνες. Ωστόσο, τα ίδια διαγράμματα εξακολουθούν να παραμένουν πολύτιμα και μετά την πρώτη έκδοση, καθώς νέοι προγραμματιστές μπορούν, μέσα από αυτά, να κατανοήσουν γρήγορα τη δομή και το σκεπτικό των αποφάσεων που προηγήθηκαν.

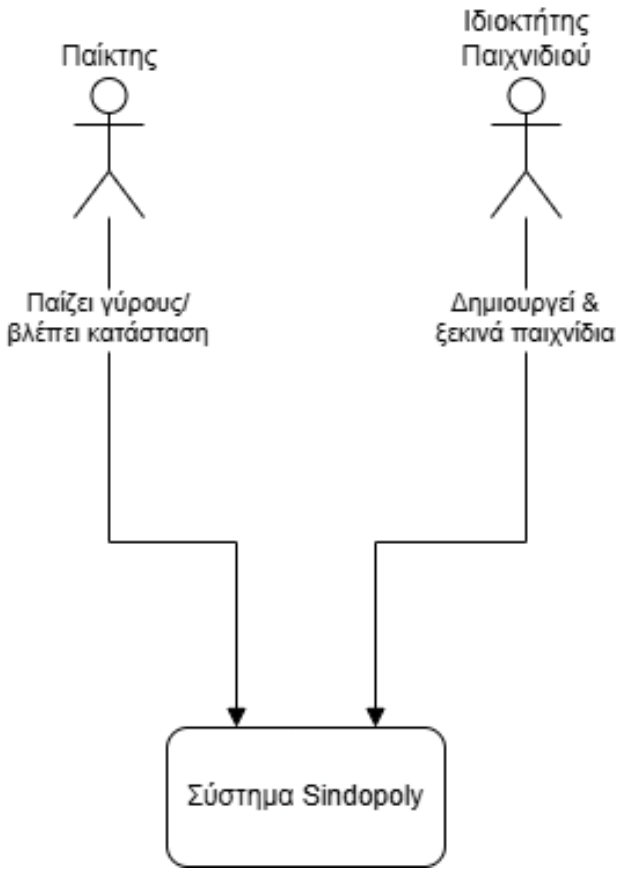
Για το Sindopoly υιοθετήθηκε το C4 model του Simon Brown [53], επειδή προσφέρει κλιμακωτή αναπαράσταση σε τέσσερις διαδοχικές όψεις: Context, Container, Component και Code-level. Κάθε επόμενο διάγραμμα εμβαθύνει μέσα στο προηγούμενο, αποκαλύπτοντας περισσότερη λεπτομέρεια εκεί όπου είναι απαραίτητη. Στις επόμενες ενότητες παρουσιάζονται αναλυτικά τα τέσσερα αυτά επίπεδα και ο ρόλος που διαδραματίζουν στην κατανόηση, και τη διαχρονική συντήρηση, της αρχιτεκτονικής του συστήματος.

3.6.1 Επίπεδο 1 του C4: System Context Diagram

Στο ανώτερο επίπεδο αφαίρεσης το Sindopoly αποτυπώνεται ως ένα σύστημα με το οποίο επικοινωνούν εξωτερικοί χρήστες. Στόχος του Context Diagram είναι να δείξει, με μία ματιά, ποιοι χρήστες επικοινωνούν με το σύστημα και για ποιον λόγο και μέσω ποιου καναλιού. Καμία λεπτομέρεια υλοποίησης ή εσωτερικής δομής δεν εμφανίζεται εδώ.

Στην περίπτωση του Sindopoly διακρίνονται δύο ρόλοι (βλ. Σχήμα 3.1):

- **Ιδιοκτήτης Παιχνιδιού (Game Owner):** Δημιουργεί και εκκινεί νέες παρτίδες, ορίζει το επίπεδο ιδιωτικότητας (public/private lobby) και κωδικό πρόσκλησης αν χρειαστεί.
- **Παίκτης (Player):** Συμμετέχει σε υπάρχουσες παρτίδες και παρακολουθεί την τρέχουσα κατάσταση.

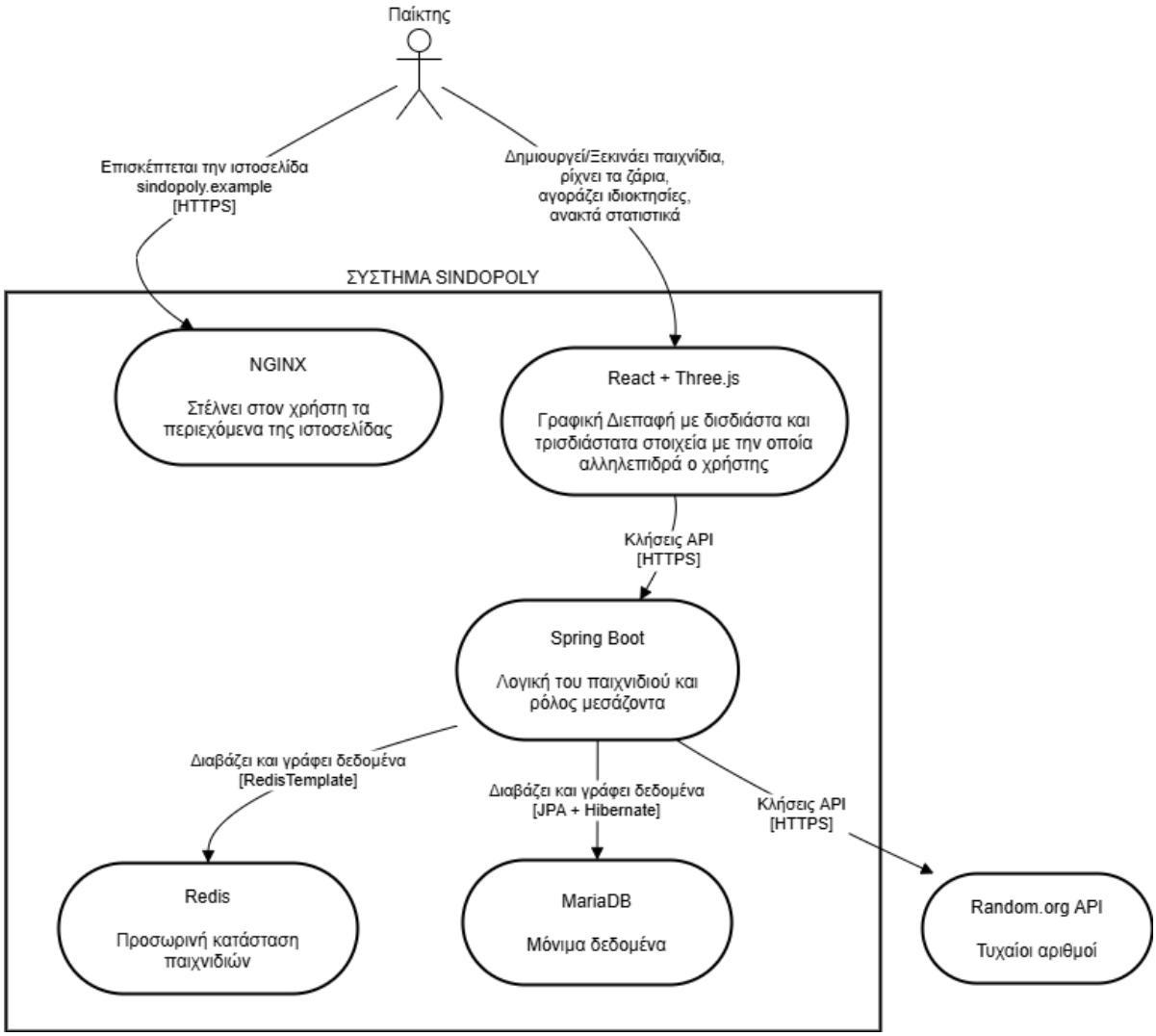


Σχήμα 3.1: Το Context Diagram του συστήματος.

Κάτι που ισχύει γενικότερα είναι πως ο Ιδιοκτήτης Παιχνιδιού είναι πάντα Παίκτης, ενώ ο Παίκτης μπορεί και να μην είναι Ιδιοκτήτης Παιχνιδιού, αν δεν είναι αυτός που έχει δημιουργήσει το παιχνίδι. Δεν εμφανίζονται ακόμη βάσεις δεδομένων, API Gateways ή εξωτερικά Web services. Αυτά θα εμφανιστούν στο ερχόμενο διάγραμμα, το Container Diagram.

3.6.2 Επίπεδο 2 του C4: Container Diagram

Το Container Diagram εμβαθύνει περισσότερο σε σύγκριση με το Context Diagram και αποκαλύπτει όλα τα containers που συνεργάζονται για να υλοποιήσουν το Sindopoly. Κάθε οντότητα λειτουργεί αυτόνομα, διαθέτει σαφείς ευθύνες και συνδέεται με τα υπόλοιπα μέσω συγκεκριμένων πρωτοκόλλων. Οι κύριες τεχνολογικές επιλογές εμφανίζονται ρητά στο διάγραμμα, ώστε να γίνουν ξεκάθαρες οι εξαρτήσεις και τα όρια.



Σχήμα 3.2: Το Container Diagram του συστήματος.

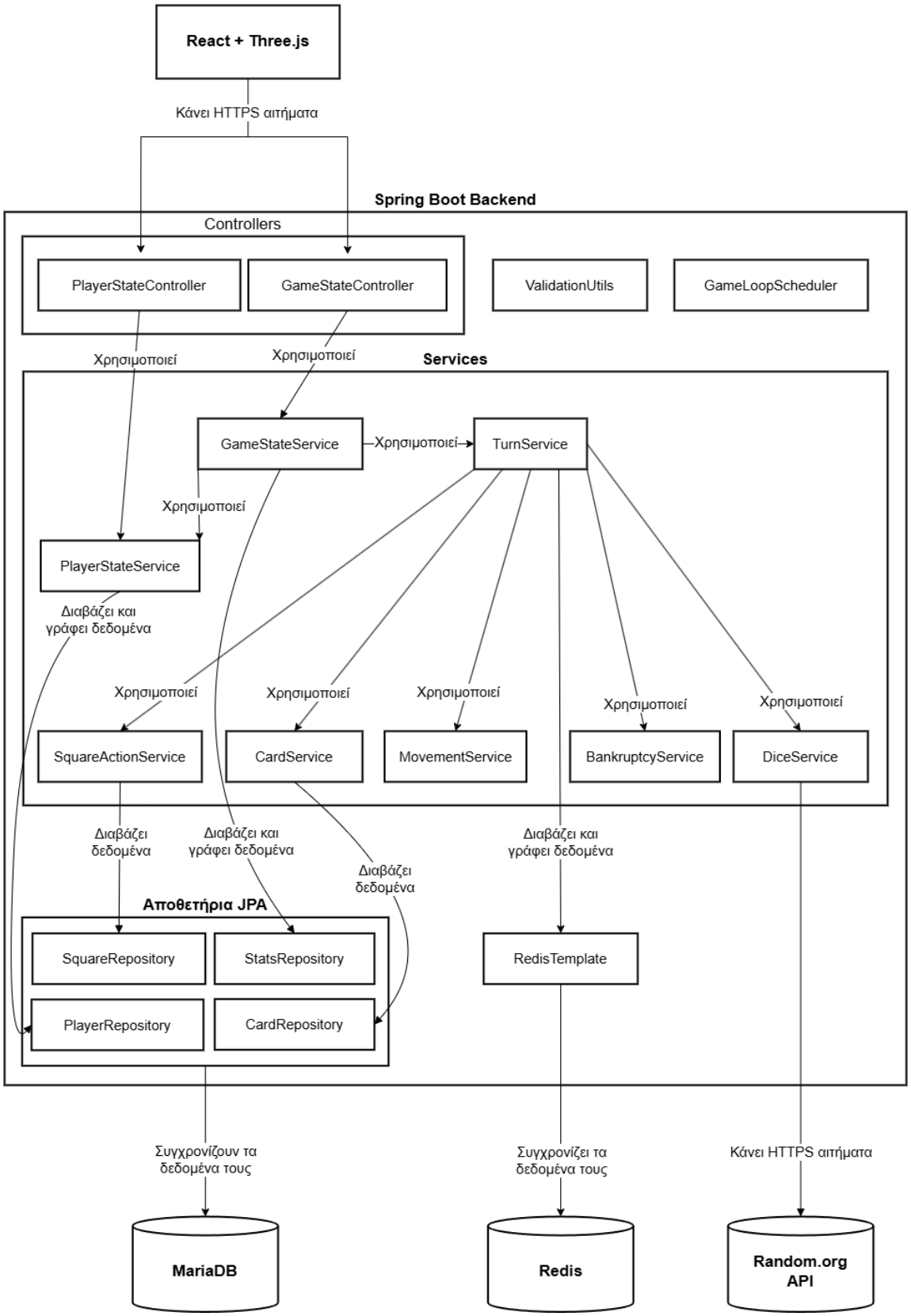
Συγκεκριμένα, στο Container Diagram που φαίνεται στο Σχήμα 3.2, παρουσιάζονται όλα τα παρακάτω συστήματα.

- **NGINX:** Το συγκεκριμένο container σερβίρει στατικά αρχεία όπως HTML, CSS και JavaScript, και στην ουσία παραδίδει όλο τον compiled κώδικα του UI που φτιάχτηκε με τη βιβλιοθήκη React. Λειτουργεί ως κοινό σημείο εισόδου στην τοποθεσία sindopoly.example.
- **React + Three.js:** Η γραφική διεπαφή φορτώνεται στον browser από το NGINX και παρέχει δισδιάστατα και τρισδιάστατα στοιχεία. Οι χρήστες μέσω αυτής δημιουργούν/ξεκινούν παιχνίδια, ρίχνουν ζάρια, αγοράζουν ιδιοκτησίες και βλέπουν τα στατιστικά τους. Επικοινωνεί με τη Spring Boot σε φανερωμένα REST endpoints της μέσω HTTPS αιτημάτων.
- **Spring Boot:** Αποτελεί τον «μεσάζοντα», καθώς υποδέχεται HTTPS αιτήματα από τη γραφική διεπαφή και της παρέχει τα απαιτούμενα δεδομένα, καθώς επίσης διαχειρίζεται την επικοινωνία και με τα υπόλοιπα containers του συστήματος. Συγκεκριμένα, αλληλεπιδρά με τη βάση δεδομένων MariaDB μέσω των JPA και Hibernate drivers, με τη βάση δεδομένων Redis μέσω του ειδικού ενσωματωμένου Redis driver της Spring Boot, και με εξωτερικά APIs όπως το API της σελίδας random.org μέσω HTTPS αιτημάτων. Βρίσκεται κυριολεκτικά στη μέση όλων των containers και των διεργασιών τους. Επίσης, αποτελεί την επιχειρησιακή λογική (business logic) του συστήματος. Στην ουσία, διαχειρίζεται και οργανώνεις όλες τις λειτουργίες του συστήματος, όπως τους κανόνες του παιχνιδιού, τη σειρά των παικτών, τις αγορές, τις χρεοκοπίες, τα στατιστικά, κλπ..
- **Redis:** Το συγκεκριμένο container φυλάσσει την προσωρινή κατάσταση κάθε ενεργής παρτίδας, δηλαδή όλα τα στοιχεία της, από τη θέση ενός παίκτη, μέχρι και τα λεφτά του και τις ιδιοκτησίες του. Η in-memory φύση του επιτρέπει χαμηλή καθυστέρηση στις ενημερώσεις των γύρων.
- **MariaDB:** Η MariaDB φυλάσσει τα μόνιμα δεδομένα του συστήματος. Αυτά είναι οι θέσεις του ταμπλό (Squares), οι λίστες των καρτών απόφασης και εντολής (Cards), τα βασικά στοιχεία των παικτών (Players), και τα συγκεντρωτικά στατιστικά του κάθε παίκτη (Stats). Σκοπός είναι να χρησιμοποιείται μόνο όταν χρειάζεται, εξαιτίας της εξάρτησής της στον σκληρό δίσκο συ σύγκριση με το Redis που εξαρτιέται κυρίως από τη RAM, το οποίο κάνει τη MariaDB συγκριτικά πιο αργή σε ενέργειες διαβάσματος και γραψίματος (read and write). Επομένως, φυλάσσει τα στοιχεία που δεν είναι άμεσα απαραίτητα στο σύστημα κάθε στιγμή ενός παιχνιδιού.
- **Random.org API:** Αποτελεί ένα εξωτερικό API το οποίο διαχειρίζεται από τρίτους. Παρέχει τυχαίους αριθμούς που αποθηκεύονται σε μια τοπική ουρά και χρησιμοποιούνται για ρίψεις ζαριών. Χρησιμοποιείται για την αξιοπιστία του σε θέματα που αφορούν το «true randomness» και τη γενικότερη εξασφάλιση ότι οι ρίψεις των ζαριών δεν θα είναι προβλέψιμες. Η σχέση αυτή είναι καθαρά καταναλωτική, χωρίς αμοιβαία εξάρτηση.

Το συγκεκριμένο διάγραμμα παρέχει την πλήρη σειρά εξαρτήσεων μεταξύ containers, στο οποίο αποσκοπεί κιόλας, χωρίς να αναλώνετε σε λεπτομέρειες σχετικές με τον κώδικα οι οποίες θα φανούν στο επόμενο επίπεδο, το Component Diagram.

3.6.3 Επίπεδο 3 του C4: Component Diagram

Το Component Diagram εστιάζει στην εσωτερική αρχιτεκτονική του πιο «πλούσιου» container, δηλαδή του back-end Spring Boot. Εδώ αποτυπώνονται τα επιμέρους συστατικά (components) και οι σχέσεις μεταξύ τους. Με αυτό τον τρόπο ένας προγραμματιστής να αντιληφθεί πού ακριβώς βρίσκεται κάθε ευθύνη μέσα στον κώδικα.



Σχήμα 3.3: Το Component Diagram του συστήματος.

Τέρμα πάνω στο διάγραμμα που φαίνεται στο Σχήμα 3.3 διακρίνεται το React + Three.js container, το οποίο λαμβάνει και στέλνει HTTPS αιτήματα από και προς το API. Το πρώτο component, κάτω από το React + Three.js container και μέσα στο Spring Boot container, είναι οι Controllers, συγκεκριμένα ο GameStateController και ο PlayerStateController. Ο GameStateController χειρίζεται ενέργειες που αφορούν ολόκληρη την παρτίδα, πχ. τη δημιουργία μιας παρτίδας, την εκκίνηση της, τον τερματισμό, την εκτέλεση των γύρων, κλπ., ενώ ο PlayerStateController εξυπηρετεί τις ενέργειες ενός μεμονωμένου παίκτη όπως αγορές ιδιοκτησιών, την είσοδο και έξοδο του από τη ΦΥΛΑΚΗ, καθώς και τη γενικότερη προσωρινή υπόστασή του στη βάση του Redis για τη διαχείριση των ενεργειών του μέσα σε μια μεμονωμένη παρτίδα.

Αμέσως από κάτω βρίσκουμε το component Services, όπου βρίσκεται η επιχειρησιακή λογική (business logic). Ουσιαστικά, τα κομμάτια που υλοποιούν όλες τις λειτουργίες των δύο προαναφερθέντων Controllers είναι το GameStateService και το PlayerStateService. Το GameStateService, για να λειτουργήσει σωστά, αξιοποιεί το TurnService.

Το TurnService είναι το κέντρο το οποίο διαχειρίζεται τη σωστή ροή και εκτέλεση των γύρων. Ουσιαστικά, επαληθεύει τη σειρά των παικτών, καλεί το DiceService για τη ρίψη των ζαριών, ζητά από το MovementService να μετακινήσει το πιόνι του παίκτη και, όπου χρειάζεται, ελέγχει αν ο παίκτης έχει φτάσει στο σημείο της χρεοκοπίας (λεφτά < 0) μέσω του BankruptcyService. Το SquareActionService αποφασίζει τι θα συμβεί όταν ένας παίκτης πατήσει πάνω σε ένα κουτί ιδιοκτησίας, φόρου ή κάρτας, ενώ το CardService εκτελεί το περιεχόμενο μιας τυχαίας κάρτας που έχει επιλεχθεί για τον παίκτη.

Διακριτό αλλά ουσιαστικό ρόλο επωμίζεται ο GameLoopScheduler, ο οποίος λειτουργεί ανεξάρτητα και σε τακτά χρονικά διαστήματα αυξάνει το χρόνο που έχει περάσει από τότε που ξεκίνησε μια παρτίδα, ανιχνεύει παίκτες που δεν έχουν πραγματοποιήσει κάποια ενέργεια για μεγάλα χρονικά διαστήματα (AFK players) και ελέγχει αν πληρούνται οι συνθήκες λήξης μιας παρτίδας. Το ValidationUtils χρησιμοποιείται σε όλο το επίπεδο για να επαληθεύει ότι το σύστημα παραμένει μέσα στο πλαίσιο που θέτουν κάποιοι κανόνες, όπως την επιβεβαίωση ότι είναι η σειρά του συγκεκριμένου παίκτη, ότι το παιχνίδι βρίσκεται σε κατάσταση «lobby» ή όχι, και ότι τα χρήματα ενός παίκτη δεν είναι ποτέ αρνητικός αριθμός πέρα από τις περιπτώσεις πτώχευσης.

Το DiceService είναι υπεύθυνο για να επικοινωνεί με το API της random.org και να παίρνει τυχαίους αριθμούς από εκεί. Σε περίπτωση που δεν μπορεί να επικοινωνήσει μαζί του, έχεις ως εναλλακτική (fallback) την ειδική συνάρτηση που υπάρχει ήδη στο Spring Boot για τη δημιουργία τυχαίων αριθμών. Φροντίζει να αποθηκεύει ένα μεγάλο αριθμό τυχαίων αριθμών σε ειδικό set στο Redis με σκοπό τη γρήγορη προσκόμιση αυτών των αριθμών στο σύστημα

Στο κατώτερο επίπεδο της επιχειρησιακής λογικής βρίσκονται τα Αποθετήρια JPA (JPA Repositories) τα οποία είναι υπεύθυνα για την επικοινωνία με τη βάση δεδομένων, και είναι τα PlayerRepository, StatsRepository, SquareRepository, CardRepository. Χρησιμοποιούν τις τεχνολογίες JPA και Hibernate για να κάνουν CRUD (Create, Read, Update, Delete) ενέργειες στη MariaDB. Υπάρχουν οι αντίστοιχοι πίνακες στη MariaDB, στους οποίους αντιστοιχίζονται τα παραπάνω Αποθετήρια. Συγκεκριμένα:

- Το PlayerStateService διαβάζει και γράφει δεδομένα στο PlayerRepository σχετικά με τα βασικά στοιχεία των παικτών, όπως το όνομά τους.
- Το SquareActionService διαβάζει μονάχα δεδομένα από το SquareRepository, καθώς το συγκεκριμένο αποθετήριο επιστρέφει τις ιδιότητες ενός τετραγώνου του ταμπλό, ώστε να

καθοριστούν οι ενέργειες που θα μπορεί να πραγματοποιήσει ο παίκτης ή που θα πραγματοποιηθούν αυτομάτως.

- Το CardService διαβάζει μονάχα δεδομένα από το CardRepository για παρόμοιο λόγο. Το συγκεκριμένο αποθετήριο θα τραβήξει από τη βάση της MariaDB το περιεχόμενο μιας τυχαίας κάρτας για να καθοριστούν η ενέργεια ή οι ενέργειες που θα πραγματοποιηθούν αυτομάτως.
- Το GameStateService διαβάζει και γράφει δεδομένα στο StatsRepository, καθώς το συγκεκριμένο αποθετήριο είναι υπεύθυνο για την αποθήκευση και την εμφάνιση των στατιστικών δεδομένων των παικτών.

Παράλληλα, μέσω του RedisTemplate, το οποίο είναι ένα βοηθητικό API της βιβλιοθήκης Spring Data Redis, πραγματοποιούνται κλήσεις στη βάση του Redis. Το χρησιμοποιεί το TurnService για να αποθηκεύει και να βλέπει την προσωρινή κατάσταση της παρτίδας.

Τέλος, το διάγραμμα υποδεικνύει ότι το DiceService επικοινωνεί με ένα εξωτερικό API, το API της πλατφόρμας random.org, από όπου ζητούνται μεγάλες παρτίδες τυχαίων αριθμών. Εφόσον αποτελεί μια εξωτερική «βάση» στη συγκεκριμένη περίπτωση, απεικονίζεται ως ένας ξεχωριστός κύλινδρος.

3.6.4 Επίπεδο 4 του C4: Code-Level Diagram

Στο τέταρτο και τελευταίο επίπεδο απεικόνισης περνάμε στις ίδιες τις κλάσεις του domain μοντέλου. Το διάγραμμα, παρότι θυμίζει UML Class Diagram, δεν ακολουθεί αυστηρά την υψηλή λεπτομέρεια του UML. Παρουσιάζει μόνο ό,τι είναι χρήσιμο για να κατανοήσει ένας προγραμματιστής τη δομή και τις βασικές σχέσεις του παιχνιδιού, χωρίς να χαθεί σε αχρείαστες λεπτομέρειες. Η λογική πίσω από αυτή την ελαφριά προσέγγιση είναι να δώσει στον νέο developer μία «big picture» των δομικών κλάσεων: τι συγκρατεί την κατάσταση του παιχνιδιού και πώς αρθρώνονται οι κανόνες.

3.7 Οντότητες

Στη συγκεκριμένη ενότητα περιγράφεται η μεθοδολογία που ακολουθήσαμε για να αναλύσουμε τη «φύση» των δεδομένων του Sindopoly και να τα μετασχηματίσουμε σε ενιαίο, εννοιολογικό σχήμα οντοτήτων–σχέσεων. Για κάθε βασική οντότητα του παιχνιδιού, όπως τα Game, Player, Square, Card και Stats, συντάσσεται πίνακας με τα κύρια χαρακτηριστικά, τον προτεινόμενο τύπο δεδομένων και την επιχειρησιακή σημασία κάθε χαρακτηριστικού. Η επιλογή των οντοτήτων προέκυψε από: (α) λεπτομερή μελέτη του Monopoly και των κανόνων του, καθώς και των ειδικών παραλλαγών του Sindopoly, (β) ανασκόπηση των REST/WebSocket μηνυμάτων που ανταλλάσσονται ήδη στον κώδικα, και (γ) μελέτη των στατιστικών που μπορεί να ενδιαφέρει τους παίκτες (π.χ. συνολικές νίκες, διάρκεια παρτίδων), αφήνοντας περιθώριο να πραγματοποιηθεί «gamification» του παιχνιδιού. Οι συσχετίσεις, όπως «ένας Player ανήκει σε ένα Game» ή «ένα Square μπορεί να ανήκει σε κανέναν ή έναν Player», θα αναλυθούν σε ξεχωριστή υποενότητα όπου καταγράφεται και η πολλαπλότητά τους.

Παρότι το Sindopoly είναι κατά βάση real-time εφαρμογή με έντονη χρήση της in-memory βάσης Redis για να αποθηκεύει μια προσωρινή κατάσταση, η σωστή μοντελοποίηση του μόνιμου σχήματος (MariaDB) παραμένει κρίσιμη. Αν τυχόν σχεδιαστεί εσφαλμένα, κινδυνεύει να χαθεί η ακεραιότητα των αποτελεσμάτων, ενώ μελλοντικές αλλαγές στους κανόνες του παιχνιδιού θα απαιτούν κοστοβόρες μετατροπές δεδομένων. Η ανάλυση που ακολουθεί κινείται σε επίπεδο conceptual, χωρίς δηλαδή να δένει το σχήμα σε κάποια συγκεκριμένη τεχνολογία, ώστε να μπορεί, εφόσον χρειαστεί, να μεταφερθεί σε άλλο RDBMS χωρίς δομικές απώλειες.

3.7.1 Οντότητα: Game

Κάθε στιγμιότυπο της οντότητας Game αντιστοιχεί σε μία συγκεκριμένη παρτίδα που υπάρχει και τρέχει εκείνη τη στιγμή (είτε σε κατάσταση «lobby», είτε σε κατάσταση «inProgress»). Η συγκεκριμένη οντότητα υπάρχει μονάχα στο Redis, καθώς η μόνιμη αποθήκευση της κατάστασης ενός παιχνιδιού δεν είναι επιθυμητή και ούτε αποσκοπεί κάπου. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.5.

Πίνακας 3.5: Οντότητα Game στο Redis

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
gameId	INTEGER	Πρωτεύον κλειδί. Ταυτίζεται με το ownerId για να γίνει πιο εύκολη η αναζήτηση του.
status	Enum {lobby, inProgress, ended}	Η τρέχουσα κατάσταση του παιχνιδιού.
visibility	ENUM {public, private}	Αν είναι δημόσιο ένα παιχνίδι ή ιδιωτικό.
inviteCode	VARCHAR(5)	Μοναδικός κωδικός για ιδιωτικά παιχνίδια. Αν το παιχνίδι είναι δημόσιο τότε αυτό το πεδίο παραμένει NULL.
creationTime	TIMESTAMP ISO 8601	Η χρονική σφραγίδα δημιουργίας ενός παιχνιδιού.
startTime	TIMESTAMP ISO 8601	Η χρονική σφραγίδα ενός παιχνιδιού. Αυτό το χαρακτηριστικό και το creationTime δεν ταυτίζονται, καθώς αυτό μετράει το χρόνο που πέρασε όσο το παιχνίδι τρέχει, άρα είναι σε κατάσταση «inProgress» και όχι «lobby». Για αυτό και όταν δημιουργείτε ένα στιγμιότυπο αυτής της οντότητας, το συγκεκριμένο χαρακτηριστικό είναι NULL, μέχρι να ξεκινήσει ο owner το παιχνίδι.
ownerId	INTEGER	Ξένο κλειδί με προέλευση την οντότητα Player στη MariaDB. Το ID του παίκτη που είναι ταυτόχρονα και ο δημιουργός του παιχνιδιού.
playerTurn	INTEGER	Το ID του παίκτη, του οποίου είναι η σειρά να παίξει.
timer	INTEGER	Πόσα δευτερόλεπτα έχουν περάσει από την έναρξη του παιχνιδιού.
maxTime	INTEGER	Το χρονικό όριο των παιχνιδιών σε δευτερόλεπτα. Σε αυτή τη φάση αυτό είναι ένα προκαθορισμένο χρονικό όριο που έχει επιλεγεί από τον

		προγραμματιστή, και όχι από τον owner του κάθε παιχνιδιού.
maxMoney	INTEGER	Το όριο των χρημάτων που απαιτείται να έχει κάποιος παίκτης για να τελειώσει το παιχνίδι. Αυτό και το maxTime είναι τα end conditions του παιχνιδιού. Σε αυτή τη φάση αυτό είναι ένα προκαθορισμένο όριο που έχει επιλεγεί από τον προγραμματιστή, και όχι από τον owner του κάθε παιχνιδιού.
Winner	INTEGER	Ξένο κλειδί με προέλευση την οντότητα Player στη MariaDB. Το ID του νικητή του παιχνιδιού. Στην αρχή παίρνει την τιμή NULL, και συμπληρώνεται στο τέλος του παιχνιδιού.
lastDice	INTEGER	Η τελευταία τιμή που χρησιμοποιήθηκε για τη ρίψη των ζαριών. Είναι χρήσιμο για τα utilities του συστήματος.
turnAge	INTEGER	Πόση ώρα έχει περάσει για έναν μονάχα γύρο ενός παίκτη.
turnTimeout	INTEGER	Πόση ώρα επιτρέπεται να περάσει για έναν παίκτη πριν γίνει παράλειψη του γύρου του λόγω αδράνειας (auto-skip turn, χαρακτηρίζεται αυτός ο παίκτης ως AFK – Away From Keyboard). Σε αυτή τη φάση αυτό είναι ένα προκαθορισμένο χρονικό όριο που έχει επιλεγεί από τον προγραμματιστή, και όχι από τον owner του κάθε παιχνιδιού.

3.7.2 Οντότητα: Player

Κάθε στιγμιότυπο της οντότητας Player αντιστοιχεί σε έναν παίκτη που βρίσκεται σε κάποιο παιχνίδι.

Πίνακας 3.6: Οντότητα Player στη MariaDB

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
playerId	INTEGER	Πρωτεύον κλειδί. Το ID του παίκτη.
p_name	VARCHAR(255)	Το όνομα (ψευδώνυμο) του παίκτη.
created_at	TIMESTAMP	Η χρονική σφραγίδα που αφορά τη δημιουργία του «λογαριασμού» του παίκτη.

Ο Πίνακας 3.6 αφορά την οντότητα Player που υπάρχει στη MariaDB. Προς το παρόν, ο τρόπος με τον οποίο έχει υλοποιηθεί επιτρέπει στον χρήστη να φτιάξει έναν προσωρινό «λογαριασμό», δίνοντας μονάχα ένα ψευδώνυμο. Για αυτό και χρησιμοποιείται το χαρακτηριστικό «created_at», επιτρέποντας να μετρηθεί ο χρόνος που υπάρχει αυτός ο λογαριασμός και να «σβηστεί» αυτόματα μετά το πέρασμα ενός συγκεκριμένου χρονικού ορίου (1 ημέρα / 24 ώρες). Αν στο μέλλον χρησιμοποιηθεί κάποιος άλλος τρόπου αυθεντικοποίησης, δεν θα υπάρχει ανάγκη πλέον για αυτό το χαρακτηριστικό.

Πίνακας 3.7: Οντότητα Player στο Redis

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
playerId	INTEGER	Πρωτεύον κλειδί. Ξένο κλειδί με προέλευση την οντότητα Player στη MariaDB. Το ID του παίκτη.
position	INTEGER	Η θέση του παίκτη στο ταμπλό.
money	INTEGER	Το τρέχον υπόλοιπο των χρημάτων του παίκτη.
properties	List< INTEGER >	Μια λίστα με τα IDs των ιδιοκτησιών που κατέχει ο παίκτης.
jail	BOOLEAN	Μια λογική τιμή που υποδεικνύει αν ο παίκτης βρίσκεται στη φυλακή ή όχι.
jailTime	INTEGER	Πόσοι γύροι απομένουν για τον παίκτη μέχρι να βγει από τη φυλακή. Ουσιαστικά πόσοι γύροι απομένουν για την παραμονή του στη φυλακή.
bankrupt	BOOLEAN	Μια λογική τιμή που υποδεικνύει αν ο παίκτης βρίσκεται σε κατάσταση χρεοκοπίας ή όχι.
afk	BOOLEAN	Μια λογική τιμή που υποδεικνύει αν ο παίκτης έχει χαρακτηριστεί ως AFK λόγω αδράνειας.

Ο Πίνακας 3.7 αφορά την οντότητα Player που υπάρχει στο Redis. Χρησιμοποιείται για να αποθηκεύει τα στοιχεία του παίκτη όσο βρίσκεται σε ένα παιχνίδι. Όπως τη θέση του στο ταμπλό, τις ιδιοκτησίες που έχει αγοράσει, το αν βρίσκεται στη φυλακή ή όχι, κλπ.. Δημιουργείται μονάχα όταν μπαίνει σε ένα παιχνίδι, και διαγράφεται όταν τελειώνει το παιχνίδι στο οποίο βρίσκεται.

3.7.3 Οντότητα: Game-Players

Κάθε στιγμιότυπο της συγκεκριμένης οντότητας αντιστοιχεί στη συσχέτιση του Game με τους Players του. Δεν είναι «πίνακας» βάσης SQL, αλλά ένα Set, καθώς η συγκεκριμένη οντότητα υπάρχει μονάχα στο Redis. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.8.

Πίνακας 3.8: Οντότητα Game-Players στο Redis

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή

game:{gameId}:players	Redis Set<String>	Κλειδί-σετ που αποθηκεύει τα playerId (ως κείμενο) των παικτών που συμμετέχουν στο συγκεκριμένο παιχνίδι.
members	Set	Τα IDs όλων των παικτών του παιχνιδιού.
size	Long	Το μέγεθος ενός «lobby», δηλαδή ο αριθμός παικτών σε ένα παιχνίδι. Αξιοποιείται για ελέγχους τύπου MAX_PLAYERS = 4 και για το auto-end όταν μείνει 1 παίκτης.

3.7.4 Οντότητα: Square

Κάθε στιγμιότυπο της οντότητας Square αντιστοιχεί σε ένα τετράγωνο στο ταμπλό. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.9.

Πίνακας 3.9: Οντότητα Square στη MariaDB

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
id	TINYINT	Πρωτεύον κλειδί. Το ID του τετραγώνου.
s_type	ENUM {START, CARD, TAX, JAIL, GOJAIL, PARKING, TRAIN, UTILITY, BROWN, CYAN, PURPLE, ORANGE, RED, YELLOW, GREEN, BLUE}	Η κατηγορία στην οποία ανήκει το τετράγωνο, βάσει της οποίας θα κριθεί ποια ή ποιες ενέργειες θα πραγματοποιηθούν.
price	SMALLINT	Η τιμή αγοράς του τετραγώνου (μόνο για ιδιοκτησίες).
rent	SMALLINT	Το ποσό του βασικού ενοικίου ή του φόρου του τετραγώνου (μόνο για ιδιοκτησίες και φόρους).

3.7.5 Οντότητα: Player-Property

Κάθε στιγμιότυπο της οντότητας Player-Property αντιστοιχεί σε ένα τετράγωνο στο ταμπλό και τον ιδιοκτήτη του. Δεν είναι «πίνακας» βάσης SQL, αλλά ένα Set, καθώς η συγκεκριμένη οντότητα υπάρχει μονάχα στο Redis. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.10.

Πίνακας 3.10: Οντότητα Player-Property στο Redis

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή

KEY_square	STRING	Πρωτεύον κλειδί. Το ID ενός τετραγώνου.
owner	STRING	Ξένο κλειδί από το Set «Player». Αποτελεί τον μοναδικό ιδιοκτήτη του συγκεκριμένου τετραγώνου (μόνο για ιδιοκτησίες)

3.7.6 Οντότητα: Card

Κάθε στιγμιότυπο της οντότητας Card αντιστοιχεί σε κάρτα απόφασης ή εντολής. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.11.

Πίνακας 3.11: Οντότητα Card στη MariaDB

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
id	SMALLINT	Πρωτεύον κλειδί. Το ID της κάρτας.
title	VARCHAR(200)	Ένας σύντομος τίτλος για την κάρτα.
description	VARCHAR(1000)	Η περιγραφή της κάρτας που εμφανίζεται στον χρήστη.
type	ENUM {money, transport}	Το είδος τη ενέργειας που πρόκειται να πραγματοποιηθεί.
amount	FLOAT	Ένας αριθμός που υποδεικνύει, είτε τον δείκτη του τετραγώνου στο οποίο πρέπει να μετακινηθεί ο παίκτης, είτε το χρηματικό ποσό που θα πάρει (θετικός αριθμός) ή θα πληρώσει (αρνητικός αριθμός) ο παίκτης.

3.7.7 Οντότητα: Card-Deck

Κάθε στιγμιότυπο της οντότητας Card-Deck αντιστοιχεί σε μία «τράπουλα», δηλαδή ένα τυχαία ανακατεμένο σύνολο των καρτών απόφασης και εντολής. Δεν είναι «πίνακας» βάσης SQL, αλλά ένα Set, καθώς η συγκεκριμένη οντότητα υπάρχει μονάχα στο Redis. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.12.

Πίνακας 3.12: Οντότητα Card-Deck στο Redis

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
KEY_deck	STRING	Πρωτεύον κλειδί. Το ID της συγκεκριμένης τράπουλας.
cardIds	LIST	Μία λίστα ξένων κλειδιών από τον πίνακα «Card». Αποτελεί μια λίστα με κάρτες από τον πίνακα «Card», οι οποίες είναι τυχαία ανακατεμένες .

3.7.8 Οντότητα: Stats

Κάθε στιγμιότυπο της οντότητας Stats αντιστοιχεί στα στατιστικά στοιχεία ενός παίκτη. Τα στοιχεία της παρουσιάζονται στον Πίνακας 3.13.

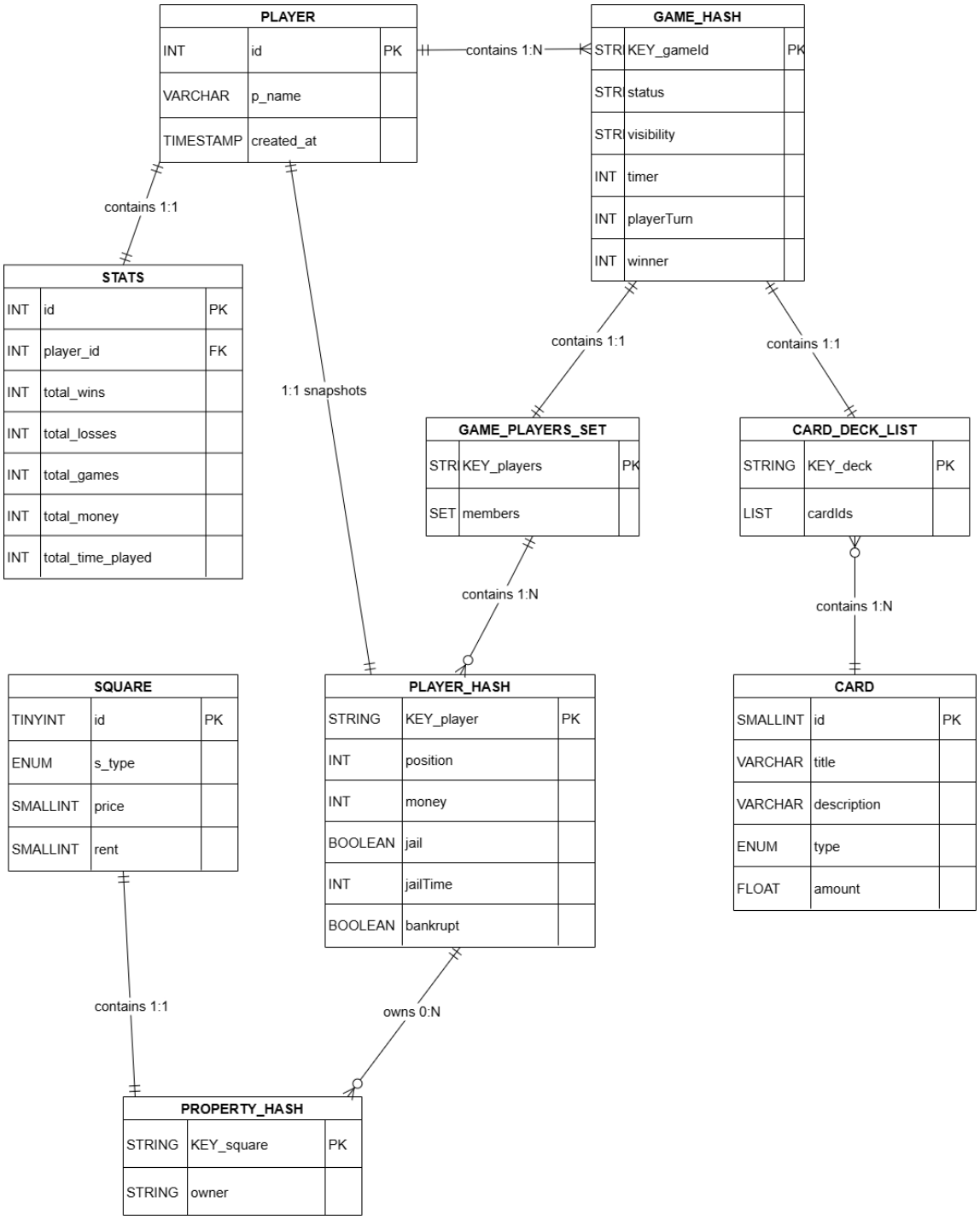
Πίνακας 3.13: Οντότητα Stats στη MariaDB

Χαρακτηριστικό	Τύπος Δεδομένων	Περιγραφή
id	INTEGER	Πρωτεύον κλειδί. Το ID των συγκεκριμένων στατιστικών.
player	INTEGER	Ξένο κλειδί. Το ID του παίκτη με τον οποίο συσχετίζονται τα συγκεκριμένα χαρακτηριστικά.
total_wins	INTEGER	Ο συνολικός αριθμός των νικών του παίκτη.
total_losses	INTEGER	Ο συνολικός αριθμός των ηττών του παίκτη.
total_games	INTEGER	Ο συνολικός αριθμός των παιχνιδιών που έχει ολοκληρώσει ο παίκτης.
total_money	INTEGER	Ο συνολικός αριθμός των χρημάτων που έχει μαζέψει ο παίκτης από τα παιχνίδια που έχει παίξει. Προσμετρούνται μονάχα τα λεφτά που είχε στο τέλος του παιχνιδιού, και όχι όλα τα λεφτά που «πέρασαν από τα χέρια του» κατά τη διάρκεια του παιχνιδιού.
total_time_played	INTEGER	Ο συνολικός χρόνος όλων των παιχνιδιών στα οποία έχει συμμετάσχει ο παίκτης.

3.8 Διάγραμμα ER

Το Entity–Relationship Diagram (ER), ή αλλιώς Διάγραμμα Σχέσεων-Οντοτήτων, αποτελεί το αφετηριακό βήμα για τη μοντελοποίηση των δεδομένων του Sindopoly. Λειτουργεί σε καθαρά εννοιολογικό επίπεδο, χωρίς να δεσμεύει το σχήμα σε κάποιο συγκεκριμένο RDBMS. Χάρη στη χαλαρότερη σύνταξή του, σε σχέση με το αυστηρό UML, επιτρέπει στους αναλυτές να εστιάσουν στις οντότητες που έχουν επιχειρησιακή αξία, όπως τις Game, Player, Square, Card, Stats, και στις μεταξύ τους σχέσεις, όπως «ένας Player ανήκει σε ένα Game» ή «ένα Square μπορεί να ανήκει σε κανέναν ή έναν Player». Η δημιουργία ER διαγράμματος προηγείται πάντα της φυσικής υλοποίησης στη MariaDB. Στη συνέχεια το ίδιο μοντέλο μπορεί, χωρίς τροποποίηση, να χαρτογραφηθεί σε άλλη σχεσιακή βάση, εάν υπάρχουν μελλοντικές ανάγκες κλιμάκωσης που το επιβάλουν.

Το Σχήμα 3.4 παρουσιάζει το πλήρες ER διάγραμμα που υιοθετήθηκε για το Sindopoly.



Σχήμα 3.4: Το Entity-Relationship Diagram του συστήματος.

Στο Σχήμα 3.4 έχει προσαρμοστεί η λογική του συστήματος ώστε να μπορούν να απεικονιστούν σε ένα μονάχα διάγραμμα και η MariaDB και το Redis και οι μεταξύ τους σχέσεις και η γενικότερη λογική που τα συνδέει. Επομένως, το σχήμα αποτυπώνει σε μία εικόνα τόσο το επίμονο (persistent) κομμάτι των δεδομένων (MariaDB) όσο και το προσωρινό, in-memory state που ζει στο Redis όσο διαρκεί μια παρτίδα.

Στο επάνω μέρος βρίσκονται οι μόνιμες οντότητες. Η πρώτη, η PLAYER, διατηρεί το σταθερό προφίλ κάθε χρήστη, ενώ η STATS επιμηκύνει το προφίλ με στατιστικά (νίκες, συνολικό χρόνο παιχνιδιού,

κέρδη, κλπ.) σε σχέση ένα-προς-ένα με το αντίστοιχο PLAYER. Δίπλα, οι πίνακες SQUARE και CARD ορίζουν το αμετάβλητο «λεξιλόγιο» του ταμπλό: τα 40 τετράγωνα του και τις κάρτες απόφασης και εντολής αντίστοιχα.

Κάθε φορά που δημιουργείται νέα παρτίδα, γεννιέται ένα hash GAME_HASH μέσα στη Redis. Εκεί αποθηκεύεται η τρέχουσα κατάσταση της παρτίδας (χρονόμετρο, παίκτης που έχει σειρά, τελευταίο ζάρι, κ.λπ.). Το hash αυτό συνδέεται μοναδικά με δύο ακόμη δομές, ένα set GAME_PLAYERS_SET που κρατά τα αναγνωριστικά των συμμετεχόντων και μια λίστα CARD_DECK_LIST με ανακατεμένες κάρτες. Η ετικέτα «contains 1:1» στο διάγραμμα υποδηλώνει ότι κάθε παρτίδα κατέχει ακριβώς ένα τέτοιο set και ένα τέτοιο deck.

Κάθε μέλος του set αντιστοιχεί σε δικό του PLAYER_HASH, ένα runtime στιγμιότυπο που μεταφέρει τα μεταβλητά στοιχεία του παίκτη εν ώρα παιχνιδιού (χρήματα, θέση στο ταμπλό, αν ο παίκτης είναι στη φυλακή, κλπ.). Το στιγμιότυπο αυτό θεωρείται «snapshot» του σταθερού PLAYER, γι’ αυτό η σχέση χαρακτηρίζεται 1-προς-1. Ένας παίκτης μπορεί να έχει μόνο ένα ενεργό snapshot μέσα σε συγκεκριμένη παρτίδα. Από το PLAYER_HASH εκκινούν, τέλος, οι συσχετίσεις προς PROPERTY_HASH. Κάθε ιδιοκτησία που αγοράστηκε δηλώνεται εκεί, δείχνοντας ποιος παίκτης την κατέχει. Η πολλαπλότητα «owns 0:N» επιτρέπει σε παίκτη να διαθέτει καμία ή πολλές ιδιοκτησίες.

Το PROPERTY_HASH συνδέεται ένα-προς-ένα με το σταθερό SQUARE, εξασφαλίζοντας ότι κάθε τετράγωνο έχει το πολύ έναν ιδιοκτήτη ανά ενεργή παρτίδα. Αντίστοιχα, το CARD_DECK_LIST παραπέμπει στην οντότητα CARD. Κατά την κλήρωση, το παιχνίδι απλώς κάνει POP την κορυφή της λίστας.

3.9 Διαγράμματα Ροής

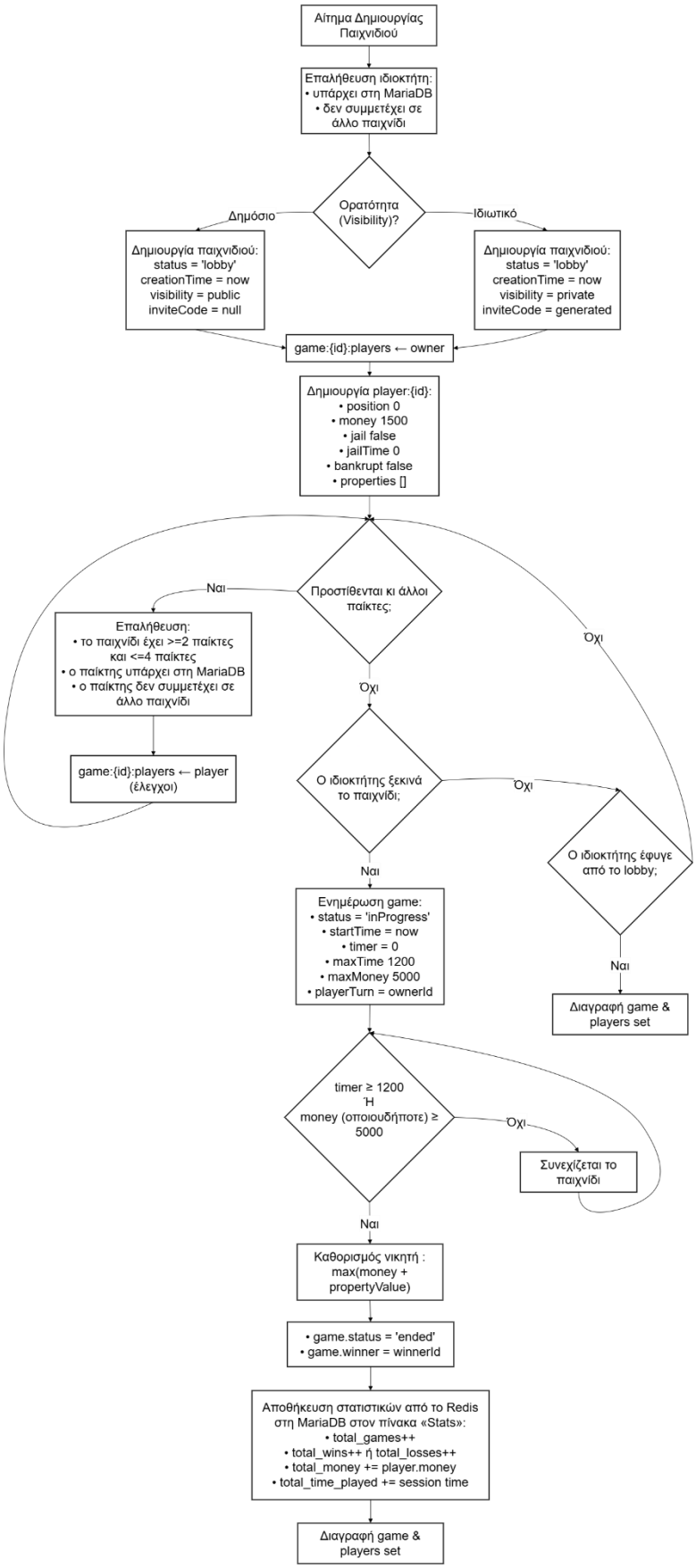
Για να γίνουν πλήρως κατανοητοί οι κανόνες και οι εσωτερικοί μηχανισμοί του Sindopoly, δεν αρκεί μια στατική περιγραφή των οντοτήτων και των κλάσεων. Τα διαγράμματα ροής αποτελούν το γραφικό «βήμα-βήμα» αποτύπωμα της συμπεριφοράς του συστήματος, δείχνοντας ποιο κομμάτι λογικής εκτελείται, πότε λαμβάνεται απόφαση και ποια εναλλακτικά μονοπάτια υπάρχουν. Στην υποενότητα αυτή παρουσιάζονται και αναλύονται δύο κομβικά διαγράμματα ροής (flow charts):

- το Game Flow Chart, που χαρτογραφεί ολόκληρο τον κύκλο ζωής μιας παρτίδας, από τη δημιουργία του lobby έως τον αυτόματο τερματισμό και την αποθήκευση στατιστικών, και
- το Turn Flow Chart, που εμβαθύνει στην εκτέλεση ενός μεμονωμένου γύρου ενός παίκτη, αποδίδοντας με σαφήνεια τις επιμέρους φάσεις (ρίψη ζαριών, μετακίνηση, ενέργεια τετραγώνου, έλεγχος χρεοκοπίας, κλπ.).

Μέσα από αυτά τα διαγράμματα ο αναγνώστης αποκτά μια οπτική και ακριβή αντίληψη του «πως» λειτουργεί το παιχνίδι εσωτερικά, διευκολύνοντας τόσο τον έλεγχο ορθότητας όσο και την προσθήκη νέων κανόνων στο μέλλον.

3.9.1 Ροή Παιχνιδιού

Στο Σχήμα 3.5 εμφανίζεται το Διάγραμμα Ροής ενός Παιχνιδιού (Game Flow Chart). Το διάγραμμα απεικονίζει βήμα-βήμα την πορεία μιας παρτίδας από τη στιγμή που ο ιδιοκτήτης πατά «Δημιουργία Παιχνιδιού» μέχρι την τελική καταγραφή στατιστικών. Η παρακάτω περιγραφή ακολουθεί τις λογικές φάσεις του σχήματος και έχει εμπλουτιστεί με κάποιες επιπλέον πληροφορίες για την πλήρη ανάλυση του σχήματος.



Σχήμα 3.5: Διάγραμμα Ροής Παιχνιδιού (Game Flow Chart).

Η διαδικασία εκκινείτε με την αίτηση δημιουργίας παιχνιδιού. Ο server επαληθεύει καταρχάς ότι ο αιτών υφίσταται στη μόνιμη βάση (PLAYER) και δεν συμμετέχει ήδη σε άλλη παρτίδα. Ακολούθως ζητείται η παράμετρος ορατότητας. Εάν επιλεγεί δημόσιο lobby, το πεδίο «visibility» καταχωρείται ως «public», ενώ σε ιδιωτικό lobby το πεδίο «visibility» καταχωρείται ως «private» και παράγεται ένας μοναδικός κωδικός πρόσκλησης (inviteCode).

Ο server δημιουργεί τότε το αντικείμενο «GAME_HASH» στο Redis με κατάσταση «lobby» και προσθέτει τον ιδιοκτήτη στο σύνολο «GAME_PLAYERS_SET». Παράλληλα δημιουργεί αντίστοιχο «PLAYER_HASH» με αρχικές τιμές: θέση 0, λεφτά 1500€ (με το αντίστοιχο «νόμισμα» του παιχνιδιού), φυλάκιση και χρεοκοπία ως FALE, καθαρή λίστα ιδιοκτησιών.

Στο στάδιο συγκέντρωσης παικτών το lobby παραμένει ανοιχτό μέχρι να συγκεντρωθούν τουλάχιστον δύο και το πολύ τέσσερις συμμετέχοντες. Κάθε νέα επιτυχής αίτηση προστίθεται στο ίδιο σύνολο παικτών. Αν ο ιδιοκτήτης το επιθυμεί, έχει τη δυνατότητα άμεσης εκκίνησης όταν πληρούνται οι ελάχιστες προϋποθέσεις. Αν ο ιδιοκτήτης εγκαταλείψει το lobby, τότε το παιχνίδι ακυρώνεται και οι πόροι της συγκεκριμένης παρτίδας απελευθερώνονται.

Με την έναρξη της παρτίδας, ο server μεταβάλλει το status σε «inProgress», αρχικοποιεί τη μεταβλητή «startTime» και μηδενίζει τον μετρητή «timer». Επίσης θέτει κάποια πολύ συγκεκριμένα όρια που αφορούν τη λήξη της παρτίδας: μέγιστη διάρκεια 1200 δευτερολέπτων και μέγιστο ποσό χρημάτων 5000€. Ο παίκτης-ιδιοκτήτης παίζει αναγκαστικά πρώτος. Η συνέχιση της παρτίδας παρακολουθείται από μια χρονικά προγραμματισμένη διεργασία, η οποία αυξάνει περιοδικά τον timer και ελέγχει αν έφτασε κάποιος παίκτης το παραπάνω χρηματικό όριο, καθώς και το χρονικό όριο.

Όταν ικανοποιηθεί ένα από τα κριτήρια λήξης (χρονικό ή οικονομικό), η παρτίδα εισέρχεται στη φάση τερματισμού. Υπολογίζεται για κάθε ενεργό παίκτη το άθροισμα των χρημάτων του και της καθαρής αξίας των ιδιοκτησιών που κατείχε στο τέλος της παρτίδας. Η μέγιστη τιμή καθορίζει τον νικητή της παρτίδας. Το «GAME_HASH» ενημερώνεται με «status=ended» (το παιχνίδι έχει τελειώσει) και «winner=playerId» (ο τάδε παίκτης είναι ο νικητής του παιχνιδιού), ενώ καταγράφεται ένα γεγονός λήξης σε WebSocket ώστε όλοι οι παίκτες να ενημερωθούν συγχρονισμένα.

Έπειτα ακολουθεί η ανανέωση των στατιστικών των παικτών. Τα στοιχεία των παικτών από τη συγκεκριμένη παρτίδα που βρίσκονται στο Redis, προστίθενται στο σύνολο των στατιστικών στοιχείων τους που βρίσκονται στη MariaDB στον πίνακα «Stats». Συγκεκριμένα, Προστίθεται ένα παιχνίδι στα συνολικά παιχνίδια που έχει συμμετάσχει ο παίκτης, αναλόγως αν ήταν ο νικητής ή όχι τότε προστίθεται ένα παιχνίδι στις συνολικές νίκες ή ήττες του αντίστοιχα, και προστίθενται επίσης τα λεφτά που είχε στο τέλος αυτής της παρτίδας καθώς και ο χρόνος της παρτίδας στο συνολικό ποσό χρημάτων που έχει καταφέρει να μαζέψει και στο συνολικό χρόνο που έχει αφιερώσει σε όλες τις παρτίδες που έχει παίξει ως εκείνη τη στιγμή.

Τέλος, εκτελείται ένας καθαρισμός των πόρων. Συγκεκριμένα, τα κλειδιά «game:{id}» και «player:{id}» διαγράφονται, απελευθερώνοντας μνήμη. Με τον τρόπο αυτό κλείνει ο κύκλος ζωής της παρτίδας, διατηρώντας συγχρόνως συνέπεια ανάμεσα σε προσωρινά και μόνιμα δεδομένα.

3.9.2 Ροή Γύρου

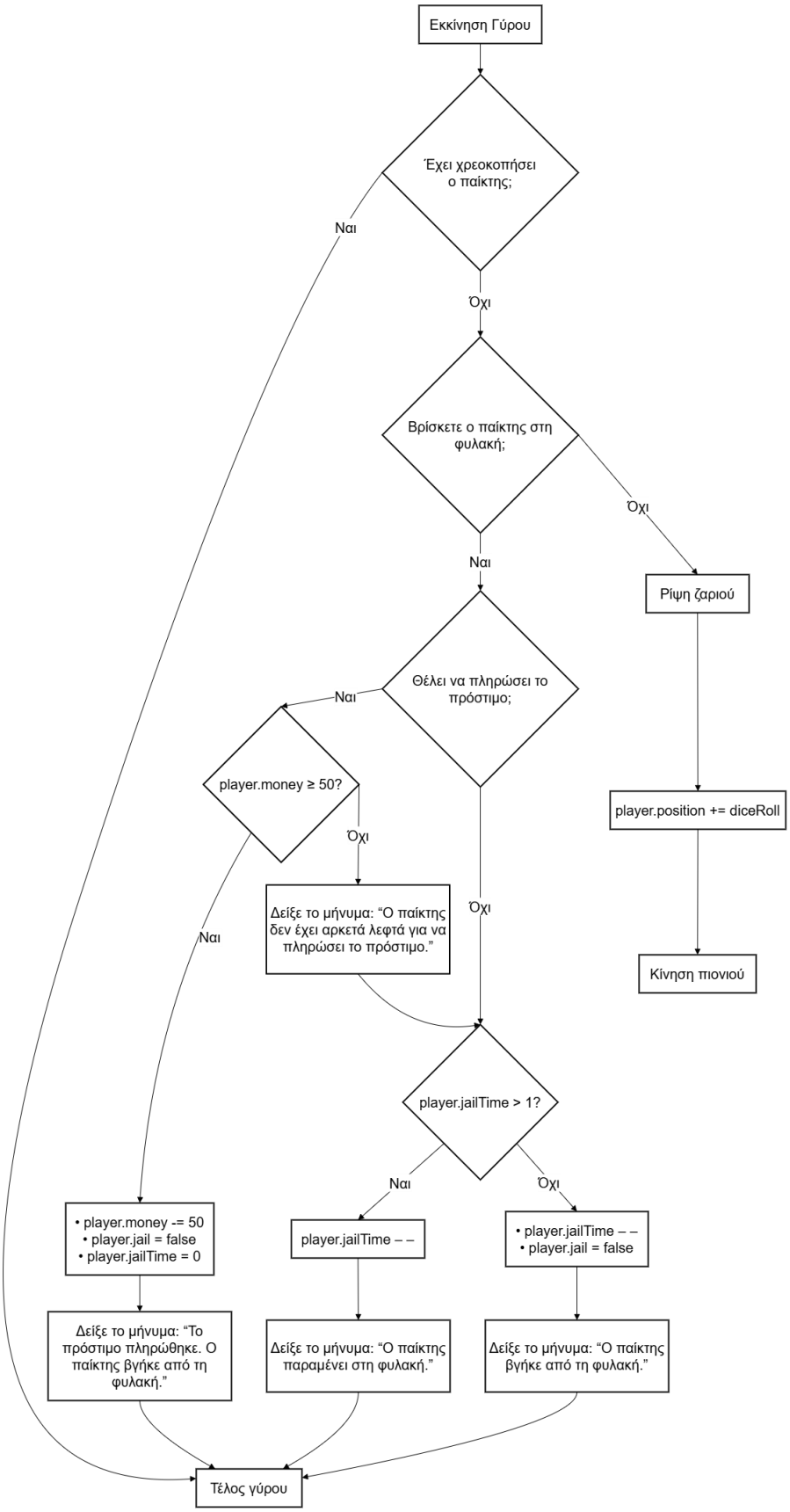
Ο γύρος αποτελεί τη μικρότερη, αλλά επαναλαμβανόμενη μονάδα λογικής του Sindopoly, και είναι το διάστημά του εκτελούνται όλοι οι κανόνες που αφορούν έναν συγκεκριμένο παίκτη. Επειδή το πλήρες διάγραμμα Turn Flow αποδείχθηκε ιδιαίτερα εκτενές, χωρίστηκε σε δύο διακριτά τμήματα, το καθένα από τα οποία αναλύεται σε δικιά του υποενότητα:

- **Μέρος Α**: από την έναρξη του γύρου έως και την αρχή της κίνησης του πιονιού.
- **Μέρος Β**: όλη η διαδικασία κίνησης, η οποία περιλαμβάνει τον έλεγχο του τετραγώνου, πληρωμές, αγορά ιδιοκτησίας, και την ενημέρωση των στοιχείων του παίκτη.

Η διάσπαση αυτή διευκολύνει τόσο την ανάγνωση όσο και τη συντήρηση, καθώς κάθε τμήμα μπορεί να εξεταστεί, να δοκιμαστεί και να τροποποιηθεί ανεξάρτητα.

3.9.2.1 Μέρος Α: Έναρξη, Έλεγχοι και Ρίψη Ζαριών

Παρακάτω στο Σχήμα 3.6 παρουσιάζεται το πρώτο κομμάτι (Μέρος Α) του Διαγράμματος Ρόης ενός Γύρου (Turn Flow Chart).



Σχήμα 3.6: Διάγραμμα Ροής Γύρου (Μέρος Α).

Με την έναρξη ενός γύρου, το σύστημα ανακτά τα δεδομένα της τρέχουσας κατάστασης του παίκτη και προβαίνει σε έλεγχο χρεοκοπίας. Εάν το υπόλοιπό του έχει ήδη καταστεί αρνητικό και έχει μαρκαριστεί ως «bankrupt», ο γύρος τερματίζεται αμέσως. Ο μηχανισμός μετάδοσης συμβάντων ενημερώνει τους υπόλοιπους συμμετέχοντες και ο δείκτης σειράς προωθείται στον επόμενο παίκτη.

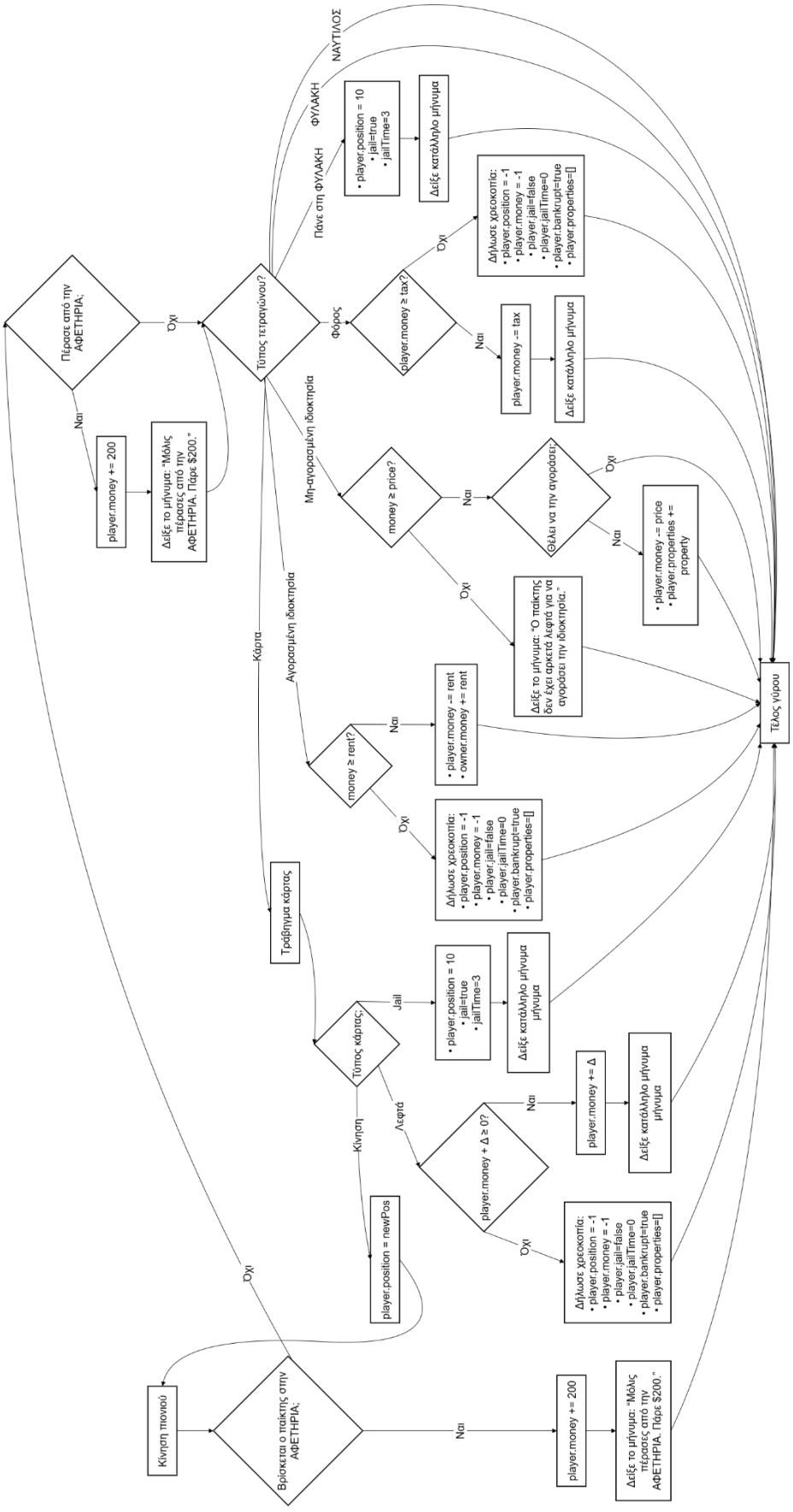
Σε διαφορετική περίπτωση ακολουθεί έλεγχος φυλάκισης. Αν ο παίκτης δεν βρίσκεται στη φυλακή, μεταβαίνει κατευθείαν στο στάδιο ρίψης ζαριών. Εάν όμως το πεδίο «jail» είναι ενεργό, ενεργοποιείται ένας κλάδος χειρισμού επιλογών: ο πελάτης καλείται να αποφασίσει αν επιθυμεί να καταβάλει το προβλεπόμενο πρόστιμο των 50€. Σε περίπτωση θετικής απόκρισης, το σύστημα επαληθεύει την επάρκεια των χρημάτων του. Αν τα χρήματα του αρκούν, το ποσό των 50€ αφαιρείται από εκεί, τα πεδία «jail» και «jailTime» μηδενίζονται, και τελειώνει ο γύρος του παίκτη. Αν το υπόλοιπο είναι ανεπαρκές, αποστέλλεται σχετικό μήνυμα και ο παίκτης παραμένει στη φυλακή. Σε αυτή την περίπτωση που δεν καταφέρνει να πληρώσει, ή εναλλακτικά εάν επιλέξει να μην πληρώσει, το «jailTime» μειώνεται κατά 1, εμφανίζεται κατάλληλο μήνυμα, και ο γύρος τελειώνει. Όταν το υπόλοιπο μηδενιστεί, ο παίκτης απελευθερώνεται αυτόματα χωρίς χρηματικό αντίτιμο, εμφανίζεται κατάλληλο μήνυμα, και ο γύρος τελειώνει.

Μόλις διασφαλιστεί ότι ο παίκτης είναι ελεύθερος, εκτελείται η ρίψη των ζαριών. Το άθροισμα προστίθεται στο position και επομένως μετακινείται ο παίκτης. Στη συνέχεια ενεργοποιείται η οπτική μετακίνηση του πιονιού.

Με την ολοκλήρωση των ελέγχων και με την έναρξη της μετακίνησης λήγει το πρώτο τμήμα του Turn Flow.

3.9.2.2 Μέρος Β: Μετακίνηση, Ενέργειες Τετραγώνου και Τερματισμός

Παρακάτω στο Σχήμα 3.7 παρουσιάζεται το δεύτερο κομμάτι (Μέρος Β) του Διαγράμματος Ρόης ενός Γύρου (Turn Flow Chart).



Σχήμα 3.7: Διάγραμμα Ροής Γύρου (Μέρος Β).

Το δεύτερο τμήμα του διαγράμματος ροής ενός γύρου συνεχίζει από τη μετακίνηση του πιονιού και αποτυπώνει όλες τις καταστάσεις που μπορούν να προκύψουν μέχρι την ολοκλήρωση του γύρου. Σκοπός του είναι να τεκμηριώσει πώς ο κινητήρας παιχνιδιού (game-engine) μεταφράζει το «πού κατέληξε ο παίκτης» σε οικονομικές συνέπειες, πιθανές κυρώσεις ή αγορές ιδιοκτησίας.

Η εκτέλεση αρχίζει με έλεγχο αν ο παίκτης πέρασε ή προσγειώθηκε στην ΑΦΕΤΗΡΙΑ. Σε κάθε διέλευση καταχωρείται πίστωση 200 € και αποστέλλεται ενημερωτικό μήνυμα. Η πίστωση πραγματοποιείται πριν από οτιδήποτε άλλο, ώστε τυχόν επόμενες χρεώσεις (φορολογία, ενοίκια) να υπολογιστούν στο αναθεωρημένο υπόλοιπο.

Στη συνέχεια το σύστημα ταυτοποιεί τον τύπο του τετραγώνου στο οποίο βρέθηκε ο παίκτης. Αν το τετράγωνο ανήκει στις κατηγορίες «Κάρτα Απόφασης» ή «Κάρτα Εντολής», ανακτάται η κορυφαία εγγραφή από την ουρά «CARD_DECK_LIST» και εφαρμόζεται το αποτέλεσμά της. Οι κάρτες χρηματικού χαρακτήρα μεταβάλλουν το υπόλοιπο του παίκτη. Όταν η μεταβολή καταλήγει σε αρνητικό αποτέλεσμα, ενεργοποιείται άμεσα ο κλάδος χρεοκοπίας. Οι κάρτες κίνησης τοποθετούν τον παίκτη στο προκαθορισμένο τετράγωνο που υποδεικνύει η κάρτα και γίνεται ξανά έλεγχος αν ο παίκτης πέρασε ή προσγειώθηκε στην ΑΦΕΤΗΡΙΑ, για να του δοθούν τα 200€ αν χρειαστεί. Η κάρτα φυλακής τοποθετεί τον παίκτη στο τετράγωνο 10 (ΦΥΛΑΚΗ), θέτει τον δείκτη φυλάκισης σε αληθές και τον μετρητή jailTime στην τιμή 3, δείχνει κατάλληλο μήνυμα και δεν γίνεται έλεγχος σχετικά με την ΑΦΕΤΗΡΙΑ σε αυτή την περίπτωση.

Εάν ο παίκτης βρίσκεται σε τετράγωνο ιδιοκτησίας, ο κινητήρας εξετάζει αρχικά αν η ιδιοκτησία είναι ήδη καταχωρισμένη σε άλλο παίκτη. Στην περίπτωση αυτή υπολογίζει το αντίστοιχο ενοίκιο και επιχειρεί να κάνει τη συναλλαγή. Όταν το υφιστάμενο κεφάλαιο αρκεί, η συναλλαγή ολοκληρώνεται, αλλιώς ο παίκτης κηρύσσεται «bankrupt», οι ιδιοκτησίες του διαγράφονται από τα στοιχεία του και γίνονται διαθέσιμες για αγορά ξανά, κλπ. (ό,τι γίνεται πάντα σε περίπτωση χρεοκοπίας). Αντιθέτως, όταν το τετράγωνο είναι ελεύθερο, το παιχνίδι προσφέρει στον παίκτη τη δυνατότητα αγοράς. Η αγορά πραγματοποιείται μόνο όταν ο χρήστης επιλέξει θετικά και διαθέτει επαρκές υπόλοιπο χρημάτων, διαφορετικά η ιδιοκτησία συνεχίζει να παραμένει διαθέσιμη προς αγορά.

Για τετράγωνα κατηγορίας «Φόρος» αφαιρείται το ποσό του φόρου. Αν ο παίκτης δεν έχει αρκετά χρήματα, ενεργοποιείται για άλλη μια φορά ο ίδιος κλάδος χρεοκοπίας. Το τετράγωνο «ΠΑΝΕ ΣΤΗ ΦΥΛΑΚΗ» μετακινεί τον παίκτη στο τετράγωνο 10 (ΦΥΛΑΚΗ), θέτει τον δείκτη φυλάκισης σε αληθές και τον μετρητή jailTime στην τιμή 3, δείχνει κατάλληλο μήνυμα και δεν γίνεται έλεγχος σχετικά με την ΑΦΕΤΗΡΙΑ σε αυτή την περίπτωση . Ειδικά τετράγωνα όπως η «ΦΥΛΑΚΗ» ή η «ΝΑΥΤΙΛΟΣ» (το «Ελεύθερο Πάρκινγκ» στην κανονική Μονόπολη) δεν απαιτούν την πραγματοποίηση κάποιας ενέργειας, γίνεται απλώς «επίσκεψη» σε αυτά.

Σε κάθε κλάδο, το σύστημα στέλνει μήνυμα κατάστασης στους παίκτες ώστε η διεπαφή να παρουσιάσει το αποτέλεσμα σε πραγματικό χρόνο. Όταν ολοκληρωθούν όλες οι ενέργειες, ο γύρος κλείνει: ενημερώνεται ο δείκτης σειράς, και αποστέλλεται γεγονός λήξης μέσω WebSocket, επιτρέποντας στον επόμενο παίκτη να ξεκινήσει.

3.10 Επίλογος

Το παρόν κεφάλαιο ανέλυσε διεξοδικά τον τρόπο με τον οποίο η αρχική σύλληψη του Sindopoly μετουσιώθηκε σε μια τεκμηριωμένη αρχιτεκτονική πρόταση. Ξεκίνησε με τις λειτουργικές και μη-λειτουργικές απαιτήσεις, οριοθετώντας με ακρίβεια τόσο το «τι» πρέπει να προσφέρει το σύστημα όσο και το «πως» πρέπει να συμπεριφέρεται ως προς τις επιδόσεις, την ασφάλεια και την επεκτασιμότητα. Στη συνέχεια παρουσιάστηκε η μεθοδολογία C4, η οποία, κλιμακωτά, αποκάλυψε τη λογική δομή του

συστήματος. Η ανάλυση ολοκληρώθηκε με τις βασικές οντότητες που υπάρχουν στο σύστημα και το διάγραμμα σχέσεων-οντοτήτων, παρουσιάζοντας τη δομή της προσωρινής μνήμης στο Redis και της μόνιμης μνήμης στη MariaDB, καθώς και το «πως» λειτουργούνε και συνεργάζονται μεταξύ τους. Τα διαγράμματα ροής παιχνιδιού και γύρου έδωσαν μια πιο κατανοητή εικόνα στον τρόπο εκτέλεσης διεργασιών μέσα στο σύστημα. Η συνάρθρωση όλων αυτών των στοιχείων δημιουργεί ένα στιβαρό, αλλά ταυτόχρονα ευέλικτο, πλαίσιο υλοποίησης.

Στο επόμενο κεφάλαιο, Υλοποίηση Συστήματος, μεταβαίνουμε από τη θεωρητική αρχιτεκτονική στην πρακτική υλοποίηση.

Κεφάλαιο 4ο: Υλοποίηση Συστήματος

4.1 Εισαγωγή

Στο παρόν κεφάλαιο επιχειρείται η μετάβαση από το θεωρητικό υπόβαθρο του Κεφαλαίου 3 στην πρακτική υλοποίηση του Sindopoly. Εξετάζονται αναλυτικά τα τεχνολογικά στρώματα που απαρτίζουν την εφαρμογή. Το back-end σε Spring Boot, υπεύθυνο για τους κανόνες του παιχνιδιού και την ορθή τήρηση της λογικής του παιχνιδιού. Το front-end σε React + Three.js, το οποίο αποδίδει το τρισδιάστατο ταμπλό και δίνει στο χρήστη δυνατότητα αλληλεπίδρασης σε πραγματικό χρόνο. Τέλος, οι μηχανισμοί ολοκλήρωσης (CI/CD) που επιτρέπουν την αυτοματοποιημένη παράδοση νέων εκδόσεων.

Η ανάλυση εκκινεί από τη δομή του κώδικα και την οργάνωση φακέλων, ώστε να καταστεί σαφές πώς υλοποιούνται πρακτικά οι αρχές διαστρωμάτωσης και συμβολισμού που περιγράφηκαν προηγουμένως. Κατόπιν παρουσιάζονται οι κρίσιμες ροές του παιχνιδιού σε επίπεδο αλγορίθμων, δηλαδή η ρίψη ζαριών, η μετακίνηση, ο υπολογισμός ενοικίου, η χρεοκοπία και ο αυτόματος τερματισμός παρτίδας. Ιδιαίτερη έμφαση δίνεται στη συνεργασία Redis – MariaDB, η οποία εξασφαλίζει χαμηλή υστέρηση κατά τη διάρκεια της παρτίδας και ταυτόχρονα μόνιμη αποθήκευση στατιστικών των παιχνιδιών για μελλοντική ανάλυση.

Η ενότητα ολοκληρώνεται με την περιγραφή του κύκλου ζωής ανάπτυξης, από τα unit tests και τα integration tests, μέχρι και την εκτέλεση σε Docker Compose περιβάλλοντος. Με αυτόν τον τρόπο, ο αναγνώστης αποκτά πλήρη εικόνα του πως κάθε σχεδιαστική επιλογή παίρνει σάρκα και οστά στον πηγαίο κώδικα και πώς διασφαλίζεται ότι η πλατφόρμα παραμένει αξιόπιστη, επεκτάσιμη και εύκολα συντηρήσιμη.

4.2 Back-end Υλοποίηση

Η υλοποίηση της εσωτερικής λογικής του Sindopoly βασίζεται αποκλειστικά στο Spring Boot 3. Ο κώδικας οργανώθηκε έτσι ώστε να αντανακλά μία καθαρή και στρωματοποιημένη, αλλά ταυτόχρονα hexagonal, αρχιτεκτονική, και όχι το παραδοσιακό MVC (Model-View-Controller) μοντέλο. Στα επόμενα υποκεφάλαια παρουσιάζονται η φυσική διάταξη των πακέτων, οι κύριοι ρόλοι των κλάσεων, ο τρόπος διασύνδεσης των layers, οι προγραμματισμένες εργασίες, η ροή γεγονότων WebSocket και, τέλος, οι συμβάσεις στον κώδικα που διασφαλίζουν συντηρησιμότητα και ποιότητα.

4.2.1 Δομή Φακέλων & Packaging

Το project χωρίζεται σε σαφείς «τομείς ενδιαφέροντος». Στον φάκελο «config» συγκεντρώνονται αποκλειστικά beans ρύθμισης, όπως η κλάση WebSocketConfig που εκθέτει το endpoint /ws και ενεργοποιεί εσωτερικό STOMP broker, καθώς και η RedisConfig που παρέχει το έτοιμο, τυποποιημένο RedisTemplate. Τα REST controllers ζουν σε ξεχωριστό πακέτο και περιορίζονται στην αντιστοίχιση URLs σε DTOs (Data Transfer Objects - Αντικείμενα Μεταφοράς Δεδομένων). Η καθαρή επιχειρησιακή λογική τοποθετείται στον φάκελο «service/core», ενώ οι κλάσεις «ορχήστρωσης», όπως ο TurnService, βρίσκονται στον φάκελο «service/orchestrator», ώστε να ξεχωρίζει η καθαρή λογική του παιχνιδιού από τους μηχανισμούς συντονισμού.

Τα JPA entities (Απθετήρια JPA) φιλοξενούνται στον φάκελο «model», ακολουθώντας σύμβαση ονομασίας (naming convention) που ταυτίζεται με το σχήμα MariaDB (π.χ. Player, Square, κλπ.). Τα

repositories, κληρονομημένα από JPARepository, ζουν σε δικό τους πακέτο για να αποφευχθεί η τροποποίηση των service κλάσεων. Ο κώδικας που αλληλεπιδρά με το Redis συγκεντρώνεται αποκλειστικά στους GameStateService και PlayerStateService; έτσι, αν αλλάξει ο μηχανισμός in-memory αποθήκευσης, αρκεί η παρεμβολή νέου adapter σε ένα και μόνον σημείο.

4.2.2 Κύριες κλάσεις

Στον πυρήνα βρίσκεται η TurnService, η οποία είναι υπεύθυνη για το πλήρες σενάριο των γύρων. Λειτουργεί ως ένα application service, δηλαδή συγκεντρώνει τα αναγκαία aggregate roots (player, game) και συντονίζει την αλληλουχία βημάτων χωρίς να ενσωματώνει επιχειρησιακές λεπτομέρειες. Οι επιχειρησιακές λεπτομέρειες ανήκουν στα domain services.

Η δημόσια μέθοδός του, playTurn(…), δέχεται ένα αμετάβλητο DTO με τα αναγνωριστικά της παρτίδας και του παίκτη και εκτελεί πέντε στάδια:

- **Κεντρικοί έλεγχοι**. Με κλήσεις στη ValidationUtils βεβαιώνεται ότι ο γύρος αντιστοιχεί στον παίκτη που έχει σειρά και ότι η παρτίδα βρίσκεται σε κατάσταση inProgress. Επίσης ελέγχεται αν ο παίκτης είναι ήδη χρεοκοπημένος (άρα δεν μπορεί να παίξει άλλο) ή στη φυλακή (άρα είτε πληρώνει το πρόστιμο και παίζει στον επόμενο γύρο, είτε περιμένει μέχρι να βγει μετά το πέρασμα μερικών γύρων). Και στις δύο τελευταίες περιπτώσεις, τελειώνει ο γύρος του εκεί. Αλλιώς προχωράει το σύστημα στα επόμενα στάδια.
- **Μετακίνηση**. Μέσω της μεθόδου diceSvc.roll(…) παράγονται δύο αριθμοί, ένας για κάθε ζάρι, και έπειτα η MovementService.move(…) παράγει ένα ζεύγος (newPos, passedGo). Εάν το δεύτερο κομμάτι του ζεύγους είναι αληθές, ενημερώνει άμεσα το υπόλοιπο των χρημάτων του παίκτη προσθέτοντας 200 €.
- **Ενέργεια τετραγώνου**. Η SquareActionService.execute(…) εφαρμόζει τις απαιτούμενες ενέργειες του τετραγώνου στο οποίο βρίσκεται πλέον ο παίκτης, βάσει του SquareType.
- **Χρεοκοπία**. Μετά από κάθε οικονομική συναλλαγή το BankruptcyService.check(…) επαληθεύει αν το υπόλοιπο παραμένει μη-αρνητικό (θετικό ή μηδέν). Σε περίπτωση αρνητικού, εκτελεί κάνει ξανά διαθέσιμες προς αγορά τις ιδιοκτησίες του, ενημερώνει το flag «bankrupt» και θέτει τον παίκτη εκτός παιχνιδιού.
- **Δημοσίευση γεγονότος**. Στο τέλος συντίθεται ένα TurnResponse DTO και αποστέλλεται στους παίκτες

Στον Κώδικας Β βλέπουμε ένα TurnResponse record, καθώς και την αποστολή του.

Κώδικας Β: TurnResponse Record

```
public record TurnResponse(
        int dice,
        int newPosition,
        int money,
        boolean passedGo,
        boolean wasInJail,
        boolean wentBankrupt,
```

```
        String tileMessage) {}

/* ──────────────── send over WebSocket & return ──────────────── */
private TurnResponse publish(int gameId, TurnResponse resp) {
        broker.convertAndSend("/topic/game/" + gameId, resp);
        return resp;
}
```

Παράλληλα με την TurnService λειτουργεί η GameStateService, η οποία επωμίζεται το ρόλο του φύλακα της μεταβλητής της παρτίδας. Όλες οι αλλαγές σε game:* hashes, π.χ. status, timer, playerTurn, διέρχονται από αυτήν. Στην ολοκλήρωση μιας παρτίδας η μέθοδος closeGame(…) διαβάζει τα προσωρινά υπόλοιπα, μεταμορφώνει τα δεδομένα σε StatsEntity και επιτελεί ένα γραπτό flush() προς τη MariaDB.

Η τυχαιότητα των ζαριών παρέχεται από τη γεννήτρια RandomIntGenerator (functional interface). Η προεπιλεγμένη κλάση SecureRandomGenerator καλύπτει τοπικές εκτελέσεις σε περιβάλλοντα χωρίς πρόσβαση στο διαδίκτυο. Για την κύρια παραγωγή όμως χρησιμοποιείται η RandomOrgGenerator, η οποία αντλεί παρτίδες 1000 (ενός χιλιάδων) ρίψεων και τις αποθηκεύει σε λίστα στο Redis. Η DiceEntropyRefillJob επανεκκινεί αυτόματα όταν το απόθεμα πέσει κάτω από 200 τιμές, ώστε το εξωτερικό API να μην προσθέσει κάποια καθυστέρηση στους γύρους του παιχνιδιού.

Ο σχεδιασμός βασίζεται στην τεχνική της έγχυσης εξάρτησης (dependency injection). Κάθε service ορίζεται ως «@Service» και λαμβάνει τα απαιτούμενα beans στον constructor του. Έτσι, κατά τα unit tests, τα core-services μπορούν να αντικατασταθούν με mocks (προσομοιώσεις) ή in-memory υλοποιήσεις. Επιπλέον, οι public API μέθοδοι είναι συναλλακτικές (@Transactional) όπου απαιτείται, ώστε οι μεταβολές στη MariaDB να ολοκληρώνονται ατομικά με τα αντίστοιχα updates της Redis.

4.2.3 Διαστρωμάτωση

Ο κώδικας ακολουθεί τέσσερα αυστηρά διαχωρισμένα επίπεδα. Το Controller layer υπηρετεί αποκλειστικά HTTP mapping και την βασική επικύρωση, χωρίς να εκτελεί λογική ούτε να ξεκινά συναλλαγές. Το Orchestrator layer συντονίζει πολλαπλά core-services και αναλαμβάνει side-effects όπως push σε WebSocket και ενημέρωση των υπαρχόντων μετρητών. Το Core-service layer φιλοξενεί μόνο καθαρούς και απομονωμένους κανόνες, κάτι που επιτρέπει γρήγορα unit tests. Τέλος, το Persistence/Gateway layer εκθέτει τις μόνες επίσημες διεπαφές για τη MariaDB και το Redis, αποτρέποντας τη «διαρροή» τεχνικών λεπτομερειών προς τα ανώτερα στρώματα.

4.2.4 Προγραμματισμένες εργασίες

Δύο κρίσιμες εργασίες λειτουργούν στο παρασκήνιο. Ο GameLoopScheduler εκτελείτε κάθε δύο (2) δευτερόλεπτα, αυξάνει τον χρονομετρητή όλων των ενεργών παρτίδων, ανιχνεύει (AFK) παίκτες που είναι ανενεργοί για περισσότερα από εξήντα δευτερόλεπτα και αξιολογεί τα κριτήρια αυτόματου τερματισμού. Στον Κώδικας C παρουσιάζεται ένα κομμάτι του GameLoopScheduler που απεικονίζει την παραπάνω λογική σε κώδικα.

Κώδικας C: Μέρος του GameLoopScheduler

```
@Scheduled(fixedRate = 2000)        // every 10 s
public void pollTurnTimers() {
   …
   for (String key : keys) {
   …
      // ------------- timeout reached -------------
      String stalled = (String) redisTemplate.opsForHash().get(key, "playerTurn");
      System.out.println("Skipping stalled player " + stalled + " in game " + gameId);

      /* mark player AFK so UI can show it */
      gameStateService.setPlayerField(Integer.parseInt(stalled), "afk", "true");

      // rotate to next player
      gameStateService.forceRotateTurn(gameId, stalled);

      // reset age
      redisTemplate.opsForHash().put(key, "turnAge", "0");
   }
}
```

Ο DiceEntropyRefillJob εκτελείται ανά τριάντα (30) δευτερόλεπτα. Αρχικά ελέγχει αν υπάρχουν παραπάνω από 200 τυχαίοι αριθμοί στην προκαθορισμένη λίστα στο Redis. Αν ναι, τότε δεν πραγματοποιεί κάποια ενέργεια, καθώς θεωρείται πως υπάρχει ακόμα «αρκετό απόθεμα» για τις επόμενες ρίψεις ζαριών μέχρι το επόμενο «refill». Αν όχι, τότε θα πραγματοποιήσει μια HTTP GET κλήση στο API της πλατφόρμα random.org για να ζητήσει χίλιες (1000) τυχαίες («true random») ακέραιες τιμές από 1 έως 6 που θα αποθηκεύσει στη λίστα, αφού κάνει τις κατάλληλες μετατροπές τύπου, προκειμένου να χρησιμοποιηθούν ως οι επόμενες ζαριές. Στον Κώδικας D παρουσιάζεται ένα κομμάτι του DiceEntropyRefillJob που απεικονίζει την παραπάνω λογική σε κώδικα.

Κώδικας D: Μέρος του DiceEntropyRefillJob

```
@Scheduled(fixedRate = 30000)           // every 30 s
public void refill() {
   Long size = redis.opsForList().size(KEY);
   if (size != null && size > 200) return;    // still plenty left

   // request 1000 integers 1-6
```

```
    String body = web.get()
        .uri("/integers/?num=1000&min=1&max=6&col=1&base=10&format=plain&
rnd=new")
        .retrieve()
        .bodyToMono(String.class)
        .block();

    if (body == null) return;

    List<Integer> vals = Arrays.stream(body.trim().split("\\s+"))
        .map(Integer::parseInt).toList();
    rng.pushBatch(vals);
}
```

4.2.5 Δίαυλος WebSocket

Η ζωντανή ενημέρωση των πελατών βασίζεται σε WebSocket STOMP με fallback SockJS. Κάθε παρτίδα αντιστοιχεί σε ένα topic που έχει τη μορφή «/topic/game/{id}». Μετά τη λογική του γύρου, η TurnService διαμορφώνει ένα αντικείμενο TurnResponse (immutable DTO) και το αποστέλλει στο topic. Ο browser λαμβάνει το μήνυμα, ενημερώνει το state και δρομολογεί την κίνηση του Three.js πιονιού χωρίς επαναφόρτωση της σελίδας. Στον Κώδικας Ε παρουσιάζεται ένα κομμάτι κώδικα του back-end που απεικονίζει την παραπάνω λογική.

Κώδικας Ε: Παράδειγμα WebSocket STOMP topic

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    /* broker for messages FROM server → clients */
    @Override
    public void configureMessageBroker(@NonNull MessageBrokerRegistry
config) {
        config.enableSimpleBroker("/topic");   // in-memory broker
        config.setApplicationDestinationPrefixes("/app"); // if clients ever
send
    }

    /* STOMP handshake endpoint (SockJS fallback for browsers) */
    @Override
```

```
   public void registerStompEndpoints(@NonNull StompEndpointRegistry
registry) {
      registry.addEndpoint("/ws")            // main endpoint
              .setAllowedOrigins("*")        // adjust for prod
              .withSockJS();                 // SockJS fallback
   }
}
```

4.2.6 Συμβάσεις Ποιότητας

Κάθε DTO δηλώνεται ως record, επιβάλλοντας την αμεταβλητότητα (immutability) και την αυτόματη υλοποίηση equals/hashCode. Τα κλειδιά Redis ονομάζονται βάσει της «snake case» σύμβασης ώστε να διευκολύνουν την αποσφαλμάτωση (debugging). Η snake case είναι μια σύμβαση ονομασίας όπου οι λέξεις διαχωρίζονται με κάτω παύλες (_) αντί για κενά («SPACE») και τα γράμματα τους είναι όλα πεζά [54]. Για το υπόλοιπο back-end, οποιαδήποτε κλάση, μεταβλητή, συνάρτηση ή μέθοδος, ονομάζεται βάσει της «camel case» σύμβασης. Η camel case σύμβαση είναι ένας τρόπος για να γράφονται φράσεις χωρίς κενά, όπου το πρώτο γράμμα κάθε λέξης είναι κεφαλαίο, εκτός από το πρώτο γράμμα της ολόκληρης σύνθετης λέξης, το οποίο μπορεί να είναι είτε κεφαλαίο είτε μικρό [55].

Η επικύρωση κανόνων γίνεται βάσει του μοτίβου «fail-fast». Ένα σύστημα που αναφέρει άμεσα οποιαδήποτε περίσταση που μπορεί να υποδηλώνει βλάβη στη διεπαφή του λέγεται σύστημα fail-fast [56]. Στην προκειμένη περίπτωση, μόλις ανιχνευτεί κάποια παράβαση «ρίχνεται» (throw) μία εξαίρεση IllegalArgumentException, την οποία λαμβάνει ο χειριστής GlobalExceptionHandler και τη μετατρέπει σε μορφή JSON απόκριση-σφάλματος με ένα σαφές errorCode και message.

Στον Κώδικας F παρουσιάζεται ένα κομμάτι κώδικα του back-end που απεικονίζει την παραπάνω λογική, δηλαδή δύο περιπτώσεις που υπάρχει κάποια παραβίαση και χρειάζεται να «ριχθεί» κάποια εξαίρεση, και τη χρήση της camel case σύμβασης για την ονομασία των συναρτήσεων.

Κώδικας F: Μέρος του ValidationUtils – Έλεγχοι Εξαιρέσεων

```
/** Ensure the player exists */
public void validatePlayerExists(int playerId) {
   if (!playerRepo.existsById(playerId)) {
      throw new IllegalArgumentException("Player with ID " + playerId + "
does not exist.");
   }
}

/** Ensure it is this player's turn */
public void ensurePlayerTurn(int gameId,
                             int playerId,
                             Map<String,String> gameHash) {
```

```
    String current = gameHash.get("playerTurn");
    if (!String.valueOf(playerId).equals(current)) {
        throw new IllegalStateException("Not your turn.");
    }
}

/** Ensure game is still in lobby state */
public void ensureLobby(String status) {
    if (!"lobby".equalsIgnoreCase(status))
        throw new IllegalStateException("Game already started.");
     }
}
```

4.3 Υλοποίηση Κρίσιμων Αλγορίθμων

Ένα διαδικτυακό παιχνίδι που τρέχει σε πραγματικό χρόνο κρίνεται τελικά από την ορθότητα και την αποδοτικότητα των αλγορίθμων του. Στην ενότητα αυτή παρουσιάζονται πέντε πυρήνες λογικής οι οποίοι, συλλογικά, εγγυώνται ότι η παρτίδα εξελίσσεται χωρίς ασυνέπειες, ότι οι οικονομικές συναλλαγές είναι ακριβείς και ότι η εμπειρία παραμένει «ζωντανή» ακόμη κι όταν οι συνδέσεις των παικτών δεν είναι. Κάθε αλγόριθμος υλοποιείται σε ξεχωριστό core-service, μετατρέποντας τους κανόνες παιχνιδιού σε ελέγξιμο και επαναχρησιμοποιήσιμο κώδικα.

4.3.1 Εκτέλεση Γύρου

Η μέθοδος TurnService.playTurn(…) λειτουργεί με έναν κατευθυντικό ρόλο πιο πολύ. Ξεκινάει με την απλή επαλήθευση «είναι όντως η σειρά του παίκτη;». Αν όχι, ρίχνεται εξαίρεση κανόνα και η κλήση τερματίζεται προκαταβολικά. Εν συνεχεία ο παίκτης περνά από το υποσύνολο ελέγχων φυλάκισης. Οι επιλογές πληρωμής προστίμου, παραμονής ή αυτόματης αποφυλάκισης δρομολογούνται βάσει του υπολειπόμενου jailTime του παίκτη. Η ρίψη ζαριών πραγματοποιείται μέσω κλήσης του RandomIntGenerator. Αφού υπολογιστεί η νέα θέση, το συμβάν «Move Pawn» μεταδίδεται σε όλους τους πελάτες. Πρώτα γίνεται η ενημέρωση στο Redis, και κατόπιν με WebSocket, ώστε να αποτραπούν race conditions όταν υπάρχουν γρήγορες σειρές γεγονότων. Ο κύκλος κλείνει με έναν απλό έλεγχο χρεοκοπίας και, εφόσον ο παίκτης παραμένει ενεργός, μετακύλιση του δείκτη σειράς στον επόμενο.

Παρακάτω παρουσιάζονται δύο κομμάτια κώδικα ως παραδείγματα της παραπάνω λογικής. Στον Κώδικας G απεικονίζεται η λογική πίσω από την υπηρεσία MovementService που χρησιμοποιεί η TurnService για την κίνηση των παικτών, με τη μέθοδο MovementResult.

Κώδικας G: Μέρος της υπηρεσίας MovementService

```
public MovementResult move(int currentPos, int dice) {
    int newPos = (currentPos + dice) % 40;      // 40 positions
    boolean passedGo = currentPos + dice >= 40; // went through the start?
    return new MovementResult(newPos, passedGo);
```

```
}
```

Στον Κώδικας Η απεικονίζεται η λογική πίσω από την υπηρεσία BankruptcyService που χρησιμοποιεί η TurnService για τον έλεγχο χρεοκοπίας.

Κώδικας Η: Μέρος της υπηρεσίας BankruptcyService

```
/** true if money ≤ 0 */
public boolean isBankrupt(int money) {
   return money <= 0;
}


/** releases all properties whose owner == playerId */
public void releaseProperties(int playerId) {
   Set<String> keys = redis.keys("square:*");
   if (keys.isEmpty()) return;
      for (String k : keys) {
         String owner = hashOps.get(k,"owner");
         if (String.valueOf(playerId).equals(owner)) {
            hashOps.delete(k,"owner");
         }
      }
   }
}
```

4.3.2 Υπολογισμός Ενοικίων

Η υπηρεσία SquareActionService αποφασίζει το τι ενέργειες θα πραγματοποιηθούν βάσει του τετραγώνου στο οποίο προσγειώνεται ο παίκτης. Για τα τετράγωνα ιδιοκτησίας καλεί την κλάση handleProperty(…), η οποία ελέγχει αν ο παίκτης είναι ο ιδιοκτήτης. Αν ναι, δεν γίνεται καμία χρέωση. Διαφορετικά, ελέγχει αν το τετράγωνο ανήκει σε κάποιον άλλον παίκτη. Αν όχι, τότε ρωτάει τον παίκτη αν θέλει να την αγοράσει. Αν ναι, τότε υπολογίζει το ενοίκιο με βάση τρία κριτήρια τα οποία έχουν κι αυτά από ένα υπο-κριτήριο το καθένα:

- Αν είναι απλή ιδιοκτησία. Αν ναι, τότε ανήκουν στον ιδιοκτήτη και όλα τετράγωνα με το ίδιο χρώμα;
- Αν είναι Υπηρεσία Κοινής Ωφέλειας (Utility). Αν ναι, τότε ανήκουν στον ιδιοκτήτη και οι δυο Υπηρεσίες Κοινής Ωφέλειας που υπάρχουν στο ταμπλό;
- Αν είναι Κιόσκι (Τρένο στην κανονική Μονόπολη). Αν ναι, πόσα τρένα ανήκουν στον ιδιοκτήτη;

Κάθε ιδιοκτησία ενημερώνει το δικό της set κατά την αγορά ή πώληση.

Στον Κώδικας I παρουσιάζεται ως παράδειγμα ένα κομμάτι κώδικα της handleProperty(…), έτσι όπως υπάρχει στον κώδικα της εφαρμογής, το οποίο κάνει τους ελέγχους και τις ενέργειες σε περίπτωση που ένας παίκτης προσγειωθεί σε ένα τετράγωνο ιδιοκτησίας δεν ανήκει σε κάποιον άλλον παίκτη, αλλά ούτε σε αυτόν.

Κώδικας I: Μέρος της μεθόδου handleProperty(…)

```
/* (3) Pay rent to another player */
int rent;
if (sq.getType() == SquareType.UTILITY) {
      int dice = readLastDice(gameId);             // use the method arg
      rent = ownsBothUtilities(owner) ? dice * 10 : dice * 4;
} else if (sq.getType() == SquareType.TRAIN) {
      rent = rentForTrainsOwned(owner);
} else {
      rent = maybeDoubleForFullSet(sq, owner);
}
return processRentPayment(playerId, owner, rent);
```

4.3.3 Τερματισμός Παρτίδας

Ο κύκλος ζωής κάθε παιχνιδιού παρακολουθείται από τον GameLoopScheduler, ο οποίος εκτελείται κάθε δέκα δευτερόλεπτα και καλεί τη μέθοδο checkEndCondition(int gameId) του GameStateService. Η κλήση αυτή γίνεται μόνο σε παρτίδες των οποίων το status είναι InProgress, έτσι ώστε τα λόμπι που ακόμη περιμένουν παίκτες να μην επιβαρύνονται με περιττούς ελέγχους. Στον Κώδικας J παρουσιάζεται αυτή η μέθοδος.

Κώδικας J: Η μέθοδος checkEndCondition(int gameId)

```
public boolean checkEndCondition(int gameId) {
      String gameKey = "game:" + gameId;
      String timerStr = hashOps.get(gameKey, "timer");
      if (timerStr != null && Integer.parseInt(timerStr) >= 1200) {
         endGame(gameId);
         return true;
      }
      Set<String> playerIds = setOps.members(gameKey + ":players");
      if (playerIds != null) {
         for (String pid : playerIds) {
            String moneyStr = (String)
redisTemplate.opsForHash().get("player:" + pid, "money");
```

```
            if (moneyStr != null && Integer.parseInt(moneyStr) >= 5000) {
               endGame(gameId);
               return true;
            }
         }
      }
      return false;
}
```

Ο αλγόριθμος ελέγχει διαδοχικά δύο ανεξάρτητα κριτήρια λήξης, ένα χρονικό όριο και ένα χρηματικό όριο. Στην ουσία καμία παρτίδα δεν πρέπει να διαρκεί περισσότερο από είκοσι λεπτά (1200 δευτερόλεπτα). Εναλλακτικά, εάν ένας παίκτης συσσωρεύσει 5000€ ή περισσότερο, θεωρείται ότι έχει αποκτήσει οικονομική υπεροχή και η παρτίδα τερματίζεται άμεσα.

4.3.4 Καθορισμός Νικητή

Όταν μία παρτίδα τερματίζεται, είτε λόγω συμπλήρωσης του διαθέσιμου χρόνου, είτε επειδή κάποιος παίκτης έφθασε το ανώτατο χρηματικό όριο, η GameStateService καλεί τη μέθοδο determineGameWinner(int gameId) για να αναδείξει τον νικητή. Ο αλγόριθμος βασίζεται σε ένα απλό, αλλά ακριβές, οικονομικό κριτήριο: το συνολικό καθαρό κεφάλαιο κάθε παίκτη, δηλαδή μετρητά + αξία ιδιοκτησιών. Αυτός παρουσιάζεται στον Κώδικας Κ.

Κώδικας Κ: Η μέθοδος determineGameWinner(int gameId)

```
public String determineGameWinner(int gameId) {
      String playersKey = "game:" + gameId + ":players";
      Set<String> playerIds = setOps.members(playersKey);
      if (playerIds == null) return null;
      String winner = null;
      int maxTotalValue = -1;
      for (String pid : playerIds) {
         String playerKey = "player:" + pid;
         int money = Integer.parseInt((String)
redisTemplate.opsForHash().get(playerKey, "money"));
         String propertiesJson = (String)
redisTemplate.opsForHash().get(playerKey, "properties");
         List<Integer> propertyIds = new ArrayList<>();
         if (propertiesJson != null && !propertiesJson.isEmpty()) {
            try {
               propertyIds = new ObjectMapper().readValue(propertiesJson,
new TypeReference<List<Integer>>() {});
```

```
            } catch (Exception ignored) {}
        }
        List<Square> squares = squareRepo.findAllById(propertyIds);
        int totalValue = money;
        for (Square s : squares) {
            if (s.getPrice() != null) {
                totalValue += s.getPrice();
            }
        }
        if (totalValue > maxTotalValue) {
            maxTotalValue = totalValue;
            winner = pid;
        }
    }
    return winner;
}
```

Ο αλγόριθμος για την ανάδειξη νικητή ξεκινά συγκεντρώνοντας όλους τους παίκτες της παρτίδας και, για καθέναν, να υπολογίζει το συνολικό του «κεφάλαιο». Πρώτα παίρνει τα διαθέσιμα χρήματα που έχει ο καθένας του και έχουν αποθηκευτεί στη μνήμη (Redis). Έπειτα αναζητεί τη λίστα με τα τετράγωνα-ιδιοκτησίες του κάθε παίκτη και ρωτά τη βάση (Square Repository) για την τιμή αγοράς τους. Η αξία κάθε ιδιοκτησίας προστίθεται στο ποσό των χρημάτων, ώστε να προκύψει μια ολοκληρωμένη εικόνα του πλούτου του κάθε παίκτη. Καθώς ο αλγόριθμος προχωρά, κρατά σημειωμένη τη μεγαλύτερη συνολική αξία που έχει συναντήσει και όταν ένας παίκτης ξεπεράσει το μέχρι τότε max, γίνεται προσωρινά πρώτος. Όταν ολοκληρωθεί ο έλεγχος όλων, ανακοινώνεται νικητής ο παίκτης με τη μεγαλύτερη συνολική περιουσία.

4.4 Επίμονα & Προσωρινά Δεδομένα

Η αρχιτεκτονική του Sindopoly διακρίνει ρητά τα επίμονα (Persistent) από τα προσωρινά (Volatile) δεδομένα. Τα πρώτα αποθηκεύονται στη MariaDB, ώστε να είναι διαθέσιμα και μετά τη λήξη ενός παιχνιδιού. Τα δεύτερα αποθηκεύονται στο Redis, ώστε να μπορεί να γίνεται πρόσβαση σε αυτά σε λιγότερο από ένα (1) χιλιοστό του δευτερολέπτου (millisecond) κατά τη διάρκεια της παρτίδας.

Η επίμονη όψη μοντελοποιείται με τη βιβλιοθήκη Spring Data JPA. Οι οντότητες Player, Square, Card και Stats αντιστοιχούν σε πίνακες 1-προς-1 με τα αντικείμενα domain. Κάθε repository κληρονομεί από JpaRepository. Έτσι οι βασικές CRUD πράξεις παράγονται από το Spring χωρίς επιπλέον κώδικα. Στον Κώδικας L παρουσιάζεται ως παράδειγμα η υλοποίηση του Player JPA αποθετηρίου σε κώδικα, το οποίο αντιστοιχείται ακριβώς με τον πίνακα Stats της MariaDB και των σειρών του.

Κώδικας L: Το Player JPA Αποθετήριο

```
@Entity
```

```
@Table(name = "Player")
public class Player {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "p_name")
    private String name;

    @Column(name = "created_at", insertable = false, updatable = false)
    private Timestamp createdAt;
}
```

Τα δεδομένα ενός συγκεκριμένου παιχνιδιού, τα οποία αλλάζουν συνεχώς, όπως τα χρήματα, η θέση του παίκτη, οι ιδιοκτησίες, κλπ, αποθηκεύονται στο Redis. Ο σχεδιασμός κλειδιών ακολουθεί πρόθεμα «οντότητα:αναγνωριστικό» (key-value pairs). Ένα hash game:42 κρατά μεταδεδομένα παρτίδας (status, timer, playerTurn), ενώ το set game:42:players αποθηκεύει τα playerIds. Κάθε ενεργός παίκτης εκπροσωπείται από hash player:{pid} με πεδία position, money, bankrupt. Οι ιδιοκτησίες χαρτογραφούνται ανά τετράγωνο σε square:{sid}. Στο πεδίο owner αποθηκεύεται το playerId ή NULL.

4.5 Front-end Υλοποίηση

Ο πελάτης (client) του Sindopoly υλοποιείται εξ' ολοκλήρου σε React 18 και αξιοποιεί τη βιβλιοθήκη Three.js για την απόδοση του τρισδιάστατου ταμπλό. Στην παρούσα υποενότητα παρουσιάζεται η συνολική αρχιτεκτονική του front-end: τα βασικά κλαδιά του component tree, τον μηχανισμό React Context που μοντελοποιεί τη σύνδεση WebSocket και διανέμει τα εισερχόμενα μηνύματα σε όλο το δέντρο, καθώς και τον renderer του Three.js που αναλαμβάνει τη δημιουργία και διαχείριση κινούμενων WebGL αντικειμένων. Επιπλέον, γίνεται αναφορά στα hooks διαχείρισης κατάστασης (useState και λοιπά hooks), τα οποία επιτρέπουν τη διαχείριση όλου του front-end χωρίς να απαιτούν επαναφόρτωση της σελίδας.

Η γραφική διεπαφή θα παρουσιαστεί με εικόνες σε επόμενο σχετικό κεφάλαιο. Η λεπτομερής ανάλυση του κώδικα του front-end θα ακολουθήσει στις επόμενες παραγράφους, καταδεικνύοντας πώς οι αρχές που τέθηκαν στο επίπεδο back-end μεταφέρονται, με συνέπεια, και στο περιβάλλον του προγράμματος περιήγησης.

4.6 Επικοινωνία Client-Server

Η αλληλεπίδραση μεταξύ του πελάτη (React) και του εξυπηρετητή (Spring Boot) υλοποιείται με έναν συνδυασμό REST και WebSocket-STOMP, ακολουθώντας το μοτίβο «CQRS». Ο Διαχωρισμός Ευθύνης Εντολής-Ερωτήματος (Command-query Responsibility Separation ή CQRS) είναι μια αρχή που δηλώνει ότι κάθε μέθοδος θα πρέπει να είναι είτε μια εντολή που εκτελεί μια ενέργεια είτε ένα ερώτημα που επιστρέφει δεδομένα στον καλούνται, αλλά όχι και τα δύο. Με άλλα λόγια, η υποβολή μιας ερώτησης δεν θα πρέπει να αλλάζει την απάντηση [57]. Στην προκειμένη περίπτωση, εντολές που τροποποιούν την κατάσταση αποστέλλονται μέσω HTTP POST αιτημάτων, ενώ τα συμβάντα που

προκύπτουν κοινοποιούνται σε όλους τους συμμετέχοντες μέσω καναλιών δημοσίευσης/εγγραφής.Οι εντολές διέρχονται επίσης από φίλτρα Spring Security και Bean Validation, προκειμένου να επιτευχθεί σταθερότητα.

Κάθε μήνυμα WebSocket φέρει επικεφαλίδες STOMP (destination, content-type, session) και σώμα JSON. Τα σχήματα ορίζονται ρητά με immutable DTOs. Το πιο συχνό είναι το TurnResponse, το οποίο περιλαμβάνει (gameId, playerId, newPosition, passedGo, deltaMoney, bankrupt, timestamp).

Το μονοπάτι WebSocket /ws προστατεύεται με πρωτόκολλο TLS (Transport Layer Security). Κατά το αρχικό handshake ο πελάτης αποστέλλει CSRF (Cross-Site Request Forgery) token που έχει λάβει από προηγούμενο REST αίτημα. Το CSRF είναι ένα ειδικό χαρακτηριστικό ασφαλείας που προστατεύει από επιβλαβή ή μη εξουσιοδοτημένα αιτήματα που υποβάλλονται σε εφαρμογές ιστού [58]. Εάν η σύνδεση διακοπεί (απώλεια δικτύου ή αλλαγή tab), αναλαμβάνει ο μηχανισμός SockJS reconnection ο οποίος είναι ενσωματωμένος στη βιβλιοθήκη @stomp/stompjs. Η βιβλιοθήκη επιχειρεί αρχικά σε σταθερό διάστημα τριών (3) δευτερολέπτων, ενώ έπειτα από δεύτερη αποτυχία ενεργοποιεί εκθετική καθυστέρηση (6 δευτερόλεπτα, 12 δευτερόλεπτα, κ.ο.κ.) μέχρι το μέγιστο των τριάντα (30) δευτερολέπτων, ώστε να αποφύγει τον καταιγισμό αιτημάτων στον εξυπηρετητή. Μόλις αποκατασταθεί η σύνδεση, ο πελάτης ζητά snapshot της κατάστασης μέσω ενός GET /api/game-state/{id} αιτήματος.

Οι εντολές περιορίζονται σε λίγες και σαφώς ορισμένες οδούς HTTP (HTTP paths):

- POST /game/create
- POST /game/{id}/start
- POST /turn/{id}/{player}
- POST /player/{gid}/{pid}/buy
- POST /player/{gid}/{pid}/pay-jail

Το σώμα τους περιέχει μόνο τα δεδομένα που απαιτούνται για την πράξη. Η ανταπόκριση 200 OK περιέχει απλώς operationStatus=ACCEPTED. Η πραγματική εξέλιξη (π.χ. μετακίνηση, νέα υπόλοιπα) έρχεται ως WebSocket γεγονός, εξαλείφοντας τα «race conditions» μεταξύ πολλών πελατών.

4.7 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε σε βάθος η πρακτική υλοποίηση του Sindopoly και απεικονίστηκε πως κάθε σχεδιαστική απόφαση του Κεφαλαίου 3 μεταφράστηκε σε συγκεκριμένο κώδικα, ροές και αυτοματισμούς. Οι δομές πακέτων του Spring Boot, το στρωματοποιημένο service model, τα σχήματα δεδομένων σε MariaDB και Redis, καθώς και τα κρίσιμα WebSocket μηνύματα, ευθυγραμμίζονται πλήρως με τα διαγράμματα Context, Container, Component και τα Flow Charts τα οποία χαρτογραφούν τη λογική του συστήματος. Οι αλγόριθμοι γύρου, ενοικίων και τερματισμού υλοποιούν χωρίς παρέκκλιση τους κανόνες που περιγράφηκαν στις λειτουργικές απαιτήσεις.

Στο επόμενο κεφάλαιο, Οδηγίες Χρήσης, θα μετατοπίσουμε την εστίαση από τον προγραμματιστή στον τελικό χρήστη και στον διαχειριστή. Θα περιγραφεί η διαδικασία δημιουργίας λογαριασμού, δημιουργίας και συμμετοχής σε παρτίδες, και ο γενικότερος τρόπος με τον οποίο παίζεται το Sindopoly μέσω της γραφικής διεπαφής του.

Κεφάλαιο 5ο: Γραφική Διεπαφή

5.1 Εισαγωγή

Το παρόν κεφάλαιο λειτουργεί ταυτόχρονα ως περιγραφή του οπτικού σχεδιασμού του Sindopoly και ως πρακτικός οδηγός χρήσης. Επικεντρώνεται στο πως ο παίκτης, μέσα από τον browser, αλληλεπιδρά με το τρισδιάστατο ταμπλό, τα αναδυόμενα παράθυρα αγορών και γενικότερα την όλη γραφική διεπαφή του συστήματος. Θα παρουσιαστούν όλα τα παράθυρα της διεπαφής, όπως:

- η οθόνη του login (για είσοδο στο σύστημα με ένα nickname)
- η αρχική οθόνη (κουμπιά για δημιουργία ή είσοδος σε lobby),
- η οθόνη του lobby (με μικρές αλλαγές αναλόγως αν είναι δημόσιο ή ιδιωτικό το παιχνίδι)
- η οθόνη του παιχνιδιού (ταμπλό, πιόνια, ζάρια, κλπ.)
- η οθόνη με τον πίνακα στατιστικών

Με οδηγό το κεφάλαιο αυτό κάθε μέλος της πανεπιστημιακής κοινότητας μπορεί να αξιοποιήσει πλήρως τις δυνατότητες του Sindopoly ως διαδικτυακή πλατφόρμα ψυχαγωγίας. Επίσης, οι κάρτες απόφασης και εντολής, αναμφισβήτητα ένα από τα πιο ξεχωριστά κομμάτια του παιχνιδιού, κάνουν συχνά αναφορά σε άτομα ή/και συμβάντα που λάβαν μέρος κατά κύριο λόγο μέσα στην Πανεπιστημιούπολη. Για αυτό το λόγο το Sindopoly μπορεί ταυτόχρονα να είναι ένας καλός τρόπος να κοινωνικοποιηθεί κανείς και να ενταχθεί στο γενικότερο κλίμα του Πανεπιστημίου.

5.2 Οθόνες

Στις σελίδες που ακολουθούν παρατίθενται χαρακτηριστικά στιγμιότυπα από κάθε βασική οθόνη της εφαρμογής. Κάθε εικόνα συνοδεύεται από επεξήγηση των λειτουργικών στοιχείων, και η σειρά των εικόνων αντιστοιχεί στη συνηθισμένη ροή της εφαρμογής.

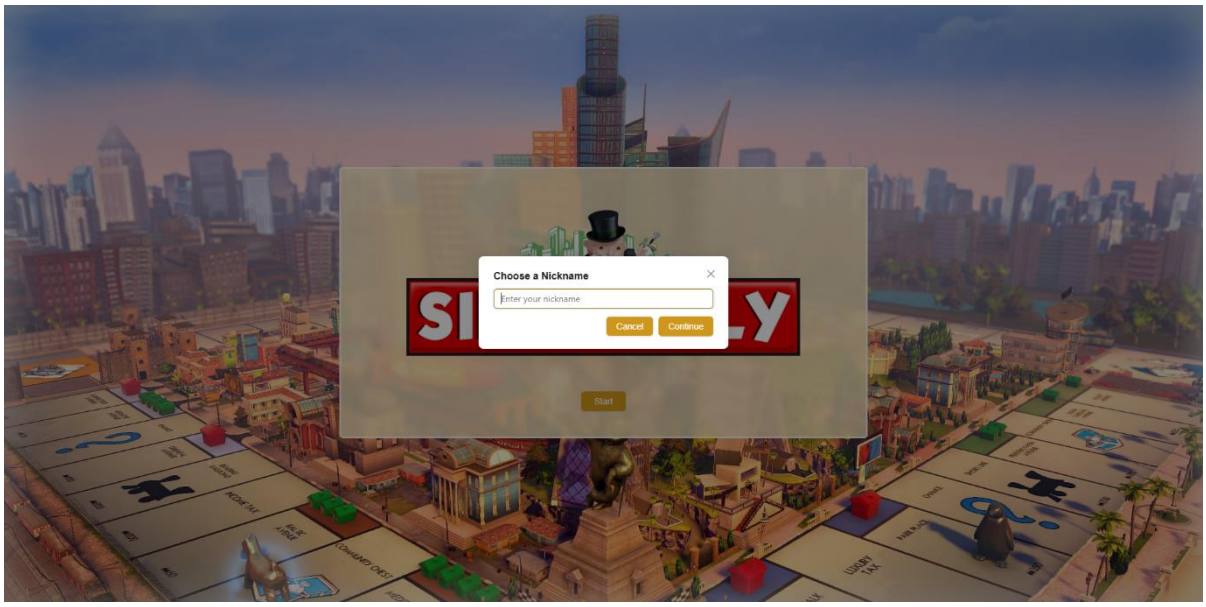
5.2.1 Οθόνη Εισόδου (Login)

Η πρώτη σελίδα που συναντά ο επισκέπτης είναι η Οθόνη Εισόδου (ή αλλιώς Login Page), και παρουσιάζεται στο Σχήμα 5.1. Στο κέντρο βρίσκεται το λογότυπο του «Sindopoly» πλαισιωμένο από τον «Mr. Monopoly», ώστε ο χρήστης να αναγνωρίζει άμεσα το θέμα του παιχνιδιού.

Σχήμα 5.1: Η Οθόνη Εισόδου (Login Page).

Με το πάτημα του κουμπιού Start (βλ. Σχήμα 5.1) εμφανίζεται ένα αναδυόμενο παράθυρο (modal) «Choose a Nickname» (βλ. Σχήμα 5.2). Ο χρήστης πληκτρολογεί ένα ψευδώνυμο (nickname) από 2 έως 15 χαρακτήρες, το οποίο περιέχει μόνο πεζά ή κεφαλαία γράμματα, αριθμούς ή κάτω παύλα (_), και πατά Continue. Και στο front-end και στο back-end η είσοδος του πεδίου γίνεται «sanitize», δηλαδή καθαρίζεται από απαγορευμένους χαρακτήρες και περνάει ως απλό κείμενο και όχι κώδικας, ώστε να αποφευχθούν κοινές επιθέσεις όπως η SQL Injection. Το front-end αποθηκεύει το ψευδώνυμο σε μεταβλητή session-storage και το περιλαμβάνει ως header σε όλες τις επόμενες κλήσεις WebSocket/REST. Δεν πραγματοποιείται εξωτερική ταυτοποίηση. Το nickname ισχύει μόνο για τη συγκεκριμένη συνεδρία (browser session) και εκκαθαρίζεται όταν ο browser κλείσει. Την επόμενη φορά που θα ανοίξει ξανά τον browser και θα επισκεφτεί την εφαρμογή, θα πρέπει να εισάγει ένα νέο nickname. Αυτός ο τύπος «ταυτοποίησης» είναι παρόμοιος με αυτόν που είχαν τα παλιά arcade games. Ο ελαφρύς αυτός μηχανισμός επιτρέπει στιγμιαία είσοδο και στο lobby και στο ταμπλό εμφανίζονται ονόματα αντιπάλων αντί για ανώνυμα IDs.

Αν ο χρήστης επιλέξει Cancel ή κλείσει το modal, επιστρέφει στην αρχική οθόνη χωρίς καμία μόνιμη εγγραφή. Με αυτόν τον μινιμαλιστικό σχεδιασμό μειώνονται τα εμπόδια εισόδου, ενώ συγχρόνως η εφαρμογή διαθέτει το ελάχιστο απαραίτητο αναγνωριστικό για να παρακολουθήσει γύρους, στατιστικά και ιστορικό παρτίδας κατά τη διάρκεια μίας μόνο συνεδρίας.



Σχήμα 5.2: Η Οθόνη Εισόδου με παράθυρο εισαγωγής nickname.

5.2.2 Κεντρική Οθόνη (Home)

Μετά την επιβεβαίωση ψευδώνυμου, ο χρήστης οδηγείται στην Κεντρική Οθόνη (βλ. Σχήμα 5.3). Στο δεξιό μέρος του πλαισίου εμφανίζεται το επιλεγμένο nickname συνοδευόμενο από προεπιλεγμένο avatar· το μικρό εικονίδιο «G» λειτουργεί ως χειριστήριο αποσύνδεσης· κάνοντας κλικ, ο χρήστης επιστρέφει στο modal εισαγωγής ονόματος ώστε να αλλάξει ψευδώνυμο χωρίς ανανέωση σελίδας.

Σχήμα 5.3: Η Κεντρική Οθόνη (Home Page).

Κάτω από το λογότυπο υπάρχουν τρία κύρια κουμπιά:

- **Create Game**: επιτρέπει στον χρήστη να δημιουργήσει ένα παιχνίδι.
- **Join Game**: επιτρέπει στον χρήστη να συνδεθεί σε ένα ήδη υπάρχον παιχνίδι.
- **Statistics**: επιτρέπει στον χρήστη να δει τα στατιστικά του καθώς και τα στατιστικά άλλων παικτών.

Η διάταξη προτάσσει αμέσως τις δύο βασικές ροές (δημιουργία ή συμμετοχή), ενώ κρατά τη σελίδα στατιστικών προσβάσιμη χωρίς να επισκιάζει την έναρξη παιχνιδιού.

Κάτω δεξιά βρίσκεται ένα ουδέτερο προεπιλεγμένο εικονίδιο «avatar», το nickname του χρήστη, καθώς και ένα κόκκινο κουμπί που πραγματοποιεί την αποσύνδεση του χρήστη. Στην προκειμένη περίπτωση, η κύρια χρήση αφορά την περίπτωση που ο χρήστης θέλει να αλλάξει το όνομά του.

Μία εναλλακτική της συγκεκριμένης οθόνης, είναι αυτή που παρουσιάζεται στο Σχήμα 5.4, κατά την οποία ο χρήστης βρίσκεται ήδη σε ένα παιχνίδι και έχει για κάποιο λόγο φύγει από την κατάλληλη οθόνη. Σε εκείνη την περίπτωση οφείλει να επιστρέψει στο παιχνίδι που βρισκόταν, μέχρι αυτό να τελειώσει. Για να γίνει αυτό, έχει στη διάθεσή του ένα και μοναδικό κουμπί, το Continue Game, το οποίο τον κατευθύνει ξανά στην οθόνη του παιχνιδιού.

Σχήμα 5.4: Η Κεντρική Οθόνη όταν ο χρήστης βρίσκεται ήδη σε παιχνίδι.

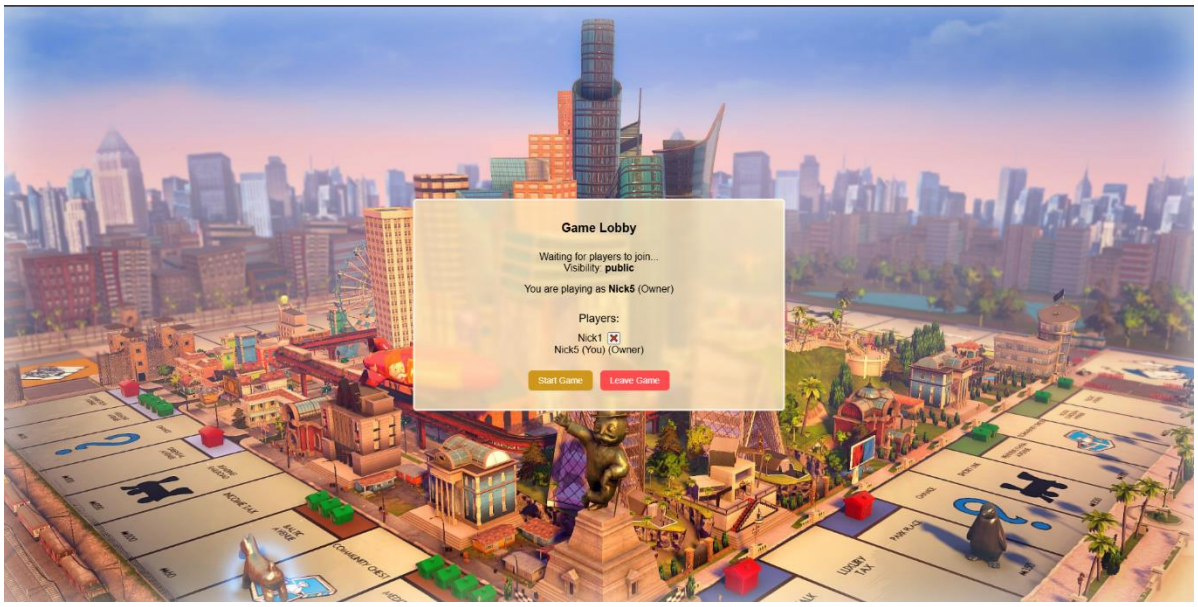
5.2.3 Δημιουργία Παιχνιδιού & Οθόνη Lobby

Στο Σχήμα 5.5 φαίνεται το modal «Create a New Game» που εμφανίζεται αφότου ο χρήστης επιλέξει Create Game στην κεντρική οθόνη. Ο διάλογος ζητά ένα μόνο πράγμα: αν το παιχνίδι θα είναι δημόσιο (Public), και επομένως ορατό σε όλους, ή ιδιωτικό (Private), και επομένως προσβάσιμο μόνο με κωδικό πρόσκλησης (invite code). Με το Create ο browser αποστέλλει αίτημα δημιουργίας.

Σχήμα 5.5: Η Κεντρική Οθόνη με παράθυρο για δημιουργία παιχνιδιού.

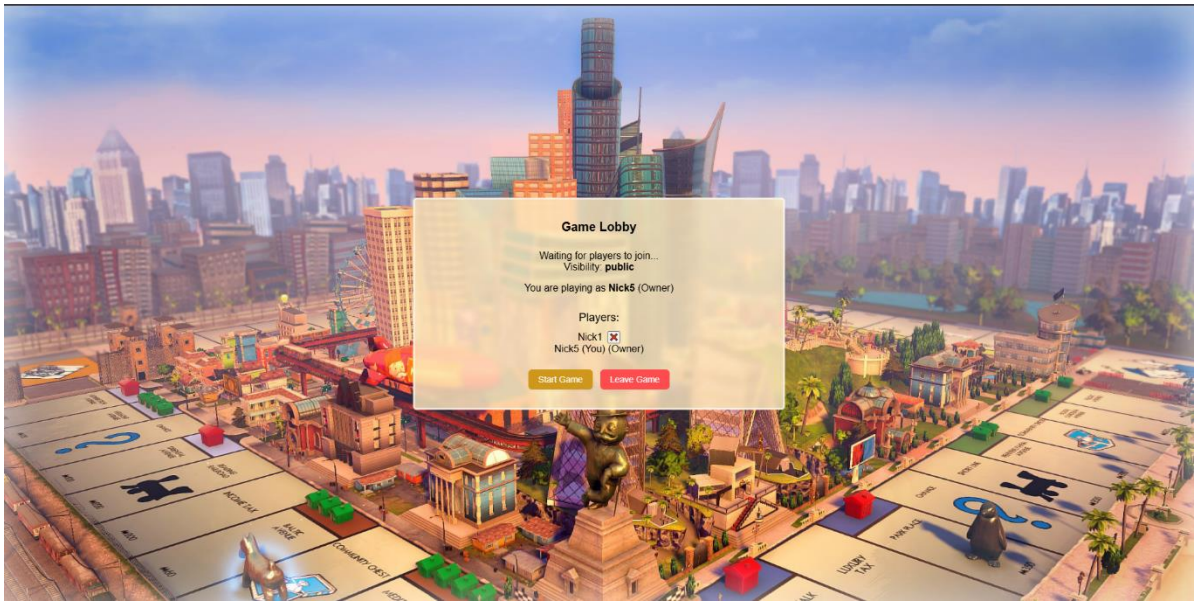
Με την επιτυχία της κλήσης ο παίκτης μεταφέρεται στην Οθόνη Lobby. Στο Σχήμα 5.6 παρουσιάζεται ένα παράδειγμα ενός δημόσιου παιχνιδιού:

- Υπάρχει μια επικεφαλίδα «Game Lobby» και από κάτω ενημέρωση για την κατάσταση «Waiting for players to join».
- Η γραμμή «Visibility: public» υποδηλώνει ότι το παιχνίδι είναι δημόσιο και ότι οποιοσδήποτε χρήστης μπορεί να το δει από την αντίστοιχη λίστα και να εισέλθει.
- Κάτω από τον τίτλο εμφανίζεται η φράση You are playing as <Nick> (Owner), ώστε ο δημιουργός να αντιλαμβάνεται τον ρόλο του.
- Στην ενότητα Players κάθε συμμετέχων εμφανίζεται με το ψευδώνυμό του. Και εκεί ενημερώνονται οι παίκτες για το ποιος είναι ο δημιουργός. Για όσους δεν είναι δημιουργοί/ιδιοκτήτες προβάλλεται εικονίδιο «X». Με το πάτημά του αποβάλλεται ο παίκτης από το παιχνίδι (kick).
- Τέλος, υπάρχουν δύο κουμπιά, το Start Game και το Leave Game. Το πρώτο ξεκινά το παιχνίδι, αφού πληρούνται οι κανόνες (2 < παίκτες μες το παιχνίδι < 4). Το δεύτερο επιτρέπει στον χρήστη να φύγει από το συγκεκριμένο παιχνίδι και να επιστρέψει στην Κεντρική Οθόνη. Αν ο χρήστης που βγει είναι και ο ιδιοκτήτης, τότε το παιχνίδι ακυρώνεται και όλοι οι παίκτες επιστρέφουν στην Κεντρική Οθόνη.



Σχήμα 5.6: Η Οθόνη Lobby ενός δημόσιο παιχνιδιού.

Το Σχήμα 5.7 παρουσιάζει ένα lobby ενός ιδιωτικού παιχνιδιού. Η μόνη διαφορά είναι η γραμμή «Visibility: private» και το πεδίο Invite Code, ένας αλφαριθμητικός κωδικός (π.χ. JJE3D) που πρέπει να κοινοποιηθεί στους συμπαίκτες του ιδιοκτήτη. Εκείνοι μπορούν να εισέλθουν στο παιχνίδι χρησιμοποιώντας αυτό τον κωδικό, καθώς το παιχνίδι δεν θα εμφανίζεται μαζί με τα δημόσια παιχνίδια στην αντίστοιχη λίστα. Ο ιδιοκτήτης εξακολουθεί να έχει το δικαίωμα εκκίνησης (Start Game) μόλις συγκεντρωθούν 2–4 παίκτες, και όλοι οι χρήστες το δικαίωμα της αποχώρησης (Leave Game).



Σχήμα 5.7: Η Οθόνη Lobby ενός ιδιωτικού παιχνιδιού.

5.2.4 Οθόνη Join Game

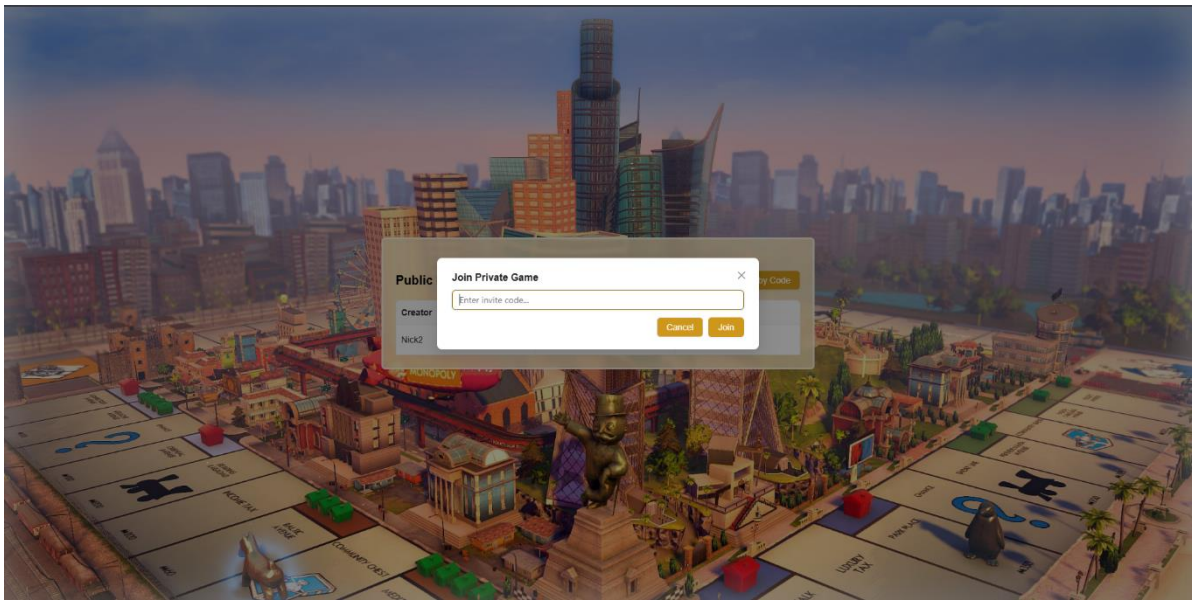
Στο Σχήμα 5.8 παρουσιάζεται η σελίδα Join Game, η οποία εμφανίζεται μόλις ο χρήστης πατήσει το κουμπί «Join Game» στην κεντρική οθόνη. Ο πίνακας ενημερώνεται σε πραγματικό χρόνο μέσω WebSocket και καταγράφει όλα τα διαθέσιμα δημόσια παιχνίδια που βρίσκονται σε κατάσταση

«lobby», δηλαδή παρτίδες που δεν έχουν ακόμη ξεκινήσει. Κάθε γραμμή αφορά ένα παιχνίδι και περιλαμβάνει τρεις (3) στήλες:

- **Creator**: το nickname του ιδιοκτήτη,
- **Players**: ο τρέχων αριθμός των παικτών που βρίσκονται στο παιχνίδι,
- **Action**: ένα κουμπί Join το οποίο επιτρέπει σε έναν χρήστη να εισέλθει στο lobby του αντίστοιχου παιχνιδιού.

Σχήμα 5.8: Η Οθόνη Join Game με όλα τα διαθέσιμα δημόσια παιχνίδια.

Για να εισέλθει ένας χρήστης σε ένα ιδιωτικό παιχνίδι, πρέπει να επιλέξει το κουμπί «Join by Code» επάνω δεξιά. Στο Σχήμα 5.9 παρουσιάζεται το modal που ανοίγει. Αυτό περιέχει ένα πεδίο εισαγωγής του πενταψήφιου invite code και δύο κουμπιά, Join και Cancel. Αν ο κωδικός είναι έγκυρος και το αντίστοιχο παιχνίδι βρίσκεται ακόμη σε κατάσταση «lobby», ο παίκτης εισέρχεται στο παιχνίδι και ανακατευθύνεται στην Οθόνη Lobby του παιχνιδιού. Διαφορετικά, επιστρέφεται κατάλληλο μήνυμα σφάλματος («Invalid code» ή «Game already started»).



Σχήμα 5.9: Η Οθόνη Join Game με παράθυρο για σύνδεση σε ιδιωτικό παιχνίδι.

5.2.5 Οθόνη Statistics

Η επιλογή Statistics από την Κεντρική Οθόνη οδηγεί στην οθόνη που παρουσιάζεται στο Σχήμα 5.10 (Οθόνη Statistics). Στο επάνω τμήμα εμφανίζονται τα προσωπικά στατιστικά στοιχεία του συνδεδεμένου χρήστη (Your Stats — Nick5). Αυτά περιλαμβάνουν τις συνολικές νίκες του, το πλήθος των παιχνιδιών που έχει παίξει και το ποσοστό επιτυχίας του (Win Rate). Τα δεδομένα αντλούνται απευθείας από τον πίνακα stats της MariaDB και ανανεώνονται αμέσως μετά το τέλος κάθε παιχνιδιού.

Σχήμα 5.10: Η Οθόνη Statistics.

Κάτω από τα ατομικά στοιχεία προβάλλεται ο πίνακας Leaderboard. Αυτός αναδεικνύει τους δέκα (10) κορυφαίους παίκτες με βάση το κριτήριο ταξινόμησης που έχει επιλεγεί. Ο χρήστης μπορεί να ταξινομήσει τη λίστα κάνοντας κλικ στις επικεφαλίδες Wins, Games ή Win Rate. Το εικονίδιο βέλους δηλώνει τη φορά ταξινόμησης και εμφανίζεται και αντίστοιχο tooltip για πιο ξεκάθαρη επεξήγηση. Με τον μηχανισμό αυτό οι παίκτες παρακολουθούν την προσωπική τους πρόοδο αλλά και τη συνολική

κατάταξη της κοινότητας, γεγονός που ενισχύει το αίσθημα ανταγωνισμού ανάμεσα στους χρήστες της εφαρμογής.

5.2.6 Κύρια Οθόνη Παιχνιδιού

Η Κύρια Οθόνη (βλ. Σχήμα 5.11) εμφανίζεται μόλις όλοι ο ιδιοκτήτης του παιχνιδιού πατήσει Start Game στο lobby και αποτελεί το κέντρο της αλληλεπίδρασης κατά τη διάρκεια του παιχνιδιού. Στο προσκήνιο υπάρχει το τρισδιάστατο ταμπλό, το οποίο είναι στην ουσία ένα μοντέλο React + Three.js τοποθετημένο έτσι ώστε να θυμίζει ένα φυσικό ταμπλό. Πάνω του κινούνται τα πιόνια των παικτών. Στην τρέχουσα έκδοση χρησιμοποιούνται βασικά γεωμετρικά σχήματα (κύβος, σφαίρα, τόρος και κώνος, εδώ εμφανίζονται μόνο τα πρώτα δύο), τα οποία όμως φέρουν μοναδικό χρώμα για γρήγορη οπτική ταύτιση. Σε επόμενη έκδοση θα αντικατασταθούν από «αληθινά» πιόνια (π.χ. καπέλο, σκύλος, κλπ.) για μεγαλύτερη παραστατικότητα.

Σχήμα 5.11: Η Κύρια Οθόνη Παιχνιδιού.

Στο επάνω μέρος της οθόνης προβάλλονται τρεις διαδοχικές ζώνες ελέγχου (nav bars). Η πρώτη, Score Nav, παρουσιάζει το χρηματικό υπόλοιπο κάθε παίκτη σε μορφή «chip» με αντίστοιχη χρωματική σήμανση. Αν ένας παίκτης δεν συμμετέχει, η τιμή εμφανίζεται ως «–». Με τον τρόπο αυτό οι συμμετέχοντες αντιλαμβάνονται ακαριαία τη σχετική οικονομική κατάσταση όλων χωρίς να ανοίγουν πρόσθετα παράθυρα. Ακριβώς από κάτω βρίσκεται η Action Nav, η οποία από αριστερά προς τα δεξιά διαθέτει:

- κουμπί εναλλαγής ήχου για την αναπαραγωγή/σίγαση του παρεχόμενου τραγουδιού που παίζει στο παρασκήνιο,
- κουμπί εμφάνισης ιδιοκτησιών που ανοίγει νέο παράθυρο με τις ιδιοκτησίες του παίκτη (βλ. Σχήμα 5.12) και
- κουμπί ρίψης ζαριών που ρίχνει τα ζάρια για τον παίκτη. Το κουμπί απενεργοποιείται αυτόματα όταν δεν είναι η σειρά του χρήστη.

Σχήμα 5.12: Αναδυόμενο Παράθυρο Ιδιοκτησιών.

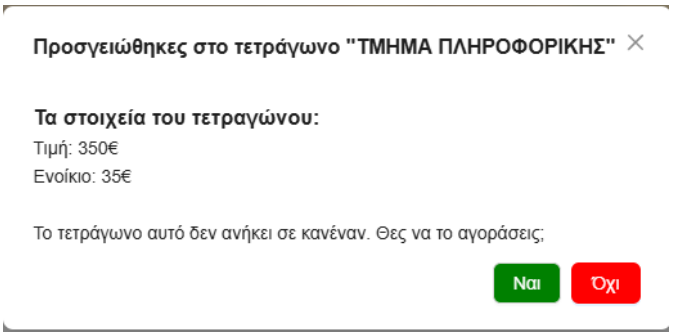
Η τρίτη λωρίδα, Dice Nav, φιλοξενεί ένα chip το οποίο φωτίζεται μετά τη ρίψη και εμφανίζει το άθροισμα των δύο ζαριών (τιμή 2-12). Αυτή η σχεδίαση αφαιρεί τον οπτικό θόρυβο από το ταμπλό, επιτρέποντας στον παίκτη να εστιάσει στην κεντρική σκηνή. Ωστόσο, σε επόμενη έκδοση θα αντικατασταθεί αυτή η λειτουργία από «αληθινά» ζάρια με κατάλληλο animation για μεγαλύτερη παραστατικότητα.

Η οθόνη ενημερώνεται σε πραγματικό χρόνο μέσω WebSocket. Μόλις ένας παίκτης ρίξει τα ζάρια, η τιμή στο Dice Nav και η θέση του αντίστοιχου πιονιού συγχρονίζονται ταυτόχρονα σε όλα τα προγράμματα περιήγησης. Οι λεπτομέρειες της ροής ενός γύρου θα αναλυθούν διεξοδικά στην επόμενη υποενότητα.

5.2.7 Παράθυρα Ενεργειών

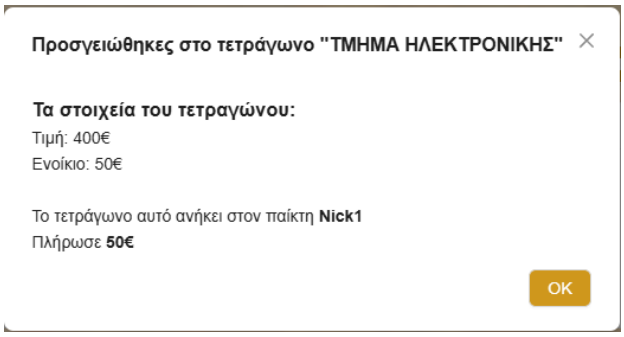
Κατά τη ροή του παιχνιδιού το Sindopoly χρησιμοποιεί μικρά αυτόματα αναδυόμενα παράθυρα (modals) για να ενημερώσει τον παίκτη και να συλλέξει την άμεση απόκριση όπου απαιτείται, αν απαιτείται. Σε αντίθεση με το προαιρετικό παράθυρο εμφάνισης ιδιοκτησιών της προηγούμενης ενότητας, που ανοίγει μόνο όταν ο χρήστης το επιλέξει, τα συγκεκριμένα modals ενεργοποιούνται από το ίδιο το σύστημα αμέσως μετά την άφιξη του πιονιού σε ένα καινούργιο τετράγωνο. Έτσι διακόπτεται προσωρινά η κίνηση, εξασφαλίζοντας ότι ο παίκτης αντιλαμβάνεται πλήρως το γεγονός πριν συνεχιστεί ο γύρος.

Όταν ο παίκτης προσγειωθεί σε ιδιοκτησία που δεν ανήκει σε κανέναν, εμφανίζεται παράθυρο (βλ. Σχήμα 5.13) με το όνομα του τετραγώνου, την τιμή αγοράς της και το βασικό ενοίκιο. Ο παίκτης ερωτάται αν θέλει να την αγοράσει. Δύο κουμπιά, «Ναι» (πράσινο) και «Όχι» (κόκκινο), επιτρέπουν στον παίκτη να δηλώσει την πρόθεσή του.



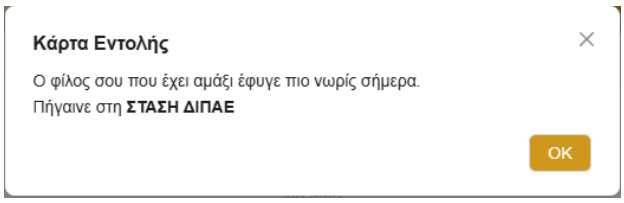
Σχήμα 5.13: Αναδυόμενο Παράθυρο για Αγορά Αδέσμευτης Ιδιοκτησίας.

Αν το τετράγωνο ανήκει ήδη σε άλλον παίκτη, το παράθυρο αναφέρει, πέρα από τα υπόλοιπα στοιχεία όπως στο Σχήμα 5.13, και τον ιδιοκτήτη και το οφειλόμενο ενοίκιο (βλ. Σχήμα 5.14). Καμία επιλογή δεν παρέχεται πέρα από το «ΟΚ», αφού η πληρωμή είναι υποχρεωτική.



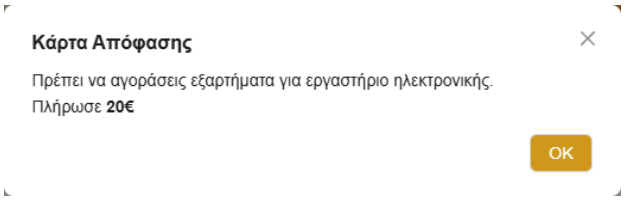
Σχήμα 5.14: Αναδυόμενο Παράθυρο για Πληρωμή Ενοικίου σε Ξένη Ιδιοκτησία.

Για τετράγωνα τύπου «Chance» (Κάρτα Εντολής) εμφανίζεται το κείμενο της κάρτας (βλ. Σχήμα 5.15), π.χ. «Πήγαινε στη ΣΤΑΣΗ ΔΙΠΑΕ». Με το πάτημα «OK» εφαρμόζεται αυτόματα η εντολή.



Σχήμα 5.15: Αναδυόμενο Παράθυρο για Κάρτα Εντολής.

Για τετράγωνα τύπου «Community Chess» (Κάρτα Απόφασης), ένα αντίστοιχο παράθυρο παρουσιάζει την περιγραφή και το χρηματικό πρόστιμο (ή αμοιβή) της κάρτας (βλ. Σχήμα 5.16), ζητώντας ξανά απλώς επιβεβαίωση.



Σχήμα 5.16: Αναδυόμενο Παράθυρο για Κάρτα Απόφασης.

Κάθε φορά που ο παίκτης διασχίζει ή προσγειώνεται στο τετράγωνο «ΑΦΕΤΗΡΙΑ», ένα λιτό παράθυρο τον ενημερώνει για την πίστωση των 200€ και κλείνει με το κουμπί «OK», όπως φαίνεται και στο Σχήμα 5.17.

Σχήμα 5.17: Αναδυόμενο Παράθυρο για Πέρασμα από Αφετηρία.

5.3 Επίλογος

Το κεφάλαιο αυτό ανέδειξε τη γραφική διεπαφή του Sindopoly ως αναπόσπαστο κομμάτι της εμπειρίας του παιχνιδιού. Παρουσιάστηκε πώς τα modals οδηγούν διαισθητικά τον χρήστη στις βασικές ενέργειες (Create, Join, Invite) και πώς το lobby ενημερώνει σε πραγματικό χρόνο για την κατάσταση των παικτών. Ο πίνακας στατιστικών και το sortable leaderboard ολοκληρώνουν τον κύκλο δίνοντας κίνητρο διαρκούς βελτίωσης και υγιούς ανταγωνισμού.

Η σχεδιαστική έμφαση στην καθαρή τυπογραφία, στα θερμά χρώματα και στη συνεχή οπτική ανατροφοδότηση μετατρέπει τους περίπλοκους κανόνες του Monopoly σε μια φιλική εμπειρία για τον χρήστη. Το επόμενο κεφάλαιο στρέφει τον φακό στην αποτίμηση του έργου, δηλαδή τις μεθοδολογίες που χρησιμοποιήθηκαν και μερικές από τις μελλοντικές βελτιώσεις που θα μπορούσαν να υλοποιηθούν ώστε να εξελίξουν ακόμη περισσότερο το Sindopoly.

Κεφάλαιο 6ο: Αποτίμηση Έργου

6.1 Μεθοδολογία Ανάπτυξης Agile

Αν και οι κανόνες και οι απαιτήσεις του «Monopoly» είναι γνωστές και συγκεκριμένες, η ανάπτυξη του Sindopoly ξεκίνησε με σχετικά ελλιπώς ορισμένες απαιτήσεις. Το μόνο σαφές ήταν ότι χρειαζόταν ένα διαδικτυακό Monopoly-like παιχνίδι με αλληλεπίδραση σε πραγματικό χρόνο και σταθερές επιδόσεις με πολλούς ταυτόχρονους χρήστες. Ο περιορισμένος χρόνος που υπήρχε για την υλοποίηση του συστήματος σήμαινε πως κατά πάσα πιθανότητα δεν θα υλοποιόντουσαν όλοι οι κανόνες του κλασικού «Monopoly», αλλά μόνο οι πιο «βασικοί». Και πράγματι, αν και δεν υπάρχει η δυνατότητα οι παίκτες να χτίσουν σπίτια για παράδειγμα, όλοι οι υπόλοιποι κανόνες και απαιτήσεις που απαιτούνται για να λειτουργήσει σωστά το σύστημα και αυτοί που γενικότερα κάνουν το Monopoly «Monopoly», υπάρχουν στο τελικό προϊόν. Για να αντιμετωπισθεί αυτή η αβεβαιότητα επιλέχθηκε η μεθοδολογία Agile Scrum.

Το Agile προέκυψε το 2001 ως ένα «Μανιφέστο» αξιών και αρχών που θέτει στο επίκεντρο τη συνεχή παράδοση λειτουργικού λογισμικού, τη συνεργασία με τον πελάτη και την ταχύτατη προσαρμογή σε μεταβαλλόμενες απαιτήσεις [59]. Οι δημιουργοί του, ανάμεσά τους προσωπικότητες όπως ο Kent Beck και ο Robert C. Martin (Uncle Bob), διαπίστωσαν ότι τα παραδοσιακά μοντέλα καταρράκτη προκαλούσαν τη συγγραφή ογκώδους τεκμηρίωσης (documentation) και αργούς κύκλους έγκρισης, αφήνοντας ελάχιστο περιθώριο για τη συγγραφή ποιοτικού κώδικα καθώς και την εξέλιξη του όσο το έργο προχωρούσε. Το Agile αντιπαραθέτει μικρούς, επαναληπτικούς κύκλους ανάπτυξης, έμφαση στη δέσμευση της ομάδας και διαρκή ανατροφοδότηση. Με αυτό τον τρόπο επιδιώκει υψηλής ποιότητας ευέλικτο λογισμικό που ακολουθεί τις πραγματικές ανάγκες του χρήστη. Παρότι οι αξίες του Agile δεν ταιριάζουν σε όλα τα είδη έργων, π.χ. σε εξαιρετικά ρυθμιζόμενα συστήματα ασφαλείας μπορεί να προτιμηθεί μοντέλο καταρράκτη [60], η φιλοσοφία του έχει επηρεάσει αποφασιστικά τη σύγχρονη μηχανική λογισμικού, καθιερώνοντας τη συνεργατική και επαναληπτική ανάπτυξη ως «καλύτερη πρακτική» για τα περισσότερα εμπορικά και ακαδημαϊκά έργα.

Πριν ξεκινήσει η υλοποίηση του Sindopoly, πραγματοποιήθηκε λεπτομερής διερεύνηση του πυρήνα του παιχνιδιού: ορισμός τετραγώνων, τύποι καρτών, κανόνες ιδιοκτησίας και μεταβλητές κατάστασης που πρέπει να επιμένουν σε κάθε παρτίδα. Αφού φτιάχτηκε ένα ικανοποιητικό αρχικό σχήμα οντοτήτων (MariaDB για μόνιμα στοιχεία, Redis για runtime state), ξεκίνησε η διερεύνηση των τεχνολογιών που θα χρησιμοποιηθούν. Έπειτα ξεκίνησε η ανάπτυξη του back-end σε Spring Boot και του front-end σε React + Three.js. Κάθε δεύτερη εβδομάδα πραγματοποιούνταν διαδικτυακή συνάντηση με τον κ. Σιδηρόπουλο, υπεύθυνο καθηγητή της συγκεκριμένης Δ.Ε., μέσω Zoom, κατά την οποία παρουσιαζόντουσαν στον κ. Σιδηρόπουλο οι λειτουργίες που υλοποιήθηκαν (π.χ. μηχανισμό lobby, WebSocket push, αλγόριθμο ενοικίων). Στο κλείσιμο κάθε συνάντησης γινόταν αναφορά και συζήτηση σχετικά με τα επόμενα user cases, λαμβάνοντας υπόψη τον διαθέσιμο χρόνο και τη σημασία τους για την εμπειρία του παίκτη.

Ο κύκλος αυτός, ανάπτυξη, παρουσίαση, και ανατροφοδότηση, εξασφάλισε τη συνεχή και ελεγχόμενη εξέλιξη του Sindopoly χωρίς απρόσμενες εκπλήξεις στην πορεία.

6.2 Μελλοντικές Βελτιώσεις

Η παρούσα έκδοση του Sindopoly καλύπτει τις θεμελιώδεις λειτουργίες ενός real-time διαδικτυακού παιχνιδιού. Ωστόσο, υπάρχουν αρκετά σημεία όπου επεκτάσεις θα μπορούσαν να εμπλουτίσουν

ουσιαστικά την εμπειρία των παικτών ή να βελτιώσουν την επιχειρησιακή απόδοση του συστήματος. Οι προτάσεις που ακολουθούν μπορούν να αποτελέσουν πεδίο ενασχόλησης για μελλοντικές Δ.Ε. ή φοιτητικά projects, παρέχοντας γόνιμο έδαφος τόσο σε back-end όσο και σε front-end κατευθύνσεις, αλλά και στην υποδομή.

6.2.1 Ranked Matchmaking & ELO Leaderboard

Μία από τις πιο συχνές προτάσεις βελτίωσης σε διαδικτυακά παιχνίδια, στα οποία χρήστες παίζουν ενάντια άλλων χρηστών, είναι η ύπαρξη ενός συστήματος βαθμολογίας ώστε κάθε παρτίδα να έχει ανταγωνιστικό κίνητρο και μετρήσιμη πρόοδο. Η καθιέρωση «Ranked Matchmaking» σε συνδυασμό με ένα «ELO Leaderboard» προσφέρει ακριβώς αυτό. Στην ουσία επιτρέπει στον παίκτη να αντιστοιχίζεται με αντιπάλους παρόμοιου επιπέδου και, ταυτόχρονα, παρέχει κάποιον μακροπρόθεσμο στόχο και ικανοποίηση στον παίκτη μέσα από την άνοδο θέσεων στον πίνακα κατάταξης.

Αρχικά, ο αλγόριθμος ELO. Για κάθε παίκτη προστίθεται πεδίο eloRating στον πίνακα stats της MariaDB, αρχικοποιημένο σε 1000 mmr (Matchmaking Rating). Μετά τον τερματισμό παρτίδας ο GameStateService υπολογίζει νέο rating βάσει του κλασικού τύπου Elo, ο οποίος φαίνεται στη σχέση (6.1), η οποία προτάθηκε από τον Άρπαντ Έλο τον Αύγουστο του 1967 [61] .

$$R'_A = R_A + K \times (S_A - E_A) \tag{6.1}$$

Ουσιαστικά για να χρησιμοποιηθεί η σχέση (6.1), γίνεται η υπόθεση ότι ο παίκτης A (με βαθμολογία R_A) αναμενόταν να σκοράρει πόντους E_A αλλά στην πραγματικότητα σκόραρε πόντους S_A. Ο παράγοντας K είναι η μέγιστη δυνατή προσαρμογή ανά παιχνίδι.

Έπειτα, σχετικά με το Matchmaking. Στο lobby προστίθεται επιλογή «Ranked Game». Όταν επιλεχθεί, ο server τοποθετεί τον παίκτη σε ουρά rankQueue ταξινομημένη κατά eloRating. Κάθε πέντε (5) δευτερόλεπτα ένα scheduled task επιχειρεί να ομαδοποιήσει τέσσερα συνεχόμενα στοιχεία με διαφορά ≤ 150 mmr. Εάν δεν υπάρξει ταίριασμα εντός ενενήντα (90) δευτερολέπτων, το επιτρεπτό εύρος ανοίγει σταδιακά (από 150 σε 300 και έπειτα σε «open»).

Σχετικά με το frontend. Η γραφική διεπαφή αποκτά νέα καρτέλα εν ονόματι Leaderboard, η οποία τραβά τους πενήντα (50) κορυφαίους παίκτες με τα μεγαλύτερα eloRating..

Η προσθήκη Ranked Mode δεν επηρεάζει κανέναν υπάρχοντα κανόνα παιχνιδιού· απαιτεί μόνο επέκταση του API και ελαφριά αλλαγή στον κύκλο τερματισμού της παρτίδας. Το όφελος, όμως, είναι πολλαπλό: κίνητρο για να συμμετέχουν οι χρήστες επανειλημμένα σε παιχνίδια, αντικειμενικό μέτρο εξέλιξης και ενίσχυση της κοινότητας μέσω υγιούς ανταγωνισμού.

6.2.2 Είσοδος με OAuth 2, Λίστα Φίλων & Προσκλήσεις

Η τρέχουσα έκδοση του Sindopoly χρησιμοποιεί απλό μηχανισμό τοπικής εγγραφής. Για να εξασφαλίσουμε ενιαία ταυτοποίηση και ευκολότερη διαχείριση λογαριασμών, προτείνεται η υιοθέτηση του πρωτοκόλλου OAuth 2.0.

Το OAuth 2.0 είναι ένα πρωτόκολλο που επιτρέπει στους χρήστες να παραχωρούν σε εφαρμογές τρίτων πρόσβαση στα δεδομένα τους σε έναν διακομιστή ιστού χωρίς να χρειάζεται να κοινοποιούν τα διαπιστευτήριά τους [62]. Είναι ένα ευρέως χρησιμοποιούμενο πρότυπο για έλεγχο ταυτότητας και εξουσιοδότηση.

Μετά την ενσωμάτωση της ταυτοποίησης με OAuth 2.0, ανοίγει ο δρόμος για τις λίστες φίλων (Friends List). Προστίθεται πίνακας friend_link (owner_id, friend_id, status) όπου το status μπορεί να πάρει τιμές {PENDING, ACCEPTED, BLOCKED}. Οι προσκλήσεις αξιοποιούν τόσο το Friends List όσο και το OAuth id. Όταν ο ιδιοκτήτης lobby πατά «Invite», ο server δημιουργεί εγγραφή game_invite στην Redis με TTL (time to live) τριάντα (30) δευτερόλεπτα και προωθεί μήνυμα στον συγκεκριμένο φίλο. Ο παραλήπτης βλέπει μια ειδοποίηση, για παράδειγμα «Ο Γιώργος σε καλεί στο Game #57», και με ένα κλικ εισέρχεται στο lobby με ειδικό κωδικό που μπαίνει αυτόματα.

Η τριπλέτα OAuth 2 + Friends + Invites μπορεί να ενισχύσει την κοινωνική διάσταση του Sindopoly, καθώς μειώνει τα εμπόδια εισόδου, διευκολύνει τον σχηματισμό κλειστών παιχνιδιών και δημιουργεί βάση για μελλοντικά χαρακτηριστικά, όπως ιδιωτικές συνομιλίες ή ομαδικές διοργανώσεις.

6.2.3 Λειτουργία Θεατή και Εξαγωγή Επανάληψης

Μια από τις πιο ελκυστικές επεκτάσεις για το Sindopoly είναι η υποστήριξη λειτουργίας θεατή (Spectator Mode), όπως υπάρχει σε πολλά άλλα multiplayer παιχνίδια στην αγορά. Η ιδέα είναι απλή: κάθε ενεργή παρτίδα μπορεί να δεχθεί επιπλέον συνδέσεις «read-only» που παρακολουθούν τη ροή των γύρων χωρίς να αλληλεπιδρούν. Σε επίπεδο back-end, ο μηχανισμός πιθανώς να απαιτεί ένα νέο topic /topic/spectate/{gameId}.

Σε δεύτερο επίπεδο, προτείνεται η δυνατότητα Εξαγωγής Επανάληψης (Live Replay Export). Η TurnService θα καταγράφει κάθε γύρο σε JSON log (μέγ. 200 KB ανά παρτίδα). Με την ολοκλήρωση του παιχνιδιού, θα μετατρέπεται το αρχείο σε compressed event stream και θα συσχετίζεται με τους παίκτες του παιχνιδιού και την ημερομηνία διεξαγωγής του. Το front-end θα έχει μια νέα σελίδα εν ονόματι «Replays», όπου ο χρήστης θα επιλέγει ένα παιχνίδι και θα το αναπαράγει τοπικά. Ουσιαστικά τα γεγονότα του παιχνιδιού θα επανεκτελούνται με χρονικές σφραγίδες, επιτρέποντας παύση, fast-forward, κλπ.. Για κοινή χρήση σε κοινωνικά δίκτυα προβλέπεται η εξαγωγή σε μορφή MP4 με τη βοήθεια κατάλληλων βιβλιοθηκών.

Η υλοποίηση Spectator Mode αυξάνει τη βιωσιμότητα του παιχνιδιού, καθώς οι θεατές θα μπορούν να μετατραπούν εύκολα σε ενεργούς παίκτες σε επόμενα παιχνίδια, ενώ το σύστημα Live Replay Export θα προσφέρει υλικό για να αναλυθούν οι στρατηγικές των παικτών, να προβληθούν τα «highlights» του παιχνιδιού και πιθανώς να μπορούν να διοργανωθούν ακόμα και τουρνουά με κριτική επιτροπή.

6.3 Επίλογος

Η μέχρι τώρα πορεία του Sindopoly αποδεικνύει ότι η επιλογή της μεθοδολογίας Agile Scrum υπήρξε καθοριστική για την επιτυχία του έργου. Παρά τα ελλιπώς ορισμένα αρχικά ζητούμενα, έγινε εφικτό να παραδοθεί ένα λειτουργικό, διαδικτυακό Monopoly-like παιχνίδι. Η συνεργασία με τον επιβλέποντα κ. Σιδηρόπουλο εγγυήθηκε την έγκαιρη διόρθωση λαθών και την εξασφάλιση παράδοσης ενός ικανοποιητικού τελικού (προς το παρόν) συστήματος.

Ταυτόχρονα, η χαρτογράφηση των πιθανών μελλοντικών επεκτάσεων τεκμηριώνει μια σαφή και ρεαλιστική στρατηγική για την εξέλιξη του προϊόντος χωρίς να διακυβεύει την υφιστάμενη αρχιτεκτονική. Συνεπώς το Sindopoly έχει πλέον θεμελιωθεί ως ένα πλήρως λειτουργικό MVP (Minimum Viable Product) που ικανοποιεί τις βασικές απαιτήσεις που υπήρχαν.

Όπως και να έχει, πάντα θα υπάρχει περιθώριο βελτίωσης. Από το back-end και τα endpoints του, μέχρι και το front-end και τη γενικότερη γραφική διεπαφή, όλα τα κομμάτια κώδικα θα μπορούσαν να βελτιωθούν και να ανασυγκροτηθούν για μια καλύτερη διαρρύθμιση και δομή. Ακόμα και η

τεκμηρίωση (documentation) όλου του έργου θα μπορούσε να γραφτεί καλύτερα, και εννοείται να συντηρείται ταυτόχρονα με όλες τις νέες αλλαγές και βελτιώσεις που γίνονται.

Κοιτάζοντας πίσω, το Sindopoly ξεκίνησε ως μια ιδέα δύο φοιτητών. Σήμερα αποτελεί μια ζωντανή ηλεκτρονική πλατφόρμα όπου συμφοιτητές και φίλοι μπορούν να «συναντιούνται», να ανταγωνίζονται και να διασκεδάζουν παρέα ο ένας με τον άλλον. Η πρώτη έκδοση μπορεί να μην περιέχει ακόμη κάθε λεπτομέρεια και κανόνα του κλασικού Monopoly, ωστόσο διαθέτει ένα σημαντικό χαρακτηριστικό: τη δυνατότητα να μετατρέψει ένα απλό βράδυ σε μια αξέχαστη εμπειρία. Το έργο μένει ανοιχτό όχι μόνο ως repository στο GitHub, αλλά ως πρόσκληση σε όποιον θέλει να βελτιώσει τον κώδικα του, να σχεδιάσει νέες κάρτες, να οργανώσει τουρνουά ή να προσθέσει περαιτέρω λειτουργίες προσβασιμότητας. Γιατί η μαγεία του Sindopoly δεν βρίσκεται μόνο στο πολύχρωμο ταμπλό, αλλά και στο ταξίδι της δημιουργίας που μόλις ξεκίνησε και αναμένεται να συνεχιστεί με τις επόμενες καινοτόμες πινελιές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] "ngUnick/Sindopoly: Sindopoly is an online game inspired by Monopoly with the theme of International Hellenic University (IHU). The project aims to create an interactive gaming experience that mirrors the academic and social environment of IHU while following Monopoly rules." Accessed: Jan. 15, 2025. [Online]. Available: https://github.com/ngUnick/Sindopoly

[2] "Buy MONOPOLY® on PC & More | Ubisoft Store." Accessed: Jan. 15, 2025. [Online]. Available: https://store.ubisoft.com/ie/monopoly-/6638e798aa201a2d55bcc775.html?lang=en-ZW

[3] "Buy MONOPOLY® on PC & More | Ubisoft Store." Accessed: Jan. 15, 2025. [Online]. Available: https://store.ubisoft.com/ie/monopoly-/6638e792aa201a2d55bcc774.html?lang=en-ZW

[4] "Monopoly - the classic board game on mobile by Marmalade Game Studio." Accessed: Jan. 15, 2025. [Online]. Available: https://www.marmaladegamestudio.com/games/monopoly/

[5] "geeeeeeeeek/monopoly: Monopoly Game on the Web: WebGL, ThreeJS and WebSocket. CMU course project." Accessed: Jan. 15, 2025. [Online]. Available: https://github.com/geeeeeeeeek/monopoly

[6] "intrepidcoder/monopoly: A fully functional Monopoly game in JavaScript and HTML/CSS. Play online: https://www.intrepidcoder.com/projects/monopoly/." Accessed: Jan. 15, 2025. [Online]. Available: https://github.com/intrepidcoder/monopoly

[7] "React." Accessed: Jan. 15, 2025. [Online]. Available: https://react.dev/

[8] M. Persson, "JavaScript DOM Manipulation Performance : Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks," 2020, Accessed: Jan. 15, 2025. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:bth-19531

[9] "Fundamentals - three.js manual." Accessed: Jan. 15, 2025. [Online]. Available: https://threejs.org/manual/#en/fundamentals

[10] "First public version. Still a lot to do · mrdoob/three.js@a90c4e1." Accessed: Jan. 15, 2025. [Online]. Available: https://github.com/mrdoob/three.js/commit/a90c4e107ff6e3b148458c96965e876f9441b147

[11] "Unity - Manual: Unity 6 User Manual." Accessed: Jan. 16, 2025. [Online]. Available: https://docs.unity3d.com/Manual/index.html

[12] "Unity - Manual: Web introduction." Accessed: Jan. 16, 2025. [Online]. Available: https://docs.unity3d.com/Manual/webgl-intro.html

[13] "3D Web-Based Experience Three.js vs Unity - Vaezr." Accessed: Jan. 16, 2025. [Online]. Available: https://vaezr.com/3d-web-based-experience-three-js-vs-unity/

[14] "Which is better for developing web-based 3D games - Three.js or Unity3D? - Quora." Accessed: Jan. 16, 2025. [Online]. Available: https://www.quora.com/Which-is-better-for-developing-web-based-3D-games-Three-js-or-Unity3D

[15] “Spring Framework Overview :: Spring Framework.” Accessed: Jan. 16, 2025. [Online]. Available: https://docs.spring.io/spring-framework/reference/overview.html

[16] “Endpoints :: Spring Boot.” Accessed: Jan. 16, 2025. [Online]. Available: https://docs.spring.io/spring-boot/reference/actuator/endpoints.html

[17] “Spring | Microservices.” Accessed: Jan. 16, 2025. [Online]. Available: https://spring.io/microservices

[18] “Spring | Blog.” Accessed: Jan. 16, 2025. [Online]. Available: https://spring.io/blog/category/releases

[19] “The History of the .NET Framework - Bocasay.” Accessed: Jan. 16, 2025. [Online]. Available: https://www.bocasay.com/history-net-framework/

[20]. “NET vs. Java: An Ultimate Comparison.” Accessed: Jan. 16, 2025. [Online]. Available: https://leobit.com/blog/net-vs-java-comparison-differences-similarities-and-use-cases/

[21] “Tags · nodejs/node-v0.x-archive.” Accessed: Jan. 26, 2025. [Online]. Available: https://github.com/nodejs/node-v0.x-archive/tags?after=v0.0.4

[22] “Node.js Overview: What is Node.js and Why It Matters.” Accessed: Jan. 26, 2025. [Online]. Available: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs

[23] “15+ Popular Companies Using Node.js in 2024.” Accessed: Jan. 26, 2025. [Online]. Available: https://www.simform.com/blog/companies-using-nodejs/

[24] “Introduction | Socket.IO.” Accessed: Jan. 26, 2025. [Online]. Available: https://socket.io/docs/v4/

[25] “Node.js Multithreading!. Multithreading in Node.js using the… | by Kareem Mohllal | Medium.” Accessed: Jan. 26, 2025. [Online]. Available: https://medium.com/@mohllal/node-js-multithreading-a5cd74958a67

[26] “About MariaDB Server - MariaDB.org.” Accessed: Jan. 27, 2025. [Online]. Available: https://mariadb.org/about/

[27] “Choosing the Right Storage Engine - MariaDB Knowledge Base.” Accessed: Jan. 27, 2025. [Online]. Available: https://mariadb.com/kb/en/choosing-the-right-storage-engine/

[28] “MariaDB vs MySQL - Difference Between Open Source Relational Databases - AWS.” Accessed: Jan. 27, 2025. [Online]. Available: https://aws.amazon.com/compare/the-difference-between-mariadb-vs-mysql/

[29] “List of Companies Using MariaDB, Market Share and Customers List.” Accessed: Jan. 27, 2025. [Online]. Available: https://www.infoclutch.com/installed-base/dbms/mariadb/

[30] “PostgreSQL: The world’s most advanced open source database.” Accessed: Jan. 27, 2025. [Online]. Available: https://www.postgresql.org/

[31] “PostgreSQL: Documentation: 17: 2. A Brief History of PostgreSQL.” Accessed: Jan. 27, 2025. [Online]. Available: https://www.postgresql.org/docs/current/history.html

[32] M. Stonebraker and L. A. Rowe, “THE DESIGN OF POSTGRES”.

[33] “PostgreSQL: Documentation: 17: 8.14. JSON Types.” Accessed: Jan. 27, 2025. [Online]. Available: https://www.postgresql.org/docs/current/datatype-json.html

[34] “PostgreSQL: Documentation: 17: 3.5. Window Functions.” Accessed: Jan. 27, 2025. [Online]. Available: https://www.postgresql.org/docs/current/tutorial-window.html

[35] “Optimizing PostgreSQL Performance with CTE, Partitioning, and Indexing | by Sanjeetkumar Mehta | Dec, 2024 | Medium.” Accessed: Jan. 27, 2025. [Online]. Available: https://medium.com/@sanjeetkmehta/optimizing-postgresql-performance-with-cte-partitioning-and-indexing-08f8540662df

[36] “Redis FAQ | Docs.” Accessed: Feb. 05, 2025. [Online]. Available: https://redis.io/docs/latest/develop/get-started/faq/

[37] “What is Redis Explained? | IBM.” Accessed: Feb. 05, 2025. [Online]. Available: https://www.ibm.com/think/topics/redis

[38] “The WebSocket API (WebSockets) - Web APIs | MDN.” Accessed: May 14, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[39] “Long Polling vs Server-Sent Events vs WebSockets: A Comprehensive Guide | by Asharsaleem | Medium.” Accessed: May 14, 2025. [Online]. Available: https://medium.com/@asharsaleem4/long-polling-vs-server-sent-events-vs-websockets-a-comprehensive-guide-fb27c8e610d0

[40] “Docker Desktop | Docker Docs.” Accessed: May 23, 2025. [Online]. Available: https://docs.docker.com/desktop/

[41] “Docker Compose | Docker Docs.” Accessed: May 23, 2025. [Online]. Available: https://docs.docker.com/compose/

[42] “REST API Documentation Tool | Swagger UI.” Accessed: May 29, 2025. [Online]. Available: https://swagger.io/tools/swagger-ui/

[43] “draw.io.” Accessed: May 14, 2025. [Online]. Available: https://app.diagrams.net/

[44] “Mermaid | Diagramming and charting tool.” Accessed: May 14, 2025. [Online]. Available: https://mermaid.js.org/

[45] “What is the MoSCoW Method?” Accessed: May 14, 2025. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method

[46] M. E. Rana and O. S. Saleh, “High assurance software architecture and design,” *System Assurances: Modeling and Management*, pp. 271–285, Jan. 2022, doi: 10.1016/B978-0-323-90240-3.00015-1.

[47] “Separation of Concerns (SoC) | GeeksforGeeks.” Accessed: May 14, 2025. [Online]. Available: https://www.geeksforgeeks.org/separation-of-concerns-soc/

[48] “SOLID Principles in Programming: Understand With Real Life Examples | GeeksforGeeks.” Accessed: May 14, 2025. [Online]. Available: https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/

[49] R. C. Martin, *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*, 1st ed. USA: Prentice Hall Press, 2017.

[50] “Event driven architecture: the good, the bad, and the ugly | Equal Experts.” Accessed: May 14, 2025. [Online]. Available: https://www.equalexperts.com/blog/tech-focus/event-driven-architecture-the-good-the-bad-and-the-ugly/

[51] “Designing for Failure. Building Resilient Systems | itversity.” Accessed: May 14, 2025. [Online]. Available: https://medium.com/itversity/designing-for-failure-60e0131c3398

[52] “Separating business and UI logic (the proper way) | by Jacob Sikorski | UX Collective.” Accessed: May 14, 2025. [Online]. Available: https://uxdesign.cc/separating-business-and-ui-logic-the-proper-way-a58a64529333

[53] “C4 model for visualising… by Simon Brown [PDF/iPad/Kindle].” Accessed: May 14, 2025. [Online]. Available: https://leanpub.com/visualising-software-architecture

[54] “Snake case - MDN Web Docs Glossary: Definitions of Web-related terms | MDN.” Accessed: May 22, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Snake_case

[55] “Camel case - MDN Web Docs Glossary: Definitions of Web-related terms | MDN.” Accessed: May 22, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Camel_case

[56] “The Benefits of Failing Fast. How do you handle an unexpected failure… | by Denat Hoxha | Medium.” Accessed: May 22, 2025. [Online]. Available: https://medium.com/@denhox/the-benefits-of-fail-fast-systems-dc72a665cfb5

[57] “CQRS.” Accessed: May 22, 2025. [Online]. Available: https://martinfowler.com/bliki/CQRS.html

[58] “What is a CSRF Token and How Does It Work? - Bright Security.” Accessed: May 22, 2025. [Online]. Available: https://www.brightsec.com/blog/csrf-token/

[59] “History: The Agile Manifesto.” Accessed: May 23, 2025. [Online]. Available: https://agilemanifesto.org/history.html

[60] “The New Methodology.” Accessed: May 23, 2025. [Online]. Available: https://www.martinfowler.com/articles/newMethodology.html

[61] M. Rohland VICE *et al.*, “Isaac Kasbdan REGIONAL VICE·PRESIDENTS ‘eter t..bde GRI!AT LAK.’’””.

[62] “OAuth 2.0 — OAuth.” Accessed: May 23, 2025. [Online]. Available: https://oauth.net/2/