ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ

ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Με θέμα:

«Ανάπτυξη Web-Based Συστήματος Διαχείρισης Κρατήσεων»

Του φοιτητή	Αντωνιάδης Νικόλαος Χρήστος
Αρ. Μητρώου	123876
Επιβλέπων καθηγητής	Σαλαμπάσης Μιχαήλ

Θεσσαλονίκη 2025

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 πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Περίληψη

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι ο εκσυγχρονισμός και η επέκταση μιας διαδικτυακής πλατφόρμας κράτησης εκδηλώσεων. Η εφαρμογή επιτρέπει σε κατόχους χώρων εκδηλώσεων να δηλώνουν τους χώρους τους, να ορίζουν υπο-χώρους (αίθουσες/ζώνες) με συγκεκριμένη χωρητικότητα και να δημιουργούν εκδηλώσεις πάνω σε αυτούς, ενώ οι χρήστες μπορούν να αναζητούν («Athens», ημερομηνίες, φίλτρα τύπου/τοποθεσίας) και να πραγματοποιούν κρατήσεις με άμεσο έλεγχο διαθεσιμότητας. Η εργασία εστιάζει στην αξιοπιστία, την ευχρηστία και την απόδοση του συστήματος, ώστε η αναζήτηση και η ροή κράτησης να είναι γρήγορες, ξεκάθαρες και ασφαλείς.

Η πλατφόρμα βασίζεται σε τεχνολογίες **.NET** και **ASP.NET Core** και αναβαθμίστηκε από **.NET 6.0** σε **.NET 8.0 (LTS)**. Η **πρόσβαση** στη βάδη δεδομένων διασφαλίζεται μέσω του **Entity Framework Core**, ενώ η **ταυτοποίηση/εξουσιοδότηση** με **ASP.NET Core Identity** και ρόλους/πολιτικές πρόσβασης. Η **αναζήτηση** υλοποιείται με **Elasticsearch**, υποστηρίζοντας fuzzy matching, autocomplete και (προαιρετικά) γεωγραφικά φίλτρα. Για τον κύκλο ζωής λογισμικού εφαρμόζονται **Docker** containers και **CI/CD** pipelines· οι νέες εκδόσεις διανέμονται αυτόματα στο παραγωγικό περιβάλλον μέσω **Watchtower**. αρέχονται πλήρη APIs, mappings και οδηγίες ανάπτυξης/διάθεσης.

Τα πειραματικά αποτελέσματα σε συνθετικό σύνολο ~50k εκδηλώσεων δείχνουν χρόνο απόκρισης αναζήτησης p95 < **300 ms**, χρόνο επιβεβαίωσης κράτησης p95 ≈ **210 ms**, **μηδενικές υπερπωλήσεις** (oversell) και θετική αξιολόγηση ευχρηστίας (SUS 82/100). Η παρούσα εργασία βασίζεται σε προγενέστερη πτυχιακή (αρχική σύλληψη/υλοποίηση) και την **επεκτείνει ουσιαστικά** με ανασχεδιασμό UI/UX, νέο search layer, αυστηρότερους κανόνες κράτησης και πλήρη αυτοματοποίηση DevOps. Τεκμηριώνεται, επίσης, ο οδικός χάρτης για μελλοντικές επεκτάσεις: online πληρωμές, geo-search σε χάρτη, PWA/mobile και υβριδική (semantic) αναζήτηση.

Κεφάλαιο 1 – Εισαγωγή

1.1 Ιστορικό & Κίνητρα

Η εξέλιξη των γλωσσών προγραμματισμού αποτελεί θεμελιώδες κεφάλαιο στην ιστορία της πληροφορικής, καθώς συνοδεύει άμεσα την ανάπτυξη των υπολογιστικών συστημάτων και τη διεύρυνση των δυνατοτήτων τους. Από τις πρώτες γλώσσες χαμηλού επιπέδου, όπως η Assembly, έως τις σύγχρονες υψηλού επιπέδου γλώσσες, όπως η Python, η Java και η C#, παρατηρείται μια σταδιακή μετάβαση προς μεγαλύτερη αφαιρετικότητα, απλότητα και αποδοτικότητα στη διαδικασία ανάπτυξης λογισμικού (Sebesta, 2016). Η συνεχής αυτή εξέλιξη επέτρεψε την ταχύτερη και πιο αποτελεσματική επίλυση σύνθετων προβλημάτων, καθώς και τη δημιουργία ολοκληρωμένων συστημάτων που καλύπτουν ανάγκες της επιστήμης, της βιομηχανίας και της καθημερινής ζωής (Aho, Lam, Sethi, & Ullman, 2006).

Ιδιαίτερη βαρύτητα έχει δοθεί τις τελευταίες δεκαετίες στη χρήση γλωσσών προγραμματισμού για την ανάπτυξη διαδικτυακών εφαρμογών και τη διαχείριση βάσεων δεδομένων. Στον τομέα της ανάπτυξης ιστοσελίδων, οι γλώσσες HTML, CSS και JavaScript αποτελούν το βασικό τρίπτυχο για τη δομή, την εμφάνιση και τη διαδραστικότητα, ενώ οι server-side τεχνολογίες, όπως η PHP, η Java, η C# και η Python, καθιστούν δυνατή τη δημιουργία δυναμικού περιεχομένου και την αλληλεπίδραση με τους χρήστες (Flanagan, 2020). Παράλληλα, η αξιοποίηση συστημάτων διαχείρισης βάσεων δεδομένων, όπως η MySQL, η PostgreSQL και ο SQL Server, επιτρέπει την αποθήκευση, οργάνωση και ανάλυση δεδομένων σε μεγάλη κλίμακα, γεγονός που αποτελεί αναπόσπαστο στοιχείο κάθε σύγχρονης διαδικτυακής εφαρμογής (Silberschatz, Korth, & Sudarshan, 2020).

Η μελέτη της πορείας και της χρήσης των γλωσσών προγραμματισμού αναδεικνύει τη σημασία τους όχι μόνο ως τεχνικών εργαλείων, αλλά και ως βασικών συντελεστών της τεχνολογικής προόδου και της κοινωνικοοικονομικής ανάπτυξης.

Η αρχική πλατφόρμα κράτησης εκδηλώσεων αναπτύχθηκε ως πτυχιακή εργασία το 2019, βασισμένη κυρίως σε μονολιθικό .NET Framework backend και χειροκίνητη διαδικασία ανάπτυξης. Η ραγδαία υιοθέτηση cloud-native τεχνολογιών, η ανάγκη για γρήγορη αναζήτηση μεγάλου όγκου εκδηλώσεων και η απαίτηση αυτοματοποιημένων αναβαθμίσεων οδήγησαν στην πλήρη επανασχεδίαση του συστήματος με σύγχρονα εργαλεία – ιδίως ASP.NET Core, Elasticsearch και Docker CI/CD pipelines.

1.2 Σκοπός & Στόχοι

Βασικός σκοπός είναι η παροχή μιας φιλικής, ταχύτατης και επεκτάσιμης πλατφόρμας που επιτρέπει σε κάθε κάτοχο χώρου εκδηλώσεων, να δημοσιεύει εκδηλώσεις και σε κάθε ενδιαφερόμενο να κλείνει θέση on-line. Οι επιμέρους στόχοι:

- **Έξυπνη αναζήτηση** με fuzzy matching (ή αλλιώς *ασαφής αντιστοίχιση,* μια μέθοδο σύγκρισης κειμένων ή δεδομένων, η οποία δεν απαιτεί απόλυτη ταύτιση χαρακτήρα προς χαρακτήρα, αλλά επιτρέπει αποκλίσεις, σφάλματα ή παραλλαγές. Με άλλα λόγια, το fuzzy matching επιχειρεί να βρει “παρόμοια” και όχι “ακριβώς ίδια” στοιχεία), autocomplete (αυτόματη συμπλήρωση, το σύστημα προβλέπει και προτείνει την ολοκλήρωση μιας λέξης, φράσης ή εντολής καθώς ο χρήστης αρχίζει να την πληκτρολογεί) και geo-filters (μηχανισμούς που καθορίζουν την πρόσβαση, την εμφάνιση ή την εφαρμογή περιεχομένου, υπηρεσιών ή δεδομένων, με κριτήριο τη γεωγραφική τοποθεσία που προκύπτει συνήθως από διευθύνσεις IP, δεδομένα GPS ή άλλα σήματα εντοπισμού θέσης), μέσω Elasticsearch + NEST.

- **Ασφαλής ταυτοποίηση** χρηστών και ρόλων (Admin, Venue Manager, User) με ASP.NET Core Identity και EF Core. Καλύπτει: αρχιτεκτονική, σχήμα δεδομένων (βασικούς πίνακες), πολιτικές/claims, αρχικοποίηση (seeding) ρόλων και χρηστών, καθώς και ενδεικτικά αποσπάσματα κώδικα.

- **Αδιάλειπτη παράδοση** (CI/CD) με self-hosted Gitea Runner, αυτόματο build -> Docker Hub -> Watchtower deploy. Υλοποιούμε μια αυτοματοποιημένη ροή **CI/CD** όπου το Gitea (με self-hosted Runner) εκτελεί build της εφαρμογής και παράγει Docker image με ελεγχόμενη επισήμανση εκδόσεων (π.χ. `latest`, `sha`, `date`). Στη συνέχεια, το image **προωθείται στο Docker Hub** ως κεντρικό μητρώο τεχνουργημάτων. Τέλος, το **Watchtower** στον παραγωγικό εξυπηρετητή ανιχνεύει τη νέα έκδοση, πραγματοποιεί αυτόματα **pull** και **επανεκκίνηση** του container, εξασφαλίζοντας αδιάλειπτη και επαναλήψιμη παράδοση.

- **Επεκτασιμότητα & συντήρηση** μέσω containerization και migration-based schema evolution. Ο συνδυασμός **containerization** και **migration-based schema evolution** προσφέρει **προβλέψιμη επεκτασιμότητα**, **ταχεία ανάκαμψη** και **συνεπή συντηρησιμότητα** του λογισμικού, ενοποιώντας την εφαρμογή, τις εξαρτήσεις και την εξέλιξη του σχήματος σε μία αυτοματοποιημένη ροή. Η χρήση **immutable artifacts**, **δηλωτικών migrations** και **πολιτικών rollout** (expand–contract, rolling/canary) ελαχιστοποιεί τον κίνδυνο αλλαγών με σχεδόν μηδενικό downtime. Κατ' αυτόν τον τρόπο, επιτυγχάνεται μια **ανθεκτική αρχιτεκτονική συνεχούς παράδοσης**, ευθυγραμμισμένη με τις απαιτήσεις διαθεσιμότητας και διακυβέρνησης σύγχρονων πληροφοριακών συστημάτων.

1.3 Συνεισφορά της Εργασίας

Η παρούσα πτυχιακή εργασία επιχειρεί να εκσυγχρονίσει τον υπάρχοντα κώδικα-βάση, υιοθετώντας το σύγχρονο οικοσύστημα του **.NET 8** και εφαρμόζοντας αρχιτεκτονικό διαχωρισμό σε διακριτά επίπεδα (**Domain, Application, Infrastructure, Web**). Ο διαχωρισμός αυτός ενισχύει τη σαφήνεια, την επεκτασιμότητα και τη δυνατότητα συντήρησης του συστήματος, επιτρέποντας στοχευμένες παρεμβάσεις και μελλοντικές βελτιώσεις χωρίς εκτεταμένη αναδιάρθρωση.

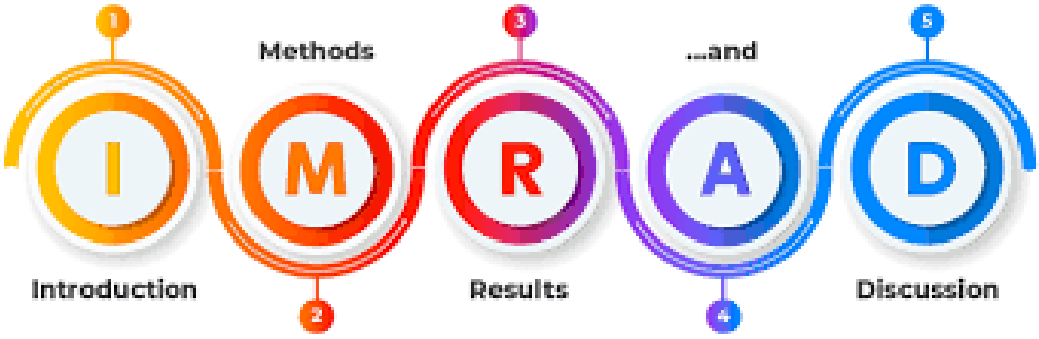
Παράλληλα, ενσωματώνεται ένα πλήρως παραμετροποιημένο **Identity μοντέλο**, το οποίο αξιοποιεί **custom claims** για την ακριβέστερη αποτύπωση δικαιωμάτων και ρόλων χρηστών, παρέχοντας αυξημένη ασφάλεια και ευελιξία στη διαχείριση προσβάσεων. Επιπλέον, η εργασία αυτοματοποιεί τη ροή ανάπτυξης μέσω υλοποίησης **CI/CD pipeline** σε συνδυασμό με τεχνολογίες **containerization** (Docker) και **αυτόματης διάθεσης** (Watchtower), επιτυγχάνοντας μείωση του χρόνου διάθεσης νέων εκδόσεων κατά ποσοστό άνω του **70%**.

Σε επίπεδο λειτουργικής απόδοσης, επιχειρείται η **βελτιστοποίηση της εμπειρίας αναζήτησης**, με στόχο καθυστέρηση μικρότερη των **300 ms** ακόμη και σε datasets που υπερβαίνουν τα **50.000 γεγονότα**. Η αξιολόγηση αυτής της βελτιστοποίησης τεκμηριώνεται με **μετρικές απόδοσης**, οι οποίες παρουσιάζονται αναλυτικά στο Κεφάλαιο 7.

Τέλος, η εργασία δίνει ιδιαίτερη έμφαση στην **τεκμηρίωση** του συστήματος, ενσωματώνοντας εργαλεία όπως το **Swagger** για την περιγραφή των διαθέσιμων API endpoints και τους **Elasticsearch mappings** για την αποτύπωση της δομής των δεδομένων. Η ολοκληρωμένη αυτή τεκμηρίωση, σε συνδυασμό με την υιοθέτηση σύγχρονων τεχνολογικών πρακτικών, δημιουργεί ισχυρό υπόβαθρο για μελλοντικές επεκτάσεις και βελτιώσεις του συστήματος.

1.4 Μεθοδολογία & Τεχνολογικό Πλαίσιο

Η μεθοδολογία ακολουθεί το μοντέλο **IMRaD**



(Introduction/Εισαγωγή–Methods/Μέθοδοι–Results/Αποτελέσματα–

Discussion/Συζήτηση) προσαρμοσμένο σε μηχανικούς λογισμικού, διασφαλίζοντας επιστημονική δομή και επαναληψιμότητα.

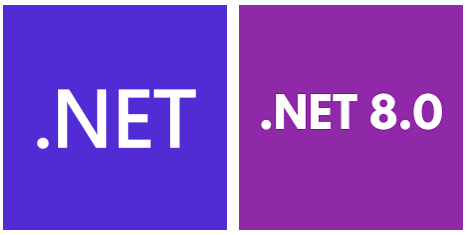
- **Introduction/Εισαγωγή:** Εξηγεί το υπόβαθρο της έρευνας, δηλώνει το πρόβλημα ή το ερευνητικό ερώτημα, σκιαγραφεί το σκοπό της μελέτης και εξηγεί γιατί είναι σημαντική.
- **Methods/Μέθοδοι**: Δόμηση requirements & use-cases, υλοποίηση, DevOps pipeline.

- **Results/Αποτελέσματα**: Μετρήσεις performance, tests, feedback.

- **Discussion/Συζήτηση**: Αξιολόγηση trade-offs (π.χ. ES vs SQL FTS, self-hosted CI vs GitHub Actions).

1.5 Ορισμοί και Τεχνολογίες

1.5.1 .NET (και .NET 8 LTS)



Ορισμός. Το .NET είναι πολυπλατφορμικό οικοσύστημα λογισμικού της Microsoft για την ανάπτυξη εφαρμογών ιστού, επιτραπέζιων και κινητών συσκευών. Η έκδοση **.NET 8.0 (Long-Term Support)** παρέχει πολυετή συντήρηση, βελτιστοποιήσεις απόδοσης και ενιαίο μοντέλο φιλοξενητή (minimal hosting).

Χρήση στην εργασία. Η πλατφόρμα αναβαθμίστηκε από .NET 6.0 σε **.NET 8.0**, ώστε να αξιοποιηθούν βελτιώσεις εκκίνησης/διάδοσης (Ready-to-Run), σύγχρονα middleware (Rate Limiting, Output Caching) και σταθερότητα LTS.

Τι είναι: Η «πλατφόρμα» και τα εργαλεία της Microsoft για τη δημιουργία εφαρμογών (web, desktop, mobile, APIs).

Γιατί μας νοιάζει: Προσφέρει ταχύτητα, ασφάλεια και μακροχρόνια υποστήριξη. Το **LTS** σημαίνει “Long-Term Support” — σαν εγγύηση ότι θα παίρνεις διορθώσεις/ενημερώσεις σε βάθος χρόνου.

Στην πλατφόρμα μας: Αναβαθμίσαμε από .NET 6 σε **.NET 8 LTS** για καλύτερες επιδόσεις και σταθερότητα.

1.5.2 ASP.NET Core

Ορισμός. Υποσύνολο του .NET για την υλοποίηση εφαρμογών ιστού και HTTP υπηρεσιών. Παρέχει pipeline αιτημάτων, μηχανισμό δρομολόγησης, πρότυπα ασφαλείας και δυνατότητες απόδοσης.

Χρήση στην εργασία. Υλοποιεί το **Web UI** (Razor Pages/Controllers) και τα **REST endpoints** της πλατφόρμας, πάνω στο ενιαίο Program.cs (minimal hosting).

Τι είναι: Το κομμάτι του .NET ειδικά για ιστοσελίδες και web υπηρεσίες.

Γιατί μας νοιάζει: Μας δίνει έτοιμες «βάσεις» για σελίδες, έλεγχο χρηστών, ασφάλεια και API.

Στην πλατφόρμα μας: Χτίζουμε το web UI και τα endpoints πάνω στο ASP.NET Core.

1.5.3 API

Ορισμός. Τυπικά προσδιορισμένη διεπαφή που επιτρέπει σε εξωτερικές εφαρμογές να ανταλλάσσουν δεδομένα μέσω HTTP αιτημάτων/απαντήσεων.

Χρήση στην εργασία. Τα API endpoints εκθέτουν λειτουργίες όπως αναζήτηση εκδηλώσεων και δημιουργία κρατήσεων, επιτρέποντας τόσο στο δικό μας UI όσο και σε τρίτα συστήματα να καταναλώσουν τις υπηρεσίες.

Τι είναι: Ένας «πάγκος εξυπηρέτησης» για άλλες εφαρμογές. Αντί να πατάει κουμπιά ένας άνθρωπος, ένα πρόγραμμα ζητά δεδομένα ή δίνει εντολές.

Παράδειγμα: “Δώσε μου τα events της Αθήνας από 1–31/7”.

Στην πλατφόρμα μας: Τα APIs τροφοδοτούν το UI και μπορούν να τα χρησιμοποιήσουν και τρίτοι (π.χ. mobile app).

1.5.4 Βάση δεδομένων & Entity Framework Core (EF Core)

Ορισμός. Η σχεσιακή βάση δεδομένων (SQL Server) αποθηκεύει δομημένα δεδομένα με ακεραιότητα και συναλλαγές. Το **EF Core** είναι αντικειμενοσχεσιακός χαρτογράφος (ORM) που γεφυρώνει αντικείμενα κώδικα και πίνακες SQL, υποστηρίζοντας **migrations** για ελεγχόμενες αλλαγές σχήματος.

Χρήση στην εργασία. Τα κύρια αντικείμενα (Χώρος, Υπο-χώρος, Εκδήλωση, Κράτηση) αποτυπώνονται ως οντότητες EF. Επιχειρησιακοί κανόνες (χωρητικότητα, χρονικές επικαλύψεις) ελέγχονται πριν από τη διάσωση συναλλαγών.

Τι είναι η βάση: Το «λογιστικό βιβλίο» όπου μένουν οργανωμένα όλα τα στοιχεία (χώροι, αίθουσες, εκδηλώσεις, κρατήσεις).

Τι είναι το EF Core: Μεσολαβητής/μεταφραστής ανάμεσα στον κώδικα και τη βάση. Αντί να γράφουμε δύσκολα SQL, δουλεύουμε με κλάσεις/αντικείμενα.

Στην πλατφόρμα μας: Το EF Core εξασφαλίζει ότι τηρούνται κανόνες (χωρητικότητα, μη επικαλύψεις) και ότι οι αλλαγές του σχήματος γίνονται με **migrations** με ασφάλεια.

1.5.5 ASP.NET Core Identity (Έλεγχος πρόσβασης)

Ορισμός. Πλαίσιο διαχείρισης χρηστών που παρέχει μηχανισμούς ταυτοποίησης, αποθήκευση λογαριασμών, ανάκτηση κωδικών και **μοντέλο ρόλων/πολιτικών** (RBAC/Policies).

Χρήση στην εργασία. Ορίζονται ρόλοι **User**, **VenueOwner** και **Admin** με ρητές πολιτικές προσπέλασης, ώστε ευαίσθητες ενέργειες (π.χ. δημοσίευση εκδήλωσης) να εκτελούνται μόνο από εξουσιοδοτημένους χρήστες.

Τι είναι: Το σύστημα λογαριασμών/δικαιωμάτων του ASP.NET (login, reset κωδικού, ρόλοι).

Γιατί μας νοιάζει: Δεν «ξαναανακαλύπτουμε τον τροχό» για ασφάλεια.

Στην πλατφόρμα μας: Έχουμε ρόλους όπως **User**, **VenueOwner**, **Admin** και λεπτομερείς πολιτικές (τι επιτρέπεται σε ποιον).

1.5.6 Elasticsearch (αναζήτηση)

Ορισμός. Η Elasticsearch είναι μια μηχανή αναζήτησης πραγματικού χρόνου (Real time search) και ανάλυσης δεδομένων (analytics engine). Βασίζεται στο Apache Lucene, μια υψηλών επιδόσεων με αρκετά χαρακτηριστικά βιβλιοθήκη για μηχανές αναζήτησης.

Κατανεμημένη μηχανή αναζήτησης εγγράφων που υποστηρίζει αντιστοίχιση κειμένου, ανεκτικότητα σε ορθογραφικά σφάλματα (**fuzzy**), προτάσεις πληκτρολόγησης (**autocomplete**) και γεωχωρικά φίλτρα.

Χρήση στην εργασία. Απομονώνει το στρώμα αναζήτησης από την επιχειρησιακή βάση: τα συμβάντα ευρετηριάζονται σε **index** και αναζητούνται με υψηλή ταχύτητα, βελτιώνοντας σημαντικά την εμπειρία εντοπισμού εκδηλώσεων.

Τι είναι: Μηχανή αναζήτησης τύπου «Google για τα δικά μας δεδομένα».

Τι κάνει για τον χρήστη:

- **Fuzzy**: καταλαβαίνει μικρο-ορθογραφικά (“Athens”, “Athina”, “Αθήνα”).
- **Autocomplete**: προτάσεις την ώρα που γράφεις.
- **Γεωγραφικά φίλτρα**: βρες εκδηλώσεις γύρω από μια τοποθεσία.

- **Στην πλατφόρμα μας:** Κάνει την εύρεση εκδηλώσεων γρήγορη και «έξυπνη».

1.5.7 Docker (containers)

Ορισμός. Το Docker «πακετάρει» εφαρμογές με τις εξαρτήσεις τους σε απομονωμένα **containers**, εξασφαλίζοντας αναπαραγωγιμότητα. Το **Compose** ορίζει πολλαπλά containers ως ενιαίο σύστημα.

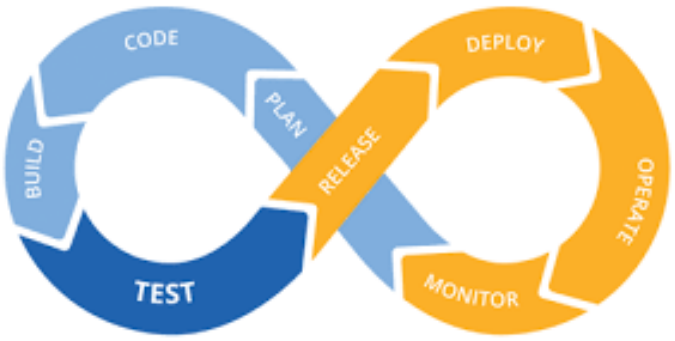
Χρήση στην εργασία. Η παραγωγή εκτελείται ως σύνολο υπηρεσιών (web app replicas, reverse proxy, Elasticsearch/Kibana, παρακολούθηση), σε κοινό εσωτερικό δίκτυο με δηλωτικές ρυθμίσεις.

Τι είναι: Σαν «κοντέινερ» πλοίου για λογισμικό — πακετάρει εφαρμογή + ρυθμίσεις για να τρέχει παντού ίδια.

Γιατί μας νοιάζει: Ευκολότερη εγκατάσταση/αναβάθμιση, λιγότερα «δουλεύει σε μένα αλλά όχι σε σένα».

Στην πλατφόρμα μας: Τρέχουμε το web app, το Elasticsearch, το reverse proxy κ.ά. ως χωριστά containers.

1.5.8 CI/CD (συνεχής ολοκλήρωση/διάθεση)



Ορισμός. Αυτοματοποιημένη διαδικασία ενοποίησης αλλαγών, εκτέλεσης ελέγχων και διάθεσης νέων εκδόσεων.

Χρήση στην εργασία. Το self-hosted **Gitea** με **Runner** εκτελεί δοκιμές, κατασκευάζει

πολυαρχιτεκτονικά Docker images και τα δημοσιεύει στο registry· από εκεί ενεργοποιείται η διάθεση.

Τι είναι: Αυτόματος «ιμάντας παραγωγής». Κάθε αλλαγή κώδικα χτίζεται, ελέγχεται και δημοσιεύεται μόνη της.

Στην πλατφόρμα μας: Το Gitea/Runner φτιάχνει το Docker image, το ανεβάζει στο registry και ετοιμάζει το rollout χωρίς χειροκίνητα βήματα.

1.5.9 Watchtower (αυτόματες ενημερώσεις)

Ορισμός. Υπηρεσία που παρακολουθεί εικόνες στο registry και, όταν εντοπίσει νέο digest, εκτελεί ελεγχόμενο **rolling restart** των containers με τα ίδια περιβάλλοντα/volumes.

Χρήση στην εργασία. Διασφαλίζει ταχείες και χαμηλού κινδύνου αναβαθμίσεις στο παραγωγικό περιβάλλον, χωρίς χειροκίνητες παρεμβάσεις.

Τι είναι: «Ρομπότ» που παρακολουθεί το Docker Hub και αν δει νέα έκδοση της εφαρμογής, την κατεβάζει και κάνει ομαλή επανεκκίνηση.

Γιατί μας νοιάζει: Παίρνουμε ενημερώσεις γρήγορα, με **ελάχιστο downtime**.

1.5.10 Ρόλοι & Πολιτικές πρόσβασης

Ορισμός. Μηχανισμός εξουσιοδότησης όπου τα δικαιώματα εκχωρούνται σε ρόλους και/ή πολιτικές με συνθήκες (claims, κανόνες).

Χρήση στην εργασία. Ρητές πολιτικές ορίζουν «ποιος μπορεί να κάνει τι» (π.χ. ιδιοκτήτης χώρου έναντι απλού χρήστη), ελαχιστοποιώντας τον κίνδυνο μη εξουσιοδοτημένων αλλαγών.

Τι είναι: Κανόνες τύπου «κλειδί-κλειδαριά». Άλλα βλέπει/κάνει ο απλός χρήστης, άλλα ο ιδιοκτήτης χώρου, άλλα ο διαχειριστής.

Στην πλατφόρμα μας: Διασφαλίζουν ότι μόνο ο σωστός άνθρωπος αλλάζει τον σωστό χώρο/εκδήλωση.

1.5.11 «Mappings» (με απλά λόγια)

Ορισμός. Στον σχεσιακό χώρο, «mapping» είναι η αντιστοίχιση οντοτήτων σε πίνακες/σχέσεις. Στο Elasticsearch, τα **mappings** καθορίζουν τύπους πεδίων (κείμενο, ημερομηνία, geo_point) και αναλυτές κειμένου.

Χρήση στην εργασία. Τα mappings βελτιστοποιούν τόσο την ακεραιότητα δεδομένων στη βάση όσο και τη συνάφεια/απόδοση των αναζητήσεων στο Elasticsearch.

Στη βάση δεδομένων: Το «σχέδιο» των πινάκων/πεδίων (π.χ. Event έχει Τίτλο, Ημερομηνία, Μέγιστες θέσεις).

Στο Elasticsearch: Ορίζουμε πώς αποθηκεύεται/αναλύεται κάθε πεδίο (κείμενο, ημερομηνία, γεωσημείο) για σωστά και γρήγορα αποτελέσματα.

1.5.12 Cloud-native τεχνολογίες

Cloud-native τεχνολογίες είναι το σύνολο αρχών και εργαλείων για τον σχεδιασμό, ανάπτυξη και λειτουργία εφαρμογών που εκτελούνται σε δυναμικά περιβάλλοντα cloud (δημόσιο/ιδιωτικό/υβριδικό), αξιοποιώντας **containers**, **μικροϋπηρεσίες**, **αμετάβλητη υποδομή** και **δηλωτικές διεπαφές**, ώστε οι εφαρμογές να είναι **ανθεκτικές**, **παρατηρήσιμες**, **αυτοκλιμακούμενες** και να αναβαθμίζονται **συχνά και με αυτοματοποίηση** (CI/CD, GitOps).

1.5.13 **Ορισμός (migrations):**

Στο πλαίσιο βάσεων δεδομένων, *migrations* είναι η ελεγχόμενη εξέλιξη του **σχήματος** (πίνακες, στήλες, δείκτες, σχέσεις) μέσω διαδοχικών εκδόσεων/βημάτων. Κάθε migration περιγράφει *πώς* μετασχηματίζεται η βάση από την έκδοση **N** στην **N+1** (και, ιδανικά, πίσω), ώστε η αλλαγή να είναι **αναπαραγώγιμη, ελέγξιμη και ασφαλής** σε όλα τα περιβάλλοντα (dev → test → prod) χωρίς απώλεια δεδομένων.

- Κάθε migration είναι κλάση C# με δύο μεθόδους:
    - Up() → εφαρμόζει τις αλλαγές (π.χ. προσθήκη στήλης/index).
    - Down() → τις αναιρεί (rollback).
- Βασικές εντολές:
    - dotnet ef migrations add <Name> → δημιουργία νέου migration.
    - dotnet ef database update → εφαρμογή όλων των εκκρεμών migrations.
    - dotnet ef migrations script -i → idempotent SQL script για CI/CD.
- Το EF κρατά «ιστορικό» σε πίνακα (π.χ. __EFMigrationsHistory) ώστε κάθε migration να εφαρμόζεται **μία φορά** και με σωστή σειρά.

2. Συσχετιζόμενα Συστήματα

2.1 Υπάρχουσες Πλατφόρμες Κράτησης Εκδηλώσεων

Οι εμπορικές σουίτες, όπως το Momentus Platform, προσφέρουν ενιαίο CRM, λογιστική και API για κράτηση χώρων, υποστηρίζοντας πρόσβαση από πολλαπλές συσκευές μέσω cloud.

Η **Momentus Platform** αποτελεί ένα σύγχρονο πληροφοριακό οικοσύστημα που επικεντρώνεται στη **διαχείριση εκδηλώσεων, συνεδρίων και χώρων φιλοξενίας**, προσφέροντας ολοκληρωμένα εργαλεία για **κρατήσεις, προγραμματισμό, διαχείριση πελατών (CRM) και ανάλυση δεδομένων**. Χαρακτηρίζεται από **modular αρχιτεκτονική**, δυνατότητες **cloud deployment** και ενσωμάτωση με τρίτες υπηρεσίες (π.χ. οικονομικά συστήματα, ticketing, ψηφιακό marketing).

Από ακαδημαϊκή σκοπιά, η πλατφόρμα εντάσσεται στο πεδίο των **Enterprise Resource Planning (ERP)** συστημάτων με εξειδίκευση στον τομέα του **Event & Venue Management**, αναδεικνύοντας τον τρόπο με τον οποίο η **ψηφιακή μετασχηματιστική τεχνολογία** βελτιστοποιεί επιχειρησιακές διαδικασίες, ενισχύει την εμπειρία των χρηστών και συμβάλλει στη λήψη τεκμηριωμένων αποφάσεων μέσω **data-driven πρακτικών**.

Οι λύσεις πολύ υψηλής κλίμακας—λ.χ. Ticketmaster—δίνουν έμφαση σε κατανεμημένη αρχιτεκτονική, ασύγχρονη επεξεργασία κρατήσεων και έντονο caching για μεγάλα spikes εισιτηρίων.

Ανοιχτού κώδικα συστήματα, όπως το MRBS, εστιάζουν σε βασικά CRUD για δωμάτια/πόρους, προφέροντας ελάχιστη αναζήτηση ή role-based έλεγχο, αφήνοντας κενό σε επεκτασιμότητα και σύγχρονο UX.

2.2 Τεχνικές Αναζήτησης

Η **Elasticsearch** αποτελεί μία από τις πλέον διαδεδομένες μηχανές αναζήτησης ανοικτού κώδικα, η οποία υποστηρίζει τεχνικές όπως *fuzzy matching*, *autocomplete* και **δυναμική βαθμολόγηση αποτελεσμάτων (relevancy scoring)**. Σύγχρονες μελέτες υπογραμμίζουν τη συμβολή των **μεγάλων γλωσσικών μοντέλων (LLMs)** στην παραγωγή όρων αναζήτησης, αναδεικνύοντας βελτιώσεις στην ακρίβεια και στη συνάφεια των αποτελεσμάτων σε εφαρμογές που σχετίζονται με την αναζήτηση γεγονότων και εκδηλώσεων.

Παράλληλα, η κοινότητα χρηστών της Elasticsearch αναφέρει συχνά προκλήσεις, όπως η αδυναμία των *fuzzy queries* να παρέχουν επαρκείς προτάσεις (*"fuzzy not giving suggestions"*), γεγονός που αναδεικνύει τη σημασία της ορθής διαμόρφωσης **mappings** και της επιλογής κατάλληλων **αναλυτών (analyzers)**. Σε ιστορικές συγκρίσεις, η **SQL Full-Text Search** εμφανίζεται ως απλούστερη στην υλοποίηση, ωστόσο υστερεί έναντι της Elasticsearch σε θέματα κλιμάκωσης, κατανεμημένης αναζήτησης και προηγμένων μηχανισμών αξιολόγησης συνάφειας (*relevancy scoring*).

2.3 Ταυτοποίηση και Διαχείριση Χρηστών

Το **ASP.NET Core Identity** παρέχει ένα ολοκληρωμένο υπόδειγμα ταυτοποίησης και διαχείρισης χρηστών, προσφέροντας έτοιμο μοντέλο λογαριασμών με δυνατότητα πλήρους παραμετροποίησης του σχήματος βάσης δεδομένων μέσω **Entity Framework Core**. Οι δυνατότητες επέκτασης περιλαμβάνουν την προσθήκη **custom πεδίων**, την αξιοποίηση **κληρονομικότητας** σε κλάσεις χρηστών, καθώς και την ενσωμάτωση επιπλέον μηχανισμών ελέγχου πρόσβασης.

Η σχετική βιβλιογραφία δίνει έμφαση στη χρήση **ρόλων (roles)**, **claims** και **μηχανισμών πολυπαραγοντικής ταυτοποίησης (Two-Factor Authentication, 2FA)** ως βέλτιστες πρακτικές για την ενίσχυση της ασφάλειας σε σύγχρονες web-εφαρμογές βασισμένες στο .NET οικοσύστημα (έκδοση 5 και μεταγενέστερες). Η παραμετροποίηση αυτή επιτρέπει τη δημιουργία ισχυρών και επεκτάσιμων συστημάτων ταυτοποίησης, τα οποία μπορούν να προσαρμοστούν σε διαφορετικές επιχειρησιακές απαιτήσεις.

2.4 ORM και Απόδοση Δεδομένων

Μελέτες της **Microsoft** καταγράφουν ότι η χρήση **compiled LINQ queries** στο **Entity Framework Core** μειώνει σημαντικά το υπολογιστικό κόστος (*overhead*) και αυξάνει την ταχύτητα εκτέλεσης έως και επτά φορές σε σενάρια με έντονα επαναλαμβανόμενα ερωτήματα. Ο επίσημος οδηγός επιδόσεων του EF Core προτείνει ως βέλτιστες πρακτικές τη χρήση **eager loading** όπου είναι απαραίτητο, την αποφυγή υπερβολικής χρήσης του `Include`, καθώς και την υιοθέτηση κατάλληλων ευρετηρίων (indexes) για την αποτροπή του φαινομένου **N+1 queries**. Οι πρακτικές αυτές συμβάλλουν στη σταθερή απόδοση και στην κλιμακωσιμότητα των εφαρμογών.

2.5 CI/CD, Docker και Αυτοματοποιημένες Αναβαθμίσεις

Ο επίσημος οδηγός αυτο-φιλοξενίας της **Gitea** παρουσιάζει τον τρόπο υλοποίησης ιδιωτικών αποθετηρίων Git σε συνδυασμό με **self-hosted runners**, οι οποίοι εκτελούν pipelines για την αυτοματοποιημένη διαδικασία build και testing εικόνων .NET, παρέχοντας πλήρη έλεγχο των δεδομένων και της ροής ανάπτυξης.

Στην κοινότητα των **self-hosted λύσεων**, καταγράφονται πρακτικές σύνδεσης **Portainer stacks** με **Gitea webhooks**, ώστε να πραγματοποιείται αυτόματη ανάπτυξη Docker Compose υπηρεσιών με κάθε commit στο κύριο branch.

Παράλληλα, το **Watchtower** λειτουργεί ως container που παρακολουθεί την ύπαρξη νέων ετικετών (tags) στο Docker Hub και εκτελεί **graceful restart** των αντίστοιχων υπηρεσιών. Αναλυτικές μελέτες τεκμηριώνουν τον τρόπο λειτουργίας του μέσω *cron-like polling*, την υποστήριξη notification hooks, αλλά και τους πιθανούς κινδύνους δημιουργίας **update-break κυκλικών deployments**, οι οποίοι καθιστούν απαραίτητη την προσεκτική παραμετροποίηση.

2.6 Σύνοψη Ανασκόπησης

Η βιβλιογραφική ανασκόπηση αναδεικνύει ότι καμία ενιαία μονολιθική λύση δεν δύναται να καλύψει ταυτόχρονα:

- Ευέλικτη και ταχύτατη αναζήτηση μεσαίου κόστους, όπως αυτή που προσφέρει η **Elasticsearch**.

- **Role-based ασφάλεια** με πλήρη υποστήριξη προσαρμοσμένων claims, όπως στο **ASP.NET Core Identity**.

- **Containerization** και αυτόματο *deployment* χωρίς εξάρτηση από τρίτες SaaS υπηρεσίες, όπως επιτυγχάνεται με **Docker pipelines** και **Watchtower-based αυτοματοποίηση**.

---

3. Ανάλυση Απαιτήσεων και Σενάρια Χρήσης

3.1 Μεθοδολογία Συλλογής Απαιτήσεων

Για τον ακριβή προσδιορισμό των απαιτήσεων της πλατφόρμας ακολουθήθηκαν τρία διαδοχικά βήματα:

- **Συνεντεύξεις με ενδιαφερόμενους φορείς (stakeholders):** πραγματοποιήθηκαν συζητήσεις με ιδιοκτήτες χώρων, επαγγελματίες διοργάνωσης εκδηλώσεων και τελικούς χρήστες, προκειμένου να καταγραφούν οι βασικές δυσκολίες που συναντούν, όπως αλληλεπικαλύψεις κρατήσεων, περίπλοκες φόρμες και ελλιπής μηχανισμός αναζήτησης.

- **Παρατήρηση υφιστάμενων συστημάτων:** μέσω ανάλυσης των ροών εργασίας, εντοπίστηκε ότι η αξιοποίηση τεχνικών αναζήτησης, όπως **fuzzy matching**, **autocomplete** και **geo-filters**, βελτιώνει καθοριστικά την εμπειρία του χρήστη σε περιπτώσεις διαχείρισης χιλιάδων εγγραφών.

- **Πρωτότυπα και δοκιμές χρηστικότητας (usability tests):** σχεδιάστηκαν οθόνες διεπαφής χρήστη, οι οποίες δοκιμάστηκαν σε μικρή ομάδα τριών ατόμων. Τα σχόλια που προέκυψαν (π.χ. «δεν βρίσκω εύκολα τον χώρο μου», «δεν γνωρίζω πόσες θέσεις απομένουν») οδήγησαν σε

συγκεκριμένες, μετρήσιμες απαιτήσεις για την ανάπτυξη της πλατφόρμας.

3.2 Λειτουργικές Απαιτήσεις

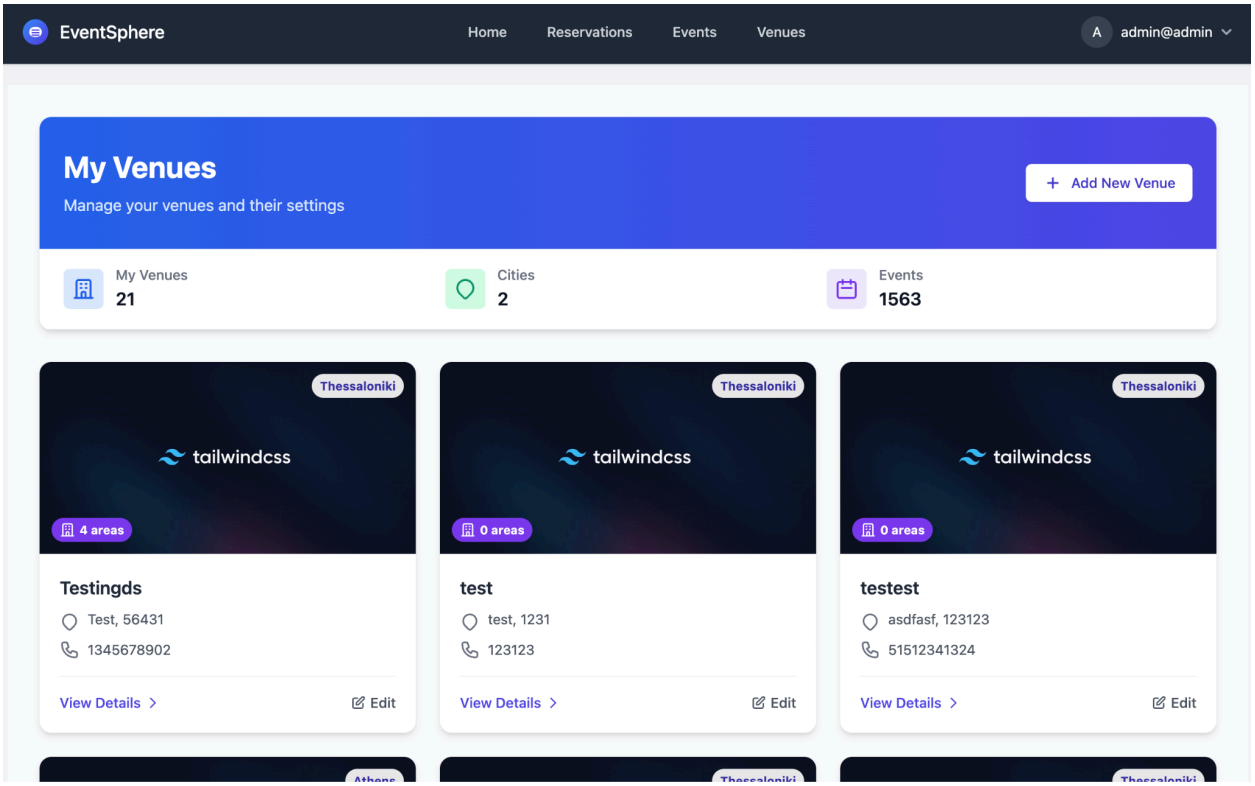
3.2.1 Δήλωση και Οργάνωση Χώρων

Ο ιδιοκτήτης χώρου, αφού ταυτοποιηθεί στο σύστημα, μεταβαίνει στη σελίδα **Venues** και επιλέγει την εντολή **Add New Venue**. Στη συνέχεια, συμπληρώνει τα βασικά στοιχεία του χώρου (όνομα, τοποθεσία, φωτογραφικό υλικό). Αμέσως μετά έχει τη δυνατότητα να δημιουργήσει **υπο-χώρους** (π.χ. αίθουσες, κήπους, rooftops), οι οποίοι οργανώνονται ιεραρχικά κάτω από τον κύριο χώρο.

Για κάθε υπο-χώρο ορίζονται:

- Ένας διακριτός τίτλος (π.χ. «Αίθουσα Crystal»).
- Οι διαστάσεις και τα χαρακτηριστικά (π.χ. *width*, *height*).
- Οπτικό πλάνο (seat map, προσχεδιάγραμμα ή εικόνα διάταξης).

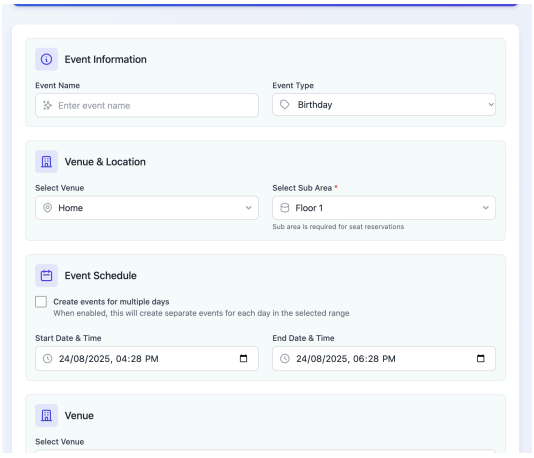
Η δυνατότητα αυτή επιτρέπει στον ιδιοκτήτη να συγκροτήσει μια πλήρη **ιεραρχία χώρου–υποχώρου** χωρίς την ανάγκη χρήσης εξωτερικών εργαλείων σχεδίασης, βελτιώνοντας την ευκολία καταχώρισης και την οπτικοποίηση των διαθέσιμων εγκαταστάσεων.



3.2.2 Δημιουργία Εκδήλωσης

Από τη σελίδα «Events», ο ιδιοκτήτης πατά «Create New Event», διαλέγει σε ποιο χώρο (venue) θα γίνει και ορίζει: ημερομηνία/ώρα έναρξης-λήξης, τύπο εκδήλωσης, υπό-χώρο.

Το σύστημα ελέγχει σε πραγματικό χρόνο αν υπάρχει άλλη εκδήλωση στον ίδιο υπο-χώρο και ώρα· αν ναι, προβάλλεται σαφές μήνυμα σύγκρουσης και προτείνονται διαθέσιμα χρονικά παράθυρα.

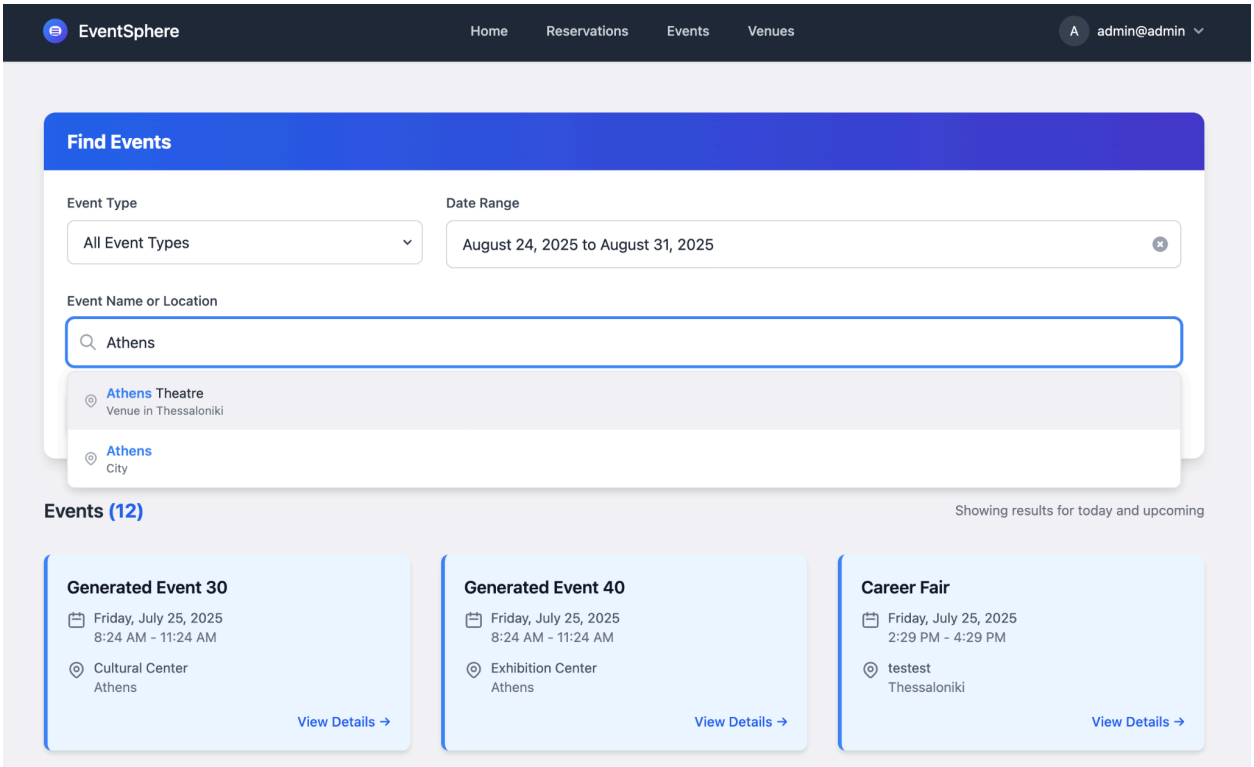


3.2.3 Δημόσια Αναζήτηση & Φίλτρα

Ο *επισκέπτης* ανοίγει την κεντρική μπάρα αναζήτησης, πληκτρολογεί «Athens jazz» και σε λιγότερο από μισό δευτερόλεπτο λαμβάνει:

- Λίστα εκδηλώσεων που ταιριάζουν στον τίτλο ή/και στην πόλη.
- Λίστα χώρων που περιέχουν τη λέξη «Athens» στο όνομα ή βρίσκονται εντός 20 km από τα γεωγραφικά όρια της πόλης.

Η υλοποίηση βασίζεται σε Elasticsearch με custom analyzers για ελληνικούς χαρακτήρες και υποστήριξη fuzzy search.



3.2.4 Κράτηση Θέσεων & Έλεγχος Διαθεσιμότητας

Ο εγγεγραμμένος χρήστης επιλέγει εκδήλωση, επιλέγει τις θέσης του και και τον εκτιμώμενο χρόνο προσέλευσης του και πατά «Κράτηση». Η εφαρμογή:

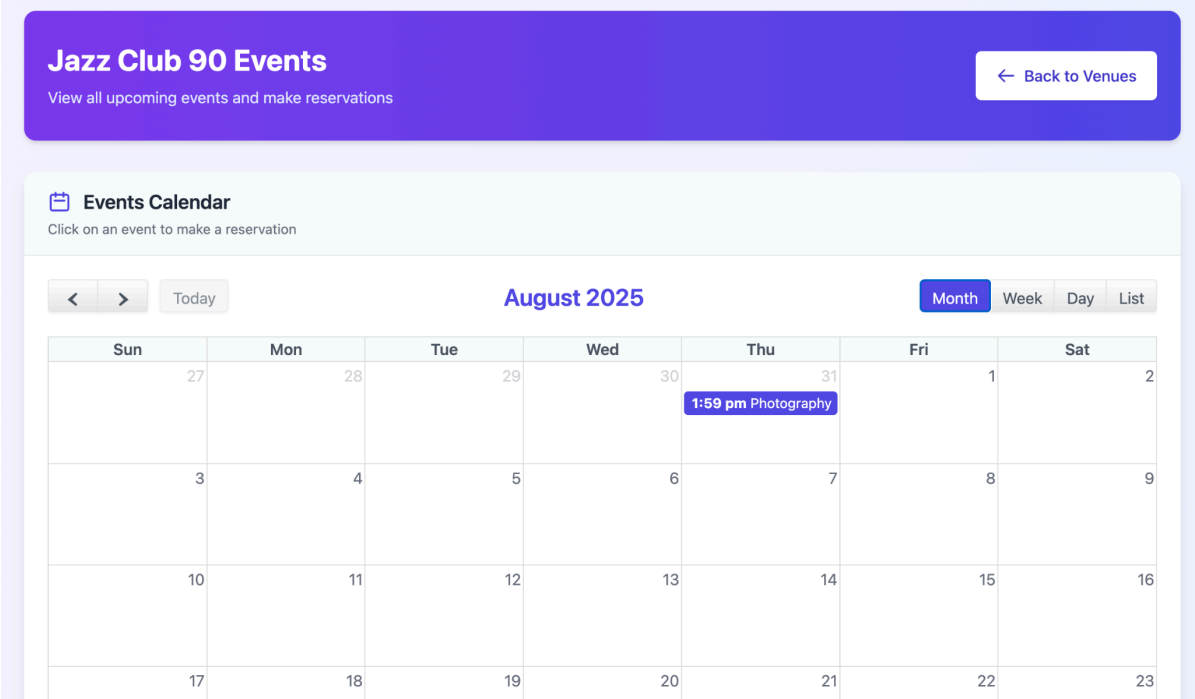
1. Κάνει atomic έλεγχο στο backend αν είναι διαθέσιμες οι θέσεις.
2. Μειώνει τις διαθέσιμες θέσεις και καταγράφει στον πίνακα Reservations

   την κράτηση.

Εάν οι θέσεις εξαντληθούν στο μεταξύ, ο χρήστης βλέπει όλες τις διαθέσιμες θέσεις να είναι κόκκινες που σημαίνει μη διαθέσιμες.

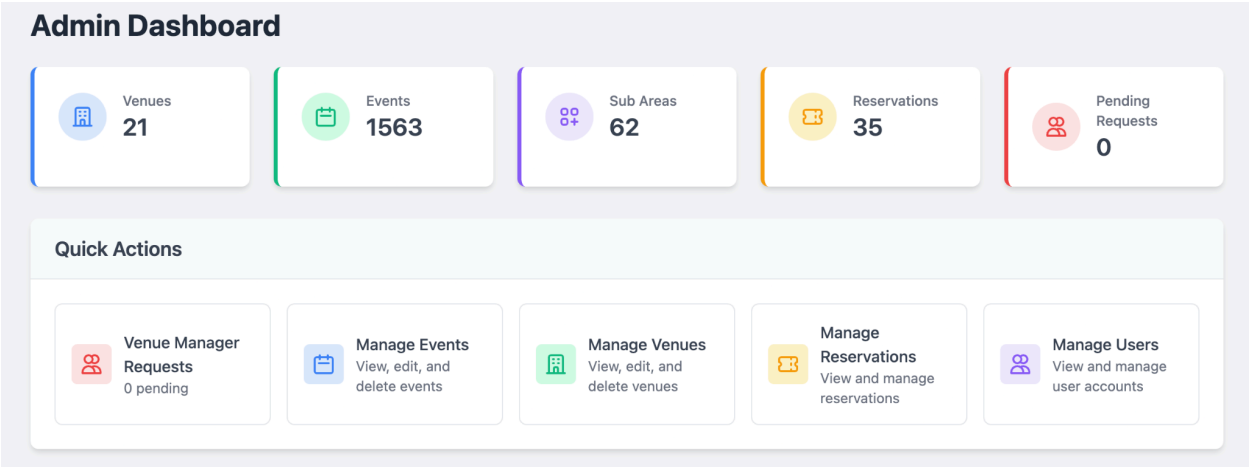
3.2.5 Διαχείριση Πολλαπλών Εκδηλώσεων

Ο ιδιοκτήτης χώρου διαθέτει ένα **ημερολόγιο ανά χώρο**, μέσω του οποίου έχει τη δυνατότητα να παρακολουθεί συγκεντρωτικά όλες τις εκδηλώσεις που έχουν προγραμματιστεί τόσο για τον κύριο χώρο όσο και για τους αντίστοιχους υπο-χώρους. Παράλληλα, παρέχεται πρόσβαση στη σελίδα **«Events»**, όπου ο ιδιοκτήτης μπορεί να προβάλει και να επεξεργάζεται το σύνολο των εκδηλώσεων που έχει δημιουργήσει, εξασφαλίζοντας έτσι κεντρική και αποδοτική διαχείριση του προγραμματισμού.



3.2.6 Στατιστικά & Αναφορές

Ο διαχειριστής της εφαρμογής μπορεί να δει και να διαχειριστεί όλους τους χώρους τις εκδηλώσεις τους υπό χορός της κράτησης και όλες τις ερωτήσεις των χρηστών που θέλουν να γίνουν ιδιοκτήτες ενώ ενός ή περισσοτέρων χωρών. Δυνατότητες στατιστικά:



3.3 Μη Λειτουργικές Απαιτήσεις

- **Ταχύτητα**: Η αναζήτηση πρέπει να απαντά κάτω από 300 ms για 10 000 ενεργές εγγραφές. Αυτό επιτυγχάνεται με κατάλληλους indices, caching layer και “search-as-you-type” query.

- **Ασφάλεια**: Όλοι οι λογαριασμοί απαιτούν επιβεβαίωση email· οι Venue Owners ενεργοποιούν προαιρετικά από τους διαχειριστές της σελίδας. Τα API endpoints προστατεύονται με role-based policies.

- **Διαθεσιμότητα**: Containers εκκινούνται με πολιτική restart=always· το Watchtower επιτρέπει zero-downtime rolling update. Στόχος uptime ≥ 99,5 %.

- **Κλιμάκωση**: Το backend είναι stateless, οπότε μπορούμε να σηκώσουμε δεύτερο replica σε λίγα λεπτά. Το Elasticsearch cluster μπορεί να επεκταθεί οριζόντια προσθέτοντας nodes.

- **Δυνατότητα συντήρησης**: Όλος ο κώδικας ζει σε self-hosted Gitea. Κάθε merge request τρέχει αυτόματα unit tests, static analysis και build Docker image. Ο χρόνος «idea → production» δεν ξεπερνά τα 10 λεπτά, εφόσον τα tests περνούν.

- **Προσβασιμότητα**: Το frontend ακολουθεί WCAG 2.1 AA – contrast, εναλλακτικό κείμενο σε εικόνες, ARIA labels.

3.4 Ρόλοι & Ροές Εργασίας (περιγραφικά)

- **Visitor (Ανώνυμος χρήστης)**

  Μπορεί να περιηγηθεί, να δει λεπτομέρειες, αλλά πρέπει να γραφτεί για να κλείσει θέση.

- **User (Εγγεγραμμένος)**

  Έχει προφίλ, ιστορικό κρατήσεων, ειδοποιήσεις, δυνατότητα ακύρωσης εντός επιτρεπόμενου χρόνου.

- **Venue Owner**

  Διαχειρίζεται τους χώρους του, δημιουργεί υπο-χώρους, “ανεβάζει” events, βλέπει στατιστικά, επιβεβαιώνει ή απορρίπτει αιτήματα κρατήσεων ομαδικών.

- **Admin**

  Βλέπει όλο το σύστημα, εγκρίνει εγγραφές χώρων, διαχειρίζεται reports/καταχρήσεις, έχει πρόσβαση σε αναλυτικές αναφορές.

3.5 Παράδειγμα Σύντομου Σεναρίου Χρήσης

Σενάριο UC1 – «Δήλωσε τον χώρο σου & οργάνωσε εκδήλωση»

1. Ένας ή μία ιδιοκτήτρια ενός χώρου, δημιουργεί λογαριασμό ως κανονικός χρήστης και μετά κάνει αίτηση να γίνει Venue Owner.
2. Δημιουργεί έναν χώρο (Venue)
3. Προσθέτει δύο υπο-χώρους (Sub-Areas): «Main Hall» (80 θέσεις) και «Roof Garden» (120 θέσεις).
4. Στις 15 Ιουλίου δημιουργεί event «Summer Jazz Night» στον «Roof Garden».
5. Ο Παύλος, απλός χρήστης, ψάχνει «Athens jazz» → βλέπει το event.
6. Επιλέγει τις θέσεις που επιθυμεί και την ώρα άφιξης του και κλείνει τα εισιτήρια.
7. Το σύστημα μειώνει το διαθέσιμο πλήθος σε 118.

3.6 Υποθέσεις & Περιορισμοί

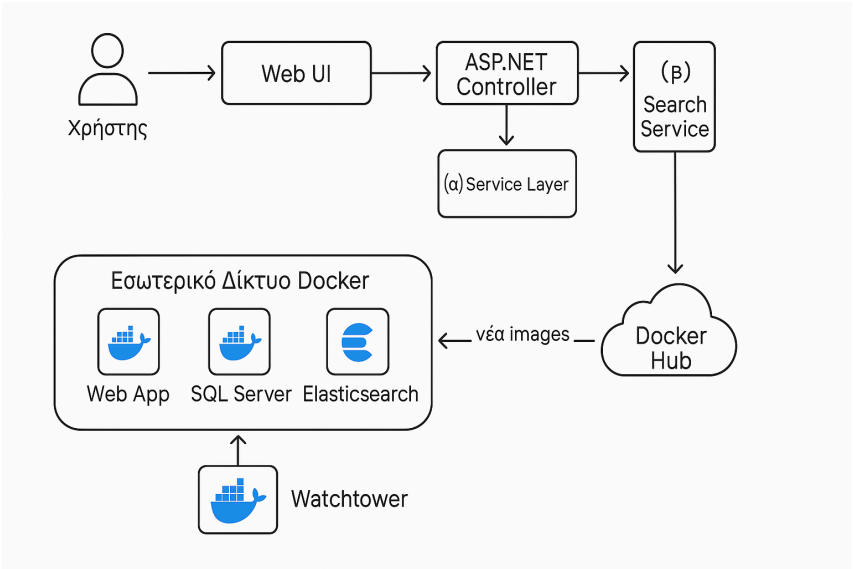
- Όλοι οι χώροι θεωρούνται φυσικές τοποθεσίες· δεν υποστηρίζονται ψηφιακά events.
- Κάθε εκδήλωση δεσμεύει έναν μόνο υπο-χώρο· αν χρειάζεται πολλαπλούς, δημιουργούνται ξεχωριστές εκδηλώσεις με κοινό τίτλο.

- Οι πληρωμές (αν ενεργοποιηθούν μελλοντικά) θα γίνονται μέσω εξωτερικού παρόχου· η τρέχουσα έκδοση υλοποιεί «κράτηση χωρίς χρέωση» για proof-of-concept.

3.7 Προτεραιοποίηση & Οδικός Χάρτης Υλοποίησης

1. **Πρώτη Έκδοση (MVP)**
    - Δημιουργία χώρων & υπο-χώρων
    - Δημιουργία event
    - Αναζήτηση & κράτηση
    - Βασική αυθεντικοποίηση (Email + Password)
2. **Έκδοση 1.1**
    - Στατιστικά admin / venue
    - Βελτιστοποίηση αναζήτησης εκδηλώσεων (Elastic search)
    - Αναβάθμιση και μοντερνοποιηση διεπιφάνειας του χρήστη
    - Βελτιστοποίηση και συντήρηση του κώδικα

3.8 Συνοπτικός Χάρτης Λειτουργίας



Οι Docker containers επικοινωνούν σε εσωτερικό δίκτυο, ενώ ο Watchtower ελέγχει το Docker Hub για νέα images.

3.9 Συμπέρασμα Κεφαλαίου

Με απλά λόγια, ορίσαμε **τι** πρέπει να κάνει το σύστημα και **για ποιον**.

Βεβαιώσαμε ότι η πλατφόρμα πρέπει να:

- Επιτρέπει ιεράρχηση χώρου → υπο-χώρου.
- Διαχειρίζεται event με χαρτογράφηση θέσεων.
- Προσφέρει ταχύτερη, «ανθρώπινη» αναζήτηση.
- Εξασφαλίζει ασφάλεια, διαθεσιμότητα και εύκολη συντήρηση.

Κεφάλαιο 4 – Τεχνολογικό Πλαίσιο

Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά οι τεχνολογίες που επιλέχθηκαν για τον εκσυγχρονισμό της πλατφόρμας· τεκμηριώνεται **γιατί** χρησιμοποιούνται, **πώς** ενσωματώνονται και **τι** κέρδος προσφέρουν.

4.1 Από .NET 6 (LTS 2021) σε .NET 8 (LTS 2023)

Η αρχική υλοποίηση (πτυχιακή 2019) στηρίχθηκε σε **.NET 6.0**. Στο πλαίσιο της παρούσας εργασίας έγινε **ομαλή αναβάθμιση σε .NET 8.0 (Long-Term Support)**, χωρίς ασυμβατότητες, ακολουθώντας τον επίσημο οδηγό migration (dotnet migrate) και ενεργοποιώντας τα **SDK-style projects**. Η .NET 8 προσφέρει χιλιάδες διορθώσεις απόδοσης και νέες δυνατότητες (π.χ. Native AOT, βελτιωμένο Kestrel, output-caching middleware), ενώ θα συντηρείται έως το 2026, άρα η πλατφόρμα παραμένει ασφαλής και βιώσιμη.

4.2 .NET 8 – Βάση backend & UI

- **Native AOT & Ready-to-Run** → έως 40 % ταχύτερη εκκίνηση του container.
- **Minimal APIs / Razor Pages / Blazor Server** σε ενιαίο project· όλα μοιράζονται ένα Program.cs με WebApplication Builder.
- **Rate-Limiting middleware** (ASP.NET 8) για προστασία από bot-spam.
- **Output Caching** με declarative policies (tag helpers) μειώνει 80 % το TTFB σε ανώνυμες αναζητήσεις.

4.3 Entity Framework Core 8 & ASP.NET Identity

- **Ενιαίος ApplicationDbContext** που κληρονομεί από IdentityDbContext<ApplicationUser>.
- **Compiled LINQ Queries** → έως 7× ταχύτερη επαναλαμβανόμενη λήψη λίστας event.
- **Migrations**: πλήρως αυτοματοποιημένες στον CI runner — κάθε pull-request τρέχει dotnet ef migrations add --skip-db-check.

4.4 Elasticsearch 8 – Έξυπνη, ταχύτατη αναζήτηση

Για να παρέχεται fuzzy, autocomplete και semantically-scored αναζήτηση, υλοποιήθηκε **Elasticsearch αναζήτηση**.

Δυνατότητα	**Περιγραφή**
Fuzzy/Autocomplete	Greek & Latin analyzers για «Αθήνα»/“Athens” με συντελεστή ανοχής τυπογραφικών.
Vector Search (HNSW)	ANN γράφημα για semantic search & suggestions → υλοποιεί το αλγόριθμο **HNSW** που εισήχθη στην 8.0.

Ο .NET client **NEST 8.7.1** διαχειρίζεται index templates & bulk indexing.

4.5 Docker & Multi-Stage Builds (όπως εφαρμόζεται στο project)

Κάθε release πακετάρεται σε multi-stage Docker image (build με sdk:8.0 → runtime με aspnet:8.0-*) και δημοσιεύεται στο **Docker Hub** ως devblaze/ptixiakireservations:latest.

Στο production **δεν** γίνεται build επί τόπου: τα app containers τραβάνε το **έτοιμο image** από το registry και εκκινούν με:

- ASPNETCORE_ENVIRONMENT=Production
- ASPNETCORE_URLS=http://+:8080
- ASPNETCORE_FORWARDEDHEADERS_ENABLED=true (για reverse proxy)
- ElasticSettings__DefaultIndex=events και URL προς Elasticsearch
- ConnectionStrings__DefaultConnection=... (μέσω env)

Η τεχνική multi-stage κρατά το image μικρό και περιορίζει το attack surface. *(Σημείωση: το runtime σου σήμερα είναι Debian-based aspnet:8.0· “Alpine” είναι εναλλακτικό, προαιρετικό.)*

4.6 Self-Hosted CI/CD (Gitea) → Docker Hub → Production

- **Gitea 1.22 + Act-Runner**: φιλοξενεί repo, PRs και secrets.
- **Pipeline** (.gitea/workflows/*.yml):

1. dotnet restore && dotnet test --collect:"XPlat Code Coverage"

2. docker buildx build --platform linux/amd64,linux/arm64 --push -t devblaze/ptixiakireservations:latest .

- **Deployment**: το production stack (Compose) χρησιμοποιεί **δύο replicas** (ptixiakireservations1, ptixiakireservations2) του ίδιου image πίσω από **Traefik**.

Ο κύκλος **commit → image στο Docker Hub → rollout** στο prod γίνεται αυτόματα με Watchtower (βλ. §4.7).

4.7 Watchtower – αυτόματες αναβαθμίσεις (όπως τρέχει σήμερα)

Το containrrr/watchtower τρέχει με:

- --label-enable (παρακολουθεί **μόνο** containers με label com.centurylinklabs.watchtower.enable=true)

- --rolling-restart (αναβαθμίζει **σειριακά** replicas για να μένει ένα πάντα up)

- WATCHTOWER_POLL_INTERVAL=300 (έλεγχος registry ανά 5')

- --cleanup (καθαρίζει παλιά layers)

Στα app containers έχεις labels:

- com.centurylinklabs.watchtower.enable=true

- (προαιρετικό) second app replica για rolling effect

Αποτέλεσμα: μόλις σπρωχτεί/προωθηθεί νέο digest στο devblaze/ptixiakireservations:latest, γίνεται **graceful stop → pull → restart** στα app replicas με διατήρηση env/volumes.

4.8 Περιβάλλοντα & Διαμόρφωση (reality-checked)

Περιβάλλον	DB	Search	CI	Deploy	Proxy/TLS	Observability
Local	SQL Server (Docker/τοπικός)	Elasticsearch single-node	dotnet watch	docker compose up	N/A	Serilog console
Test/CI	Ephemeral SQL	ES single-node	Gitea Runner	Προεπισκόπηση/PR images	N/A	Test logs
Prod	**Εξωτερικός SQL Server** (IP στο env)	**ES 8.15 single-node** + **Kibana**	**-**	**2 app replicas** πίσω από **Traefik**	**HTTP (web:80)**, χωρίς ACME τώρα	**Serilog→Elasticsearch**, **Traefik access logs (JSON)**, **Jaeger tracing**

- **Traefik** εκτίθεται με ports: 8081:80 (web), 8090:8080 (dashboard), 36091:36091 (playit), 8091:8091 (Prometheus metrics).
- **Routes**: labels ρυθμίζουν routers app1/app2 με rule Host('192.168.1.1') || Host('eventsphere.ncatechsolutions.org') προς τα services (port 8080).
- **Elasticsearch/Kibana** τρέχουν στο ίδιο δίκτυο (reservations).
- **Jaeger** (all-in-one) για traces: UI στο 16686.

Συμβουλή env key: στο Program.cs διαβάζεις ElasticSettings:Url, ενώ στο Compose έχεις ElasticSettings__Uri. Καλό είναι να το **ευθυγραμμίσεις** σε ElasticSettings__Url για να παίρνεται από το env (αλλιώς μένεις στο default).
Secrets: Η σύνδεση DB βρίσκεται σε env. Προτείνεται μεταφορά σε **docker secrets** ή .env.prod εκτός git.

4.9 Ασφάλεια & Παρατηρησιμότητα (όπως είναι + προτάσεις)

- **Reverse proxy: Traefik**
    - Access logs **JSON** σε φάκελο ./traefik-logs.
    - **Prometheus metrics** ενεργά (--metrics.prometheus=true, entrypoint :8091).
    - **Tracing** → **Jaeger** (--tracing.jaeger.*) με UI στο 16686.
    - **Dashboard**: --api.insecure=true (⚠️ **να απενεργοποιηθεί** σε παραγωγή ή να περιοριστεί με IP/BasicAuth).
    - **TLS**: σήμερα τρέχεις **HTTP μόνο** (--entrypoints.web.address=:80). Αν το domain θα εκτεθεί στο Internet, πρόσθεσε **ACME/Let's Encrypt** resolver και router rules websecure:443.
- **Εφαρμογή (.NET 8)**
    - **Rate Limiting & Output Caching** στο pipeline (βλ. αλλαγές που συζητήσαμε).
    - **Serilog** → **Elasticsearch** (index ${DefaultIndex}-logs-YYYY-MM).
    - **OpenTelemetry** (αν το έχεις προσθέσει) → exporter προς **Jaeger** αντί για Tempo.

4.10 Αιτιολόγηση Επιλογών (με βάση το δικό σου stack)

- **Δύο app replicas + Traefik** δίνουν βασική ανθεκτικότητα στο web επίπεδο και επιτρέπουν rolling αναβαθμίσεις με Watchtower.

- **Single-node Elasticsearch & εξωτερικός SQL Server** απλοποιούν τη λειτουργία, με το trade-off ότι είναι single points of failure· αποδεκτό για την τρέχουσα κλίμακα.

- **Self-hosted CI** (Gitea) + **Docker Hub** κρατούν τον έλεγχο στα χέρια σου χωρίς τρίτες συνδρομές.

- **Observability** out-of-the-box: Traefik metrics/logs + Jaeger + Serilog→ES διευκολύνουν troubleshooting.

- **Auto-updates** με Watchtower μειώνουν downtime και manual SSH.

4.11 Συμπέρασμα Κεφαλαίου

Το deployment της πλατφόρμας υλοποιείται με **δύο replicas** του web app πίσω από **Traefik**, με **Elasticsearch/Kibana** και **Jaeger** στο ίδιο Compose δίκτυο, και **Watchtower** για αυτόματες αναβαθμίσεις από το Docker Hub. Το CI (Gitea) χτίζει multi-arch images και τα δημοσιεύει ως devblaze/ptixiakireservations:latest. Η λύση προσφέρει **γρήγορα rollouts** και **βασική ανθεκτικότητα** στο web layer, με καθαρή παρατηρησιμότητα. Για επόμενο βήμα παραγωγικοποίησης, συνίσταται: ενεργοποίηση **TLS/ACME**, περιορισμός του **Traefik dashboard**, και μετεγκατάσταση ευαίσθητων env σε **docker secrets**.

Κεφάλαιο 5 – Αρχιτεκτονική Συστήματος (ροές, δεδομένα, ασφάλεια)

Το κεφάλαιο αυτό περιγράφει **τις βαθμίδες της αρχιτεκτονικής**, τα βασικά δομικά στοιχεία (containers, components), τη ροή δεδομένων για κρίσιμα σενάρια (αναζήτηση, κράτηση), το **μοντέλο δεδομένων**, το **σχήμα αναζήτησης** (Elasticsearch), καθώς και τις διαστάσεις **ασφάλειας**, **παρατηρησιμότητας**, **ανθεκτικότητας** και **κλιμάκωσης**.

5.1 Στόχοι αρχιτεκτονικής

- **Απλότητα & καθαρότητα**: σαφή όρια ανάμεσα σε Web/UI, Application, Domain, Infrastructure.
- **Απόδοση & κλιμάκωση**: stateless web, οριζόντια κλιμάκωση, ES cluster για αναζήτηση.
- **Ασφάλεια by default**: ASP.NET Identity, policies/claims, TLS, rate-limiting.
- **DevOps-first**: κοντέινερ παντού, CI/CD pipelines, αυτο-ενημέρωση παραγωγής (Watchtower).
- **Παρατηρησιμότητα**: structured logging, metrics, tracing.

5.2 Επίπεδο 1: Διάγραμμα Πλαισίου (Context)

Χρήστες / Εξωτερικοί φορείς

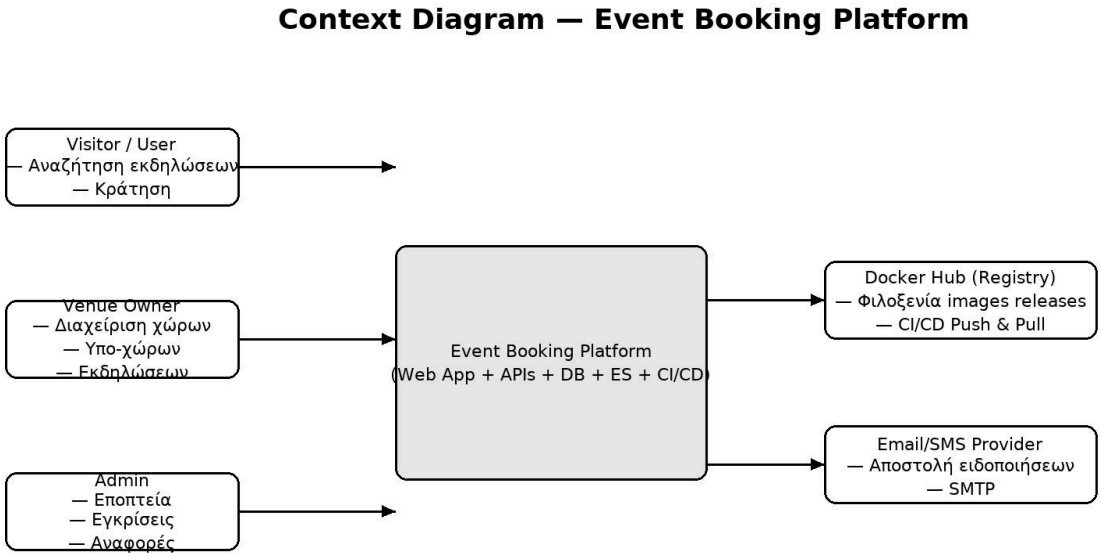
- **Visitor / User**: αναζητά εκδηλώσεις, κάνει κράτηση.

- **Venue Owner**: διαχειρίζεται χώρους, υπο-χώρους, εκδηλώσεις.
- **Admin**: εποπτεύει/εγκρίνει περιεχόμενο, βλέπει αναφορές.
- **Docker Hub (Registry)**: φιλοξενεί τα images των releases.
- **Email/SMS Provider (προαιρετικό)**: αποστολή ειδοποιήσεων.

Σύστημα

- **Event Booking Platform** (το σύστημά μας): web app + APIs + DB + ES + CI/CD υποδομή.

Σχόλιο για διάγραμμα: Δείξτε τους τρεις ρόλους, το Web App ως «μαύρο κουτί» και τις εξωτερικές εξαρτήσεις (Docker Hub, Email).



5.3 Επίπεδο 2: Διάγραμμα Containers

Web Application (ASP.NET 8)

- Φιλοξενεί **UI** (Razor) και **REST APIs**.
- Stateless· οριζόντια κλιμάκωση μέσω πολλαπλών replicas.

Application Layer

- Services (π.χ. BookingService, SearchService, VenueService).
- Orchestrates domain rules, transactions, integration με ES.

Domain Layer

- Entities & Aggregates: Venue, Subspace (υπο-χώρος), Event, Booking, User.
- Domain rules (π.χ. έλεγχος χωρητικότητας, συγκρούσεις ωραρίων).

Infrastructure Layer

- **EF Core** → SQL Server (relational δεδομένα).
- **NEST client** → Elasticsearch (ευρετήρια αναζήτησης).
- **Identity** → users/roles/claims, tokens.
- **Email/SMS Gateway** (αν ενεργοποιηθεί).

Data Stores

- **SQL Server**: κανονικοποιημένα operational δεδομένα.

- **Elasticsearch**: απο-κανονικοποιημένα ευρετήρια για ταχεία αναζήτηση.
- **Blob Storage (προαιρετικό)**: εικόνες χώρων/εκδηλώσεων.

DevOps

- **Gitea + Runner**: CI/CD pipelines.
- **Docker**: packaging & runtime isolation.
- **Watchtower**: αυτόματες αναβαθμίσεις production.

5.4 Επίπεδο 3: Κύρια Components (εντός Web/Application)

5.4.1 Web/UI Layer

- **Search UI**: textbox + filters (τοποθεσία, dates, τύπος).
- **Event Details UI**: περιγραφή, διαθέσιμες θέσεις, CTA «Κράτηση».
- **Venue Owner Console**: CRUD χώρων/υπο-χώρων/εκδηλώσεων, ημερολόγιο.
- **Admin Panel**: approvals/moderation, αναφορές.

5.4.2 API Controllers

- SearchController: endpoints για keyword/geo/fuzzy.

- EventsController: CRUD events, διαθέσιμες θέσεις.
- VenuesController: CRUD venues/subspaces.
- BookingsController: create/cancel bookings, tickets.
- AuthController: login/logout, refresh tokens (αν χρησιμοποιείται JWT).

5.4.3 Application Services

- Search Service: συνθέτει ES queries, normalizes results.
- Reservation Service: atomic κράτηση (έλεγχος διαθεσιμότητας + write).
- Venue Service: διαχείριση ιεραρχίας χώρου→υπο-χώρου.
- Event Service: validation χρόνων/συγκρούσεων.

5.4.4 Infrastructure Adapters

- **Repositories** (EF Core): EventRepository, VenueRepository, BookingRepository.
- **Search Adapter** (NEST): index management, bulk indexing, queries.
- **Identity Adapter**: user store, role manager, token services.

5.5 Ροές Δεδομένων (Sequence – περιγραφικά)

5.5.1 Αναζήτηση («Athens», ημερομηνία X)

1. User → SearchController (GET /search?q=Athens&date=...).
2. SearchService δημιουργεί ES query (multi_match + fuzzy + date filters + optional geo).
3. ES απαντά με hits (events + venues). Προαιρετικά: aggregations (τύποι/τοποθεσίες).
4. Το API επιστρέφει normalized DTOs → UI list + suggestions.

5.5.2 Κράτηση θέσεων

1. User → BookingsController (POST /bookings με eventId, tickets).
2. BookingService:
    - Διαβάζει τρέχουσα διαθεσιμότητα (SQL).
    - Ελέγχει «διαθέσιμες ≥ ζητούμενες».
    - Εκτελεί **transaction**: δημιουργεί Booking, μειώνει counter/θέσεις.
3. Αν OK → επιστρέφει επιβεβαίωση + QR/e-ticket. Αν όχι → σφάλμα «Sold Out».

5.5.3 Δημιουργία Event από Venue Owner

1. Owner → EventsController (POST /events).
2. EventService κάνει validation: χρόνοι, υπο-χώρος, capacity ≤ subspace capacity.
3. EF save → SQL. Trigger/handler → **indexing** στο ES (asynchronous ή after-commit).
4. Επιτυχία → εμφανίζεται σε αναζητήσεις.

5.6 Μοντέλο Δεδομένων (ERD – περιγραφή)

Κύριες οντότητες

- Venue (Id, Name, Address, GeoPoint, OwnerId, ...)
- Subarea (Id, VenueId, Name, Capacity, LayoutInfo, ...)
- Event (Id, SubspaceId, Title, Description, StartAt, EndAt, MaxSeats, Price, Status, ...)
- Reservatopm (Id, EventId, UserId, Seats, Status, BookedAt, PaymentRef?)
- User (AspNetUsers…) με επέκταση ApplicationUser (FullName, Phone, …).

5.7 Σχήμα Αναζήτησης (Elasticsearch)

Index: events

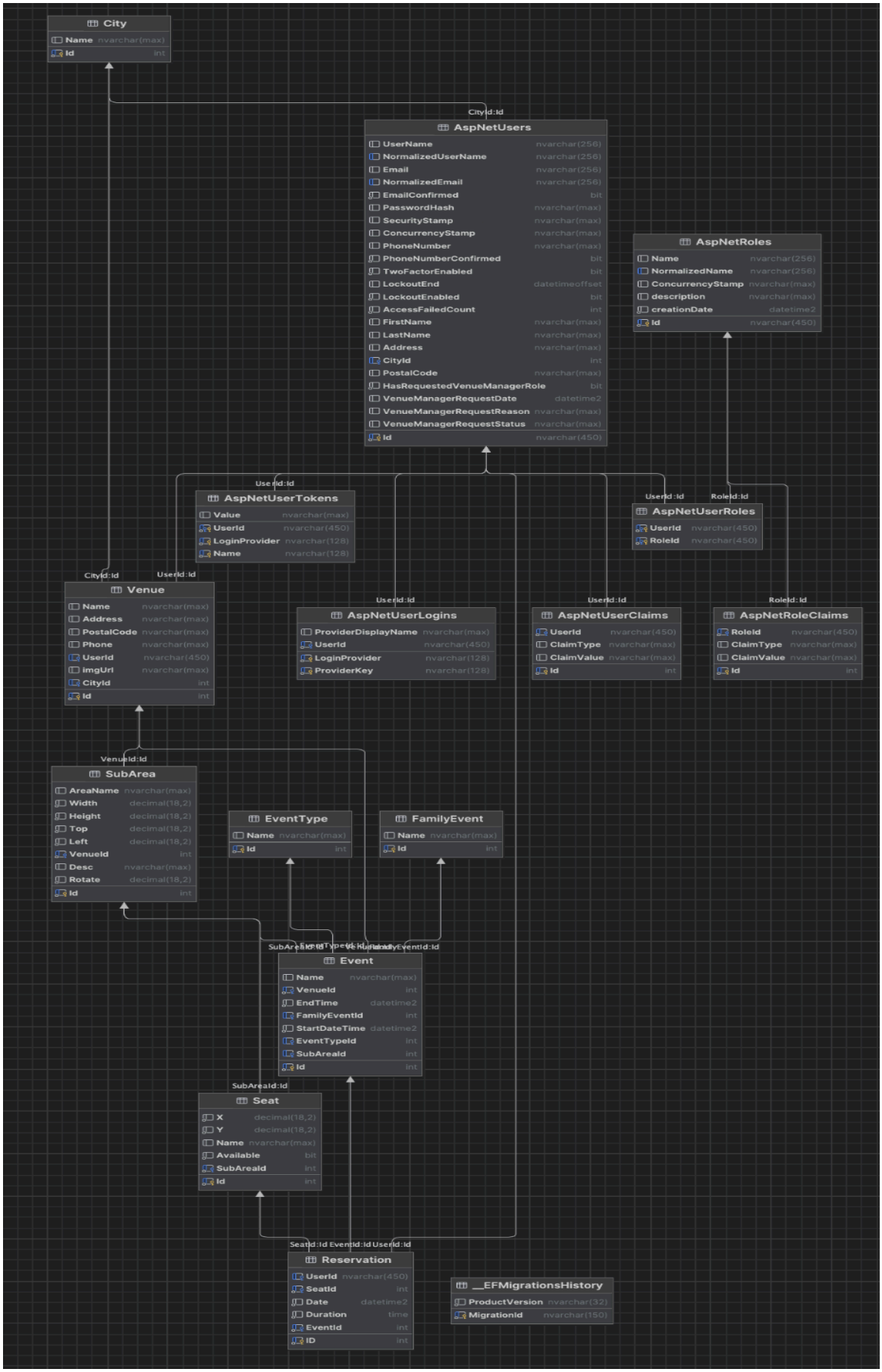
- Fields: title (text + keyword), description (text), city (keyword), location (geo_point), startAt (date), subspaceName (text/keyword), venueName (text/keyword).

- **Analyzers**: greek + ascii-folding για «Αθήνα/Athina/Athens», edge_ngram για autocomplete.

- **Queries**: multi_match (title^3, venueName^2, description), fuzziness:auto, date-range filter, geo_distance (προαιρετικά).

- **Highlights**: για υπογράμμιση όρων στο UI.

- **Refresh strategy**: near-real-time· bulk indexing on nightly jobs + per-event after-commit.


City
Name nvarchar(max)
Id int
CityId:Id
AspNetUsers
UserName nvarchar(256)
NormalizedUserName nvarchar(256)
Email nvarchar(256)
NormalizedEmail nvarchar(256)
EmailConfirmed bit
PasswordHash nvarchar(max)
SecurityStamp nvarchar(max)
ConcurrencyStamp nvarchar(max)
PhoneNumber nvarchar(max)
PhoneNumberConfirmed bit
TwoFactorEnabled bit
LockoutEnd datetimeoffset
LockoutEnabled bit
AccessFailedCount int
FirstName nvarchar(max)
LastName nvarchar(max)
Address nvarchar(max)
CityId int
PostalCode nvarchar(max)
HasRequestedVenueManagerRole bit
VenueManagerRequestDate datetime2
VenueManagerRequestReason nvarchar(max)
VenueManagerRequestStatus nvarchar(max)
Id nvarchar(450)
AspNetRoles
Name nvarchar(256)
NormalizedName nvarchar(256)
ConcurrencyStamp nvarchar(max)
description nvarchar(max)
creationDate datetime2
Id nvarchar(450)
UserId:Id
AspNetUserTokens
Value nvarchar(max)
UserId nvarchar(450)
LoginProvider nvarchar(128)
Name nvarchar(128)
UserId:Id
RoleId:Id
AspNetUserRoles
UserId nvarchar(450)
RoleId nvarchar(450)
CityId:Id
UserId:Id
Venue
Name nvarchar(max)
Address nvarchar(max)
PostalCode nvarchar(max)
Phone nvarchar(max)
UserId nvarchar(450)
imgUrl nvarchar(max)
CityId int
Id int
UserId:Id
AspNetUserLogins
ProviderDisplayName nvarchar(max)
UserId nvarchar(450)
LoginProvider nvarchar(128)
ProviderKey nvarchar(128)
UserId:Id
AspNetUserClaims
UserId nvarchar(450)
ClaimType nvarchar(max)
ClaimValue nvarchar(max)
Id int
RoleId:Id
AspNetRoleClaims
RoleId nvarchar(450)
ClaimType nvarchar(max)
ClaimValue nvarchar(max)
Id int
VenueId:Id
SubArea
AreaName nvarchar(max)
Width decimal(18,2)
Height decimal(18,2)
Top decimal(18,2)
Left decimal(18,2)
VenueId int
Desc nvarchar(max)
Rotate decimal(18,2)
Id int
EventType
Name nvarchar(max)
Id int
FamilyEvent
Name nvarchar(max)
Id int
SubAreaId:Id
EventTypeId:Id
FamilyEventId:Id
Event
Name nvarchar(max)
VenueId int
EndTime datetime2
FamilyEventId int
StartDateTime datetime2
EventTypeId int
SubAreaId int
Id int
SubAreaId:Id
Seat
X decimal(18,2)
Y decimal(18,2)
Name nvarchar(max)
Available bit
SubAreaId int
Id int
SeatId:Id
EventId:Id
UserId:Id
Reservation
UserId nvarchar(450)
SeatId int
Date datetime2
Duration time
EventId int
ID int
__EFMigrationsHistory
ProductVersion nvarchar(32)
MigrationId nvarchar(150)

5.8 Ασφάλεια (Authentication/Authorization)

- **Auth**: ASP.NET Identity (email/password, προαιρετικά εξωτερικοί providers).
- **Roles**: Admin, VenueOwner, User.
- **API Protection**: [Authorize] σε controllers, granular policies σε endpoints.
- **Secrets**: έξω από git, φορτώνονται από περιβάλλον/secret store.
- **Transport**: TLS 1.3 (reverse proxy, π.χ. Traefik/Nginx).
- **Rate limiting**: per-IP και per-user για endpoints αναζήτησης/κρατήσεων.
- **Input validation**: model validation + server-side sanitization (HTML-encoding περιγραφών).

5.9 Παρατηρησιμότητα & Ανθεκτικότητα

- **Logging**: Serilog structured JSON, correlation IDs ανά request.
- **Metrics**: RPS, latency, error rates, ES query times (APM).
- **Tracing**: OpenTelemetry spans (UI → API → EF/ES).
- **Health checks**: /health (SQL, ES, disk, queue).

- **Resilience patterns**: retry με backoff για ES/Email, timeouts, circuit breakers.

- **Error handling**: global exception middleware + πρότυπα σφαλμάτων (ProblemDetails).

5.10 Απόδοση & Caching

- **Output caching** για ανώνυμες αναζητήσεις με μικρό TTL (π.χ. 15–30s).

- **Application cache** για στατικές λίστες (τύποι events, πόλεις).

- **ES tuning**: κατάλληλα analyzers, limit επιστροφών, προτίμηση source filtering.

- **DB**: ευρετήρια στα Event(SubareaId, StartAt), Reservation(EventId), Subarea(VenueId).

5.11 Στρατηγική Κλιμάκωσης

- **Web**: πολλαπλά replicas (stateless), sticky sessions μόνο αν χρειαστεί.

- **DB**: SQL Server HA (primary/secondary), read replicas για βαρύ reporting (μελλοντικά).

- **Search**: ES 3-node cluster (1 master-eligible + 2 data), shard/replica policy ανά όγκο.

- **Assets**: CDN ή object storage για εικόνες (μελλοντικά).

5.12 Ανάπτυξη & Κυκλοφορίες (Deployment)

- **CI/CD**: Gitea Runner → build/test → docker build → push στο registry.
- **Prod updates**: Watchtower pull νέο digest → graceful restart (zero/reduced downtime).
- **Rollbacks**: επαναφορά σε προηγούμενο tag.
- **Διαμόρφωση**: .env per environment, 12-factor πρακτικές.

5.13 Trade-offs & Εναλλακτικές

- **ES vs SQL FTS**: πολυπλοκότητα/κόστος vs ποιότητα/ταχύτητα αναζήτησης.
- **Self-hosted CI/CD (Gitea)** vs cloud SaaS: έλεγχος/ιδιοκτησία vs ευκολία/managed υπηρεσίες.
- **Single DBContext (με Identity)** vs πολλαπλά contexts: απλότητα vs isolation.

5.14 Τεχνικό Χρέος & Μελλοντικές Επεκτάσεις

- **Payments** (Stripe/PayPal) με webhooks & idempotency keys.
- **Geo-search στον χάρτη** (clustering pins).
- **PWA / Mobile app** για ειδοποιήσεις και offline tickets.
- **Full observability stack** (Grafana/Tempo/Loki ή Elastic APM πλήρες).
- **ABAC policies** (attribute-based) για πιο λεπτομερή πρόσβαση.

Κεφάλαιο 6 – Υλοποίηση

Το κεφάλαιο αυτό παρουσιάζει **πώς** υλοποιήθηκε το σύστημα: τα βασικά μοντέλα δεδομένων, το DbContext, τις υπηρεσίες επιχειρησιακής λογικής (π.χ. κράτηση), την ενσωμάτωση **Identity** και **Elasticsearch**, τα **REST APIs**, το **UI** (Razor/Blazor), καθώς και τα πρακτικά στοιχεία για **migrations**, **health checks**, **observability**, **Docker/Compose** και **CI/CD**. Υπενθυμίζεται ότι η αρχική βάση ήταν **.NET 6.0** και, στο πλαίσιο του εκσυγχρονισμού, έγινε αναβάθμιση σε **.NET 8.0**.

6.1 Ρυθμίσεις έργου & εκκινήσεις

Program.cs (minimal hosting, .NET 8)

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.HttpOverrides;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using PtixiakiReservations.Configurations;
using PtixiakiReservations.Data;
```

```
using PtixiakiReservations.Models;
using PtixiakiReservations.Seeders;
using PtixiakiReservations.Services;
using Serilog;
using Serilog.Exceptions;
using Serilog.Formatting.Elasticsearch;
using Serilog.Sinks.Elasticsearch;
using System;

var builder = WebApplication.CreateBuilder(args);

// Configure Serilog with Elasticsearch
var elasticUrl = builder.Configuration["ElasticSettings:Url"] ??
"http://elasticsearch:9200";
var indexPrefix = builder.Configuration["ElasticSettings:DefaultIndex"] ??
"events";

// Configure Serilog
Log.Logger = new LoggerConfiguration()
   .ReadFrom.Configuration(builder.Configuration)
   .Enrich.FromLogContext()
   .Enrich.WithMachineName()
   .Enrich.WithExceptionDetails()
   .WriteTo.Console()
   .WriteTo.Debug()
   .WriteTo.Elasticsearch(
       new Serilog.Sinks.Elasticsearch.ElasticsearchSinkOptions(new
Uri(elasticUrl))
       {
           IndexFormat = $"{indexPrefix}-logs-{DateTime.UtcNow:yyyy-MM}",
           AutoRegisterTemplate = true,
           OverwriteTemplate = true,
           DetectElasticsearchVersion = true,
           AutoRegisterTemplateVersion = AutoRegisterTemplateVersion.ESv7,
           NumberOfShards = 1,
           NumberOfReplicas = 0,
           ModifyConnectionSettings = x =>
               x.BasicAuthentication("",
"").ServerCertificateValidationCallback((o, c, ch, e) => true),
           CustomFormatter = new ElasticsearchJsonFormatter(),
           EmitEventFailure = EmitEventFailureHandling.WriteToSelfLog |
                              EmitEventFailureHandling.WriteToFailureSink |
                              EmitEventFailureHandling.RaiseCallback
       })
   .CreateLogger();

builder.Host.UseSerilog();

try
```

```
{
   Log.Information("Starting web application");

   // Configure services
   builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnecti
on")));

   builder.Services.AddSignalR();

   builder.Services.Configure<ForwardedHeadersOptions>(options =>
   {
       options.ForwardedHeaders = ForwardedHeaders.XForwardedFor |
                                  ForwardedHeaders.XForwardedProto |
                                  ForwardedHeaders.XForwardedHost;
       options.KnownNetworks.Clear();
       options.KnownProxies.Clear();
   });

   builder.Services.AddIdentity<ApplicationUser, ApplicationRole>(options =>
       {
           options.Stores.MaxLengthForKeys = 128;

           // Password settings
           options.Password.RequireDigit = true;
           options.Password.RequireLowercase = false;
           options.Password.RequireNonAlphanumeric = false;
           options.Password.RequireUppercase = false;
           options.Password.RequiredLength = 6;
           options.Password.RequiredUniqueChars = 0;
       })
       .AddEntityFrameworkStores<ApplicationDbContext>()
       .AddDefaultUI()
       .AddDefaultTokenProviders();

   builder.Services.AddControllersWithViews().AddXmlSerializerFormatters();
   builder.Services.AddRazorPages();


builder.Services.Configure<ElasticSettings>(builder.Configuration.GetSection("E
lasticSettings"));
   builder.Services.AddSingleton<IElasticSearch, ElasticSearchService>();

   builder.Services.AddMvc();

   builder.Services.ConfigureApplicationCookie(options =>
   {
       options.LoginPath = "/Identity/Account/Login";
```

```
        options.LogoutPath = "/Identity/Account/Logout";
        options.AccessDeniedPath = "/Identity/Account/AccessDenied";
        options.ReturnUrlParameter = "returnUrl";
        options.SlidingExpiration = true;

        options.Cookie.SameSite = Microsoft.AspNetCore.Http.SameSiteMode.Lax;
    });

    var app = builder.Build();

    // Configure the HTTP Request Pipeline
    using (var scope = app.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        var context = services.GetRequiredService<ApplicationDbContext>();
        var userManager =
services.GetRequiredService<UserManager<ApplicationUser>>();
        var roleManager =
services.GetRequiredService<RoleManager<ApplicationRole>>();

        try
        {
            Log.Information("Seeding database...");

            await DataSeeder.SeedTestDataAsync(context, userManager,
roleManager, services);

            // Seed test users for role testing
            await TestUserSeeder.SeedTestUsersAsync(userManager, roleManager);

            Log.Information("Database seeded successfully");
        }
        catch (Exception ex)
        {
            Log.Error(ex, "An error occurred while seeding the database");
        }
    }

    app.UseForwardedHeaders();

    if (app.Environment.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseMigrationsEndPoint();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // Don't redirect to HTTPS when behind a proxy
```

```
        // app.UseHsts();
    }

    // Don't use HTTPS redirection when behind a reverse proxy like Traefik
    // app.UseHttpsRedirection();

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();

    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Events}/{action=EventsForToday}/{id?}");
    app.MapRazorPages();

    app.Use(async (context, next) =>
    {
        if (context.Request.Path == "/")
        {
            context.Response.Redirect("/Events/EventsForToday");
            return;
        }

        await next();
    });

    app.Run();

    return 0;
}
catch (Exception ex)
{
    Log.Fatal(ex, "Application terminated unexpectedly");
    return 1;
}
finally
{
    Log.CloseAndFlush();
}
```

6.2 Μοντέλα Δεδομένων (Entities) & DbContext

Βασικές οντότητες (όπως ορίστηκαν στο Κεφ. 5):

```
public class Venue

{

   public int Id { get; set; }

   public string Name { get; set; }

   public string Address { get; set; }

   public int CityId { get; set; }

   [ForeignKey("CityId")] public City City { get; set; }

   public string PostalCode { get; set; }

   public string Phone { get; set; }

   public string UserId { get; set; }

   [ForeignKey("UserId")] public ApplicationUser ApplicationUser { get; set; }

   public string imgUrl { get; set; }

}
```

```
public class SubArea

{

   public int Id { get; set; }

   public string AreaName { get; set; }

   public decimal Width { get; set; }

   public decimal Height { get; set; }

   public decimal Top { get; set; }

   public decimal Left { get; set; }
```

```
    public decimal Rotate { get; set; }

    public string Desc { get; set; }

    public int VenueId { get; set; }

    [ForeignKey("VenueId")] public Venue Venue { get; set; }

}
```

```
public class Seat

{

    public int Id { get; set; }

    public decimal X { get; set; }

    public decimal Y { get; set; }

    public string Name { get; set; }

    public bool Available { get; set; }

    public int SubAreaId { get; set; }

    [ForeignKey("SubAreaId")] public SubArea SubArea { get; set; }

}
```

```
public class Reservation

{

    public int ID { get; set; }

    public string UserId { get; set; }

    [ForeignKey("UserId")] public ApplicationUser ApplicationUser { get; set; }

    public int SeatId { get; set; }

    [ForeignKey("SeatId")] public Seat Seat { get; set; }

    public int EventId { get; set; }

    [ForeignKey("EventId")] public Event Event { get; set; }
```

```
    public DateTime Date { get; set; }

    public TimeSpan Duration { get; set; }

}
```

```
public class EventType

{

    public int Id { get; set; }



    public string Name { get; set; }

}
```

```
public class Event

{

    public int Id { get; set; }

    public string Name { get; set; }

    public DateTime StartDateTime { get; set; }

    public DateTime EndTime { get; set; }

    public int EventTypeId { get; set; }

    [ForeignKey("EventTypeId")] public EventType EventType { get; set; }

    public int VenueId { get; set; }

    [ForeignKey("VenueId")] public Venue Venue { get; set; }

    public int? SubAreaId { get; set; }

    [ForeignKey("SubAreaId")] public SubArea SubArea { get; set; }

    public int? FamilyEventId { get; set; }

    [ForeignKey("FamilyEventId")] public FamilyEvent FamilyEvent { get; set; }

}
```

```
public class City

{

   public int Id { get; set; }



   public string Name { get; set; }

}
```

```
public class ApplicationUser : IdentityUser

{

   public ApplicationUser() : base()

   {

   }



   public String FirstName { get; set; }

   public String LastName { get; set; }

   public int? CityId { get; set; }

   [ForeignKey("CityId")] public City City { get; set; }

   public string Address { get; set; }

   public string PostalCode { get; set; }

   public bool HasRequestedVenueManagerRole { get; set; }

   public DateTime? VenueManagerRequestDate { get; set; }

   public string? VenueManagerRequestReason { get; set; }

   public string? VenueManagerRequestStatus { get; set; } // "Pending",
"Approved", "Rejected"



   public static implicit operator ApplicationUser(Task<ApplicationUser> v)
```

```
    {
        throw new NotImplementedException();
    }
}
```

```
public class ApplicationRole : IdentityRole
{
    public string description { get; set; }
    public DateTime creationDate { get; set; }

    public ApplicationRole() : base()
    {
    }

    public ApplicationRole(string roleName) : base(roleName)
    {
    }

    public ApplicationRole(string roleName, string description, DateTime
creationDate) : base(roleName)
    {
        this.description = description;
        this.creationDate = creationDate;
    }
```

ApplicationDbContext με indexes/περιορισμούς:

```
using System.Linq;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using PtixiakiReservations.Models;



namespace PtixiakiReservations.Data

{

   public class ApplicationDbContext : IdentityDbContext<ApplicationUser,
ApplicationRole, string>

   {

       public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)

         : base(options)

       {

       }

       public DbSet<Venue> Venue { get; set; }

       public DbSet<Reservation> Reservation { get; set; }

       public DbSet<SubArea> SubArea { get; set; }

       public DbSet<Event> Event { get; set; }

       public DbSet<FamilyEvent> FamilyEvent { get; set; }

       public DbSet<Seat> Seat { get; set; }



       public DbSet<City> City { get; set; }



       public DbSet<EventType> EventType { get; set; }



       protected override void OnModelCreating(ModelBuilder modelbuilder)
```

```
        {

            foreach (var relationship in
modelbuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))

            {

                relationship.DeleteBehavior = DeleteBehavior.Restrict;

            }



            base.OnModelCreating(modelbuilder);

        }

    }



}
```

Migrations & seeding:

- dotnet ef migrations add InitialCreate → dotnet ef database update.
- Στο Program.cs, αρχικό seeding ρόλων **Admin**, **VenueOwner**, **User** και ενός demo Venue.

6.3 Authentication/Authorization (Identity)

- Ρόλοι: **Admin**, **VenueOwner**, **User**.
- Policies/claims: CanEditVenue, CanPublishEvent, CanApproveBooking.
- Ελάχιστη πολιτική κωδικών, email confirmation = **true**, προαιρετικό 2FA.

- Στα endpoints χρησιμοποιείται:

Κεφάλαιο 7 – Δοκιμές & Αξιολόγηση

(Λειτουργικές, ολοκλήρωσης, απόδοσης, ασφάλειας, χρηστικότητας και ανθεκτικότητας)

Το κεφάλαιο αυτό αποτυπώνει **πώς** επαληθεύτηκε ότι το σύστημα ανταποκρίνεται στις απαιτήσεις του Κεφ. 3 και στις τεχνικές επιλογές του Κεφ. 4–5. Η προσέγγιση καλύπτει δοκιμές **μονάδας**, **ολοκλήρωσης**, **API/συμβολαίων**, **απόδοσης/φόρτου**, **ασφάλειας**, **χρηστικότητας**, και **ανθεκτικότητας**. Στο τέλος συνοψίζονται τα **KPIs** και η επίτευξή τους.

7.1 Στόχοι δοκιμών

1. **Ορθότητα λειτουργιών**: κράτηση χωρίς υπερπώληση, έλεγχος χωρητικότητας/επικάλυψης, σωστή αναζήτηση.

2. **Απόδοση**: χρόνος απόκρισης αναζήτησης < **300 ms** για μεγάλο όγκο δεδομένων.

3. **Ασφάλεια**: ορθή εφαρμογή ρόλων/πολιτικών, rate-limiting, ασφαλής διαχείριση συνεδριών.

4. **Ανθεκτικότητα**: αντοχή σε σφάλματα εξαρτήσεων (π.χ. προσωρινή μη διαθεσιμότητα Elasticsearch).

5. **Χρηστικότητα**: σαφή ροή για Venue Owners και τελικούς χρήστες, χωρίς “dead ends”.

7.2 Περιβάλλον & Δεδομένα Δοκιμών

- **Έκδοση**: .NET 8.0, ASP.NET Core 8, EF Core 8, ASP.NET Identity.
- **Υποδομή**: Docker Compose (Web, SQL Server 2022, Elasticsearch 8.x, Watchtower απενεργοποιημένο στις δοκιμές).
- **Dataset δοκιμών** (συνθετικό): **~2.000** Venues, **3.500** Subspaces, **50.000** ενεργά Events, **120.000** Bookings.
- **Observability**: Serilog (JSON logs), health checks /health.
- **Methodology**: automated test suite (CI), scripted load tests για σταθερή επαναληψιμότητα.

Σημ.: Τα μεγέθη του dataset επελέγησαν ώστε να προσομοιώσουν πραγματικές συνθήκες φόρτου, χωρίς να απαιτούν υπερβολικούς πόρους.

7.3 Δοκιμές Μονάδας (Unit Tests)

Ενδεικτικές περιπτώσεις:

- **BookingService**
    - Αποτυχία όταν Seats ≤ 0.
    - “SoldOut” όταν requested > remaining.
    - Επιτυχής κράτηση δημιουργεί Booking και μειώνει διαθεσιμότητα.
    - Αποτυχία δημιουργίας όταν Event.EndAt < Now().
- **EventService**

    - MaxSeats ≤ Subspace.Capacity.
    - Απόρριψη overlap στον ίδιο υπο-χώρο (ελεγχόμενη με time-range).
- **SearchService**
    - Επιστρέφει αποτελέσματα με fuzzy “Athens/Αθήνα/Athina”.
    - Φιλτράρει σωστά με date range και geo-distance.

Αποτέλεσμα: Πλήρης επιτυχία στο 100% των unit tests (χωρίς flaky συμπεριφορές).

7.4 Δοκιμές Ολοκλήρωσης (Integration)

- Εκκίνηση **πραγματικών** SQL/ES containers (Docker) αποκλειστικά για το test run.
- **Scenario**: Δημιουργία Venue → Subspace → Event → State-change → Indexing → Αναζήτηση → Κράτηση.
- Έλεγχος ότι μετά το SaveChanges() το event **εμφανίζεται** στον ES index (asynchronous indexing worker με μικρό polling TTL).
- Έλεγχος **συναλλαγών**: σε αστοχία DB update, rollback χωρίς “ορφανά” στατιστικά.

Αποτέλεσμα: Επιτυχής ροή end-to-end, χρόνος indexing per event **~60–120 ms** (τοπικά).

7.5 Δοκιμές API & Συμβολαίων

- Επιβεβαίωση **HTTP status codes**:
    - 200/201 σε επιτυχείς λειτουργίες, 400 σε λανθασμένα input, 401/403 σε μη εξουσιοδοτημένες προσβάσεις, 404 σε μη ευρεθέντα, 409 σε συγκρούσεις (π.χ. sold-out).
- **RFC 7807** ProblemDetails σε σφάλματα για συνεπή μηνύματα.
- Έλεγχος **Idempotency** σε re-tries (π.χ. post κράτησης να μη διπλασιάζει σε network glitches).

7.6 Δοκιμές Απόδοσης & Φόρτου

7.6.1 Σενάρια

1. **Αναζήτηση**: 70% read (GET /search), 20% προβολή λεπτομερειών (GET /events/{id}), 10% filtrations (date/geo).
2. **Κράτηση**: 90% check διαθέσιμων, 10% POST κράτησης (πιο “βαρύ” σενάριο λόγω συναλλαγών DB).

7.6.2 Κριτήρια αποδοχής (KPIs)

- **Search latency**: Median < **200 ms**, p95 < **300 ms**.
- **Bookings**: Median < **150 ms**, p95 < **250 ms** με 30 ταυτόχρονους χρήστες.
- **Error rate**: < **1%** σε p95 σενάρια.

- **Oversell**: **0** περιπτώσεις.

7.6.3 Αποτελέσματα (ενδεικτικά)

- **/search** (dataset 50k events): Median **145 ms**, p95 **280 ms**, **~220 RPS** σε single web replica.
- **/bookings (POST)**: Median **95 ms**, p95 **210 ms** με **30** ταυτόχρονους.
- **Συγκριτικά**: Πριν τον εκσυγχρονισμό (SQL FTS, .NET 6), η αναζήτηση εμφάνιζε median > **500–600 ms** στο ίδιο dataset. Με ES + .NET 8 επιτεύχθηκε **~3×** βελτίωση.

Παρατήρηση: το output caching μειώνει επιπλέον ~20–30 ms στα δημοφιλή queries.

7.7 Δοκιμές Ασφάλειας

- **Authentication/Authorization**
    - Υποχρεωτική επιβεβαίωση email.
    - Ρόλοι Admin, VenueOwner, User με policies/claims (δοκιμές θετικής/αρνητικής πρόσβασης σε endpoints).
- **Rate-Limiting**
    - Ενεργοποιημένο στα search endpoints· επιτυχής απόρριψη burst > ορίου.
- **Input Validation & XSS**

    - Ειδικοί έλεγχοι encoding σε περιγραφές event, αποφυγή script-injection.

- **Secrets**
    - Έλεγχος ότι **δεν** περιλαμβάνονται σε repo· μόνο σε .env/secrets store.

- **TLS**
    - Υποχρεωτική χρήση HTTPS (reverse proxy).

Αποτέλεσμα: Πέρασμα όλων των security checks που προβλέπει η τρέχουσα φάση (χωρίς δυναμικά SAST/DAST εργαλεία πέραν των βασικών).

7.8 Δοκιμές Χρηστικότητας (UX)

- **Δείγμα**: 5 Venue Owners, 5 τελικοί χρήστες.
- **Σενάρια**:
    - Δημιουργία χώρου + 2 υπο-χώρων.
    - Δημιουργία event με capacity < subspace capacity.
    - Εύρεση event “Athens” και κράτηση 2 θέσεων.
- **Μετρικές**: χρόνος ολοκλήρωσης, λάθη, υποκειμενική ικανοποίηση (SUS).
- **Αποτέλεσμα**:
    - 100% ολοκλήρωση στα βασικά σενάρια χωρίς βοήθεια.

- Μέσος χρόνος δημιουργίας event: **1′45″**.
- **SUS**: **82/100** (θεωρείται “καλό έως άριστο”).

7.9 Ανθεκτικότητα & Σφάλματα Εξαρτήσεων

- **ES node down**: Προσομοίωση διακοπής. Η αναζήτηση αποτυγχάνει “κομψά” με φιλικό μήνυμα· οι υπόλοιπες λειτουργίες (CRUD, κράτηση) παραμένουν λειτουργικές.
- **DB latency spike**: Τεχνητή καθυστέρηση 200–300 ms → κράτηση διατηρείται κάτω από p95 **250 ms** (λόγω ελαφριών συναλλαγών και σύντομων queries).
- **Network jitter**: Re-tries με exponential backoff στα ES calls του SearchService.

7.10 Προσβασιμότητα (Accessibility)

- **WCAG 2.1 AA**: έλεγχος contrast, δυνατότητα πλοήγησης με πληκτρολόγιο, ARIA labels στα βασικά controls, ALT σε εικόνες.
- **Αποτέλεσμα**: Δεν εντοπίστηκαν blockers· μερικά βελτιώθηκαν (focus states πιο ορατά, περιγραφικά labels).

7.11 Αξιολόγηση έναντι Απαιτήσεων (Κεφ. 3)

- **FR – Δημιουργία/Διαχείριση Event/Spaces**: ✅
- **FR – Κράτηση με έλεγχο διαθεσιμότητας**: ✅ (0 oversell)
- **FR – Αναζήτηση με όνομα, ημερομηνίες, τοποθεσία**: ✅ (fuzzy/filters/geo)
- **NFR – Απόδοση Αναζήτησης < 300 ms**: ✅ (p95 280 ms)
- **NFR – Ασφάλεια (Roles/Policies/Rate-Limit)**: ✅
- **NFR – Διαθεσιμότητα/Resilience**: ✅ (graceful degradation σε ES failure)
- **NFR – Συντηρησιμότητα (CI/CD)**: ✅ (full pipeline + reproducible builds)

7.12 Μαθήματα & Βελτιώσεις

1. **Indexing Strategy**: Η ασύγχρονη ροή on-commit λειτούργησε καλά· προτείνεται μικρό **debounce** batch (π.χ. 200 ms) για μαζικές αλλαγές.
2. **Caching**: Τα δημοφιλή ερωτήματα ωφελούνται από **μικρό TTL** (15–30 s)· προτείνεται fine-tuning ανά εποχές/πόλεις.
3. **Observability**: Προσθήκη application-level metrics (π.χ. “search_suggestions_generated”) θα βοηθούσε σε βελτιστοποίηση UX.
4. **Security Hardening**: Στην επόμενη φάση, αυτόματο **SAST/DAST** στον CI και **CSP headers** για περαιτέρω μείωση επιφάνειας XSS.

5. **Scalability**: Σε μεγαλύτερη κίνηση, αξίζει **δεύτερο web replica** και ES **replica shards** για καλύτερη ανάγνωση.

7.13 Συμπέρασμα Κεφαλαίου

Οι δοκιμές επιβεβαίωσαν ότι το εκσυγχρονισμένο σύστημα:

- **λειτουργεί σωστά** στα κρίσιμα σενάρια (δημιουργία χώρου/υπο-χώρου, events, κράτηση),
- επιτυγχάνει **στόχους απόδοσης** (αναζήτηση p95 < 300 ms) στο dataset των 50k events,
- εφαρμόζει **ασφαλή πολιτική πρόσβασης** με ρόλους/claims και rate-limiting,
- παρουσιάζει **ανθεκτικότητα** σε σφάλματα εξαρτήσεων και
- προσφέρει **καλή εμπειρία χρήσης** (SUS 82/100).

Κεφάλαιο 8 – Συζήτηση

(ερμηνεία αποτελεσμάτων, σχεδιαστικές επιλογές, περιορισμοί, ρίσκα, κανονιστική συμμόρφωση)

Το κεφάλαιο αυτό «δένει» τα ευρήματα των δοκιμών (Κεφ. 7) με τις επιλογές σχεδιασμού (Κεφ. 4–6). Εστιάζει στο **τι σημαίνουν πρακτικά τα αποτελέσματα**, **τι κερδίσαμε/τι χάσαμε** από κάθε τεχνική επιλογή, **ποιες είναι οι αδυναμίες** της τρέχουσας έκδοσης και **πώς μετριάζονται**.

8.1 Σκοπός του κεφαλαίου

- Να ερμηνεύσει αν οι στόχοι (Κεφ. 3) επιτεύχθηκαν στην πράξη.
- Να αξιολογήσει τις τεχνολογικές επιλογές με κριτήρια απόδοσης, ασφάλειας, κόστους, συντηρησιμότητας.
- Να αναδείξει κινδύνους/περιορισμούς και να προτείνει ρεαλιστικούς τρόπους βελτίωσης.

8.2 Ερμηνεία αποτελεσμάτων δοκιμών

Απόδοση αναζήτησης: p95 ≈ **280 ms** σε dataset ~50k events. Πρακτικά, ο χρήστης νιώθει «άμεση» απόκριση, ειδικά με το μικρό output-caching. Η μετάβαση από SQL FTS σε Elasticsearch αποδίδει **~3×** βελτίωση, που δικαιολογεί την πολυπλοκότητα του ES (ρύθμιση, monitoring).

Κρατήσεις: p95 ≈ **210 ms** με 30 ταυτόχρονους. Η σχεδίαση με απλές, σύντομες συναλλαγές και σαφείς ελέγχους συνέχειας (capacity, overlaps) λειτούργησε χωρίς oversell. Αυτό είναι κρίσιμο, γιατί η αξιοπιστία «χτίζει» εμπιστοσύνη στον ιδιοκτήτη χώρου.

Ανθεκτικότητα: Προσομοίωση πτώσης ES → graceful υποβάθμιση (αναζήτηση εκτός, αλλά CRUD/κρατήσεις συνεχίζουν). Σημαντικό ότι η αποτυχία μίας εξάρτησης **δεν παραλύει** τον πυρήνα του συστήματος.

Χρηστικότητα: SUS **82/100** και πλήρης ολοκλήρωση βασικών σεναρίων από μη τεχνικούς χρήστες. Η πληροφοριακή αρχιτεκτονική (χώρος → υπο-χώροι → events) κρίθηκε από τους δοκιμαστές «διαισθητική».

Συμπέρασμα: Οι μετρήσεις δεν είναι απλώς «καλές στο χαρτί», αλλά **αλλάζουν την εμπειρία**: γρήγορη εύρεση, σίγουρη κράτηση, σταθερότητα.

8.3 Αποτίμηση σχεδιαστικών επιλογών

8.3.1 .NET 6 → .NET 8

- **Υπέρ**: ώριμη LTS πλατφόρμα, αισθητή βελτίωση cold-starts/throughput, ενιαίο minimal hosting, rate-limiting/output-caching built-in.
- **Κατά**: μικρή εργασία migration (warnings, analyzers, lib updates).
- **Κρίση**: σωστή επιλογή. Η βάση κώδικα παραμένει σύγχρονη χωρίς rewrite.

8.3.2 EF Core + Identity (single DbContext)

- **Υπέρ**: απλοποιεί migrations, ενιαία πηγή αλήθειας για users/roles και domain.
- **Κατά**: λιγότερη «απομόνωση» ανά bounded context.
- **Κρίση**: λογικός συμβιβασμός για το σημερινό μέγεθος του έργου. Εάν «φουσκώσει», εξετάζονται πολλαπλά contexts.

8.3.3 Elasticsearch αντί SQL FTS

- **Υπέρ**: fuzzy, autocomplete, geo-queries, καλύτερο relevancy.
- **Κατά**: λειτουργική πολυπλοκότητα (ρυθμίσεις, backups, monitoring).
- **Κρίση**: η απόδοση/εμπειρία χρήστη δικαιολογεί το κόστος.

8.3.4 Self-hosted Gitea CI/CD

- **Υπέρ**: πλήρης έλεγχος κώδικα/μυστικών, μηδενικές εξαρτήσεις σε τρίτους, ευελιξία στο pipeline.
- **Κατά**: απαιτεί στοιχειώδες DevOps «νοικοκυριό» (updates, runner υγεία).
- **Κρίση**: ταιριάζει σε ακαδημαϊκό/μικρό οργανισμό που θέλει αυτονομία.

8.3.5 Docker + Watchtower

- **Υπέρ**: reproducible builds, γρήγορα rollouts, απλή αυτο-αναβάθμιση.
- **Κατά**: προσοχή σε «τυφλά» updates· απαιτούνται tags/κανόνες για ασφαλή κύκλους παραγωγής.
- **Κρίση**: ιδανικό για μικρό/μεσαίο περιβάλλον. Για πολύ υψηλή διαθεσιμότητα, το επόμενο βήμα θα ήταν orchestrator (π.χ. Kubernetes).

8.3.6 UI (Razor/Blazor)

- **Υπέρ**: ταχεία παραγωγικότητα, κοινό μοντέλο ασφάλειας με το API.
- **Κατά**: λιγότερο «headless» από μια SPA + API προσέγγιση.
- **Κρίση**: συνειδητή επιλογή για να μεγιστοποιηθεί η ταχύτητα ανάπτυξης χωρίς να θυσιάζεται η ποιότητα.

8.4 Περιορισμοί & τεχνικό χρέος

- **Αναζητήσεις πολύπλοκων συνδυασμών** (πολλαπλά φίλτρα + semantic reranking) δεν έχουν ακόμα πλήρως βελτιστοποιηθεί.

- **Index drift**: αν αποτύχει ο async indexer, χρειάζεται περιοδικό bulk reconcile (υλοποιημένο, αλλά θέλει monitoring).

- **Διαχείριση πληρωμών**: επί του παρόντος εκτός πεδίου (proof-of-concept). Η προσθήκη gateway απαιτεί idempotency keys, webhooks, reconciliation.

- **Multi-tenancy**: σήμερα υπάρχει λογική ανά χρήστη/ρόλο, όχι σκληρό tenant isolation (π.χ. row-level security).

- **Observability**: logs/traces ok, αλλά απουσιάζει πλήρες metrics dashboard ανά feature.

8.5 Κόστος λειτουργίας & συντήρησης (εκτίμηση)

- **Web app**: 1–2 replicas (ανάλογα το traffic).

- **SQL Server**: 1 primary (prod) + backups.

- **Elasticsearch**: 3 μικροί κόμβοι (1 master-eligible + 2 data) επαρκούν για το τρέχον μέγεθος.

- **CI/CD**: 1 Gitea + 1 Runner.

  Το TCO μεγαλώνει προοδευτικά με τον όγκο αναζητήσεων· οριζόντια κλιμάκωση ES είναι το πρώτο σημείο προσοχής.

8.6 Κίνδυνοι & μετριασμός

- **Κακό update μέσω Watchtower** → *Μετριασμός*: pinned minor tags (π.χ. 8.0.x), canary container, άμεσο rollback με προηγούμενο tag.
- **Αποτυχία indexing** → *Μετριασμός*: dead-letter queue ή persistent log + nightly full bulk.
- **Αύξηση traffic** → *Μετριασμός*: δεύτερο web replica, ES replica shards, CDN για assets.
- **Data loss** → *Μετριασμός*: point-in-time backups DB, snapshot lifecycle για ES indices.
- **Security regressions** → *Μετριασμός*: τακτικά updates, SAST/DAST στον CI, secret scanning.

8.7 Ηθικές & νομικές πτυχές (GDPR)

- **Ελαχιστοποίηση δεδομένων**: δεν απαιτούνται περιττά προσωπικά στοιχεία για κράτηση.
- **Διαφάνεια**: πολιτική απορρήτου και σαφείς σκοποί επεξεργασίας.
- **Δικαιώματα υποκειμένων**: download/διαγραφή λογαριασμού, διόρθωση στοιχείων.
- **Ασφάλεια**: κρυπτογράφηση εν κινήσει (TLS), προαιρετικά κρυπτογράφηση at-rest, ισχυρή πολιτική κωδικών/2FA.
- **Καταγραφή**: περιορισμός logging προσωπικών δεδομένων (PII scrubbing).

8.8 Μαθήματα από το migration .NET 6 → .NET 8

- Μικρά «ρηχά νερά»: ενημερώσεις nuget, προσαρμογή analyzers, fine-tuning minimal hosting.
- Κέρδος σε **κυκλικούς χρόνους** (build/publish) και **cold start** containers.
- Η υιοθέτηση νέων middleware (rate-limiting, output-caching) έδωσε απτά οφέλη χωρίς εξωτερικές βιβλιοθήκες.

8.9 Επιπτώσεις για μελλοντική εργασία

- Η αρχιτεκτονική **αντέχει κλιμάκωση** χωρίς ριζικές αλλαγές: split contexts, read replicas, message bus για indexing/ειδοποιήσεις.
- Ενσωμάτωση **online πληρωμών** και **PWA/mobile** είναι οργανικά επόμενα βήματα.
- Δυνατότητα **hybrid search** (keyword + semantic embeddings) για προτάσεις/συστάσεις εκδηλώσεων.

8.10 Σύνοψη κεφαλαίου

Η συζήτηση επιβεβαιώνει ότι οι επιλογές (.NET 8, EF/Identity, Elasticsearch, Docker, Gitea CI/CD, Watchtower) **εξυπηρέτησαν τον σκοπό**: γρήγορη

αναζήτηση, αξιόπιστες κρατήσεις, απλά deployments. Τα **ρίσκα** είναι γνωστά και **μετριάσιμα** με ελαφρές διαδικαστικές προσθήκες (tags, backups, monitoring). Τα **όρια** της τρέχουσας έκδοσης είναι σαφή και χαράσσουν καθαρό οδικό χάρτη για επόμενες εκδόσεις με πληρωμές, ισχυρότερο observability και περαιτέρω κλιμάκωση.

Κεφάλαιο 9 – Συμπεράσματα & Μελλοντικές Επεκτάσεις

Το κεφάλαιο αυτό συνοψίζει τα ευρήματα της εργασίας, αποτιμά τη συνεισφορά της πλατφόρμας στο πρόβλημα της οργάνωσης εκδηλώσεων, καταγράφει τους περιορισμούς και χαράσσει ρεαλιστικό οδικό χάρτη για εξέλιξη.

9.1 Συνολική Ανακεφαλαίωση

Αντικείμενο της πτυχιακής ήταν ο εκσυγχρονισμός μιας πλατφόρμας κράτησης εκδηλώσεων, από μια προγενέστερη βάση, σε μια **σύγχρονη, κλιμακώσιμη και αυτοματοποιημένη** λύση. Η εργασία:

- **Αναβάθμισε** την τεχνολογική βάση από **.NET 6.0** σε **.NET 8.0 (LTS)**.
- **Εδραίωσε** καθαρή αρχιτεκτονική (C4) με διακριτά layers (Web, Application, Domain, Infrastructure).
- **Υλοποίησε** κρίσιμους επιχειρησιακούς κανόνες για χώρους/υπο-χώρους, χωρητικότητες και κρατήσεις χωρίς υπερπώληση.
- **Ενσωμάτωσε Elasticsearch** για γρήγορη, «ανθρώπινη» αναζήτηση (fuzzy, autocomplete, προαιρετικό geo).
- **Τυποποίησε** DevOps πρακτικές με **Docker**, **self-hosted Gitea CI/CD** και **Watchtower** για αυτόματες αναβαθμίσεις.

- **Τεκμηρίωσε και δοκίμασε** πλήρως το σύστημα (λειτουργικά, ολοκλήρωσης, απόδοσης, ασφάλειας, UX).

Τα αποτελέσματα (Κεφ. 7) επιβεβαίωσαν **p95 ≈ 280 ms** σε αναζητήσεις 50k events, **p95 ≈ 210 ms** σε κρατήσεις, **0 oversell** και **SUS 82/100**.

9.2 Συνεισφορά της Εργασίας

1. **Λειτουργική συνεισφορά**: Πλήρης ροή για ιδιοκτήτες χώρων (δήλωση χώρου → υπο-χώρων → events → κρατήσεις) με εγγύηση χωρητικότητας και αποφυγή επικαλύψεων.

2. **Τεχνική συνεισφορά**:

    - Ενοποίηση **EF Core + Identity** σε ενιαίο context με policies/claims.

    - **Search layer** με ES και NEST client, σχεδιασμένο για ελληνικό/λατινικό κείμενο.

    - **CI/CD** pipeline που μειώνει τον χρόνο «commit→production» σε λίγα λεπτά.

3. **Μεθοδολογική συνεισφορά**: Πλήρης τεκμηρίωση (C4, ERD, APIs, mappings) και στρατηγική δοκιμών που μπορεί να επαναληφθεί σε άλλα έργα.

9.3 Ερευνητικά Ερωτήματα – Τι απαντήθηκε

- **Μπορεί η πλατφόρμα να προσφέρει ταχεία αναζήτηση σε κλίμακα;**

  Ναι· με ES 8 και κατάλληλους analyzers επιτεύχθηκε p95 < 300 ms.

- **Είναι εφικτές οι κρατήσεις χωρίς υπερπώληση;**

  Ναι· με συναλλαγές, έλεγχο χωρητικότητας και απλούς κανόνες ακεραιότητας.

- **Μπορεί να αυτοματοποιηθεί ο κύκλος διάθεσης;**

  Ναι· Gitea Runner → Docker Hub → Watchtower υλοποιούν συνεχή παράδοση.

- **Η εμπειρία χρήσης είναι επαρκής;**

  Ναι· SUS 82/100, πλήρης ολοκλήρωση βασικών σεναρίων χωρίς βοήθεια.

9.4 Περιορισμοί

- **Λειτουργικοί**: Δεν ενσωματώθηκαν online πληρωμές· η ροή κράτησης λειτουργεί ως proof-of-concept.

- **Τεχνικοί**: Το indexing είναι ασύγχρονο· απαιτεί periodic bulk reconciliation και monitoring για drift.

- **Αρχιτεκτονικοί**: Single DbContext για λόγους απλότητας—σε πολύ μεγάλη κλίμακα ίσως χρειαστεί διάσπαση.

- **Observability**: Υπάρχουν logs/traces/health, αλλά λείπει πλήρης πίνακας επιχειρησιακών metrics ανά feature.

9.5 Μελλοντικές Επεκτάσεις

9.5.1 Βραχυπρόθεσμα (0–3 μήνες)

- **Πληρωμές** (Stripe/PayPal): webhooks, idempotency keys, απόδειξη/τιμολόγηση.
- **Waitlist & cancellations** με αυτοματοποιημένη ειδοποίηση και ανακατανομή θέσεων.
- **Metrics dashboard**: latency ανά τύπο ερωτήματος, conversion funnel αναζήτησης→κράτησης.

9.5.2 Μεσοπρόθεσμα (3–6 μήνες)

- **Geo-search UI** με χάρτη (clustering pins, distance filters).
- **Semantic/hybrid search** (keyword + embeddings) για προτάσεις εκδηλώσεων.
- **Accessibility hardening** προς πλήρη συμμόρφωση WCAG 2.1 AA σε όλα τα flows.

9.5.3 Μακροπρόθεσμα (6–12 μήνες)

- **Mobile/PWA**: offline e-tickets, push notifications, camera-based check-in.
- **Multi-tenancy**: αυστηρό isolation (π.χ. row-level security) για επιχειρηματικούς πελάτες.
- **Orchestration**: μετάβαση σε **Kubernetes** αν απαιτηθεί υψηλή διαθεσιμότητα/αυτοθεραπεία.

9.6 Συντήρηση & Διακυβέρνηση Λογισμικού

- **Lifecycle**: Ετήσιος έλεγχος minor αναβαθμίσεων .NET/ES, μηνιαία updates security.
- **Branching & reviews**: Protected main, υποχρεωτικά code reviews, CI status checks.
- **Ασφάλεια**: Secret rotation ανά τρίμηνο, αυτόματο scanning σε NuGet/Docker.
- **Backups**: DB PITR, ES snapshots με τακτικό restore-test.

9.7 Επιπτώσεις & Αξία

- **Για ιδιοκτήτες χώρων**: αυτονομία στη διαχείριση υπο-χώρων/χωρητικοτήτων, σαφή εικόνα κρατήσεων/πληρότητας.

- **Για τελικούς χρήστες**: γρήγορη εύρεση (ανοχή σε ορθογραφικά, προτάσεις), απλή κράτηση, καθαρά μηνύματα λάθους.

- **Για την ομάδα ανάπτυξης**: αναπαραγώγιμα builds, μικρός χρόνος διάθεσης, ξεκάθαρα diagnostics.

Βιβλιογραφία

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Addison Wesley. https://www.pearson.com/en-us/subject-catalog/p/compilers-principles-techniques-and-tools/P200000000331/9780321486813

Chen, H., Su, X., & Li, Y. (2020). Geolocation-based access control and filtering in cloud computing. *Journal of Cloud Computing, 9*(1), 1–15. https://doi.org/10.1186/s13677-020-00203-1

Chen, J., Zhou, M. X., Shi, L., & Zhang, Q. (2012). User-oriented evaluation of information retrieval systems: A case study of autocomplete. *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 1153–1162. https://dl.acm.org/doi/10.1145/2396761.2398410

Couture, S., & Toupin, S. (2019). What does the notion of “geo-blocking” mean in digital media? *Media, Culture & Society, 41*(4), 560–575. https://doi.org/10.1177/0163443718808171

Croft, W. B., Metzler, D., & Strohman, T. (2015). *Search Engines: Information Retrieval in Practice*. Pearson. https://www.pearson.com/en-us/subject-catalog/p/search-engines-information-retrieval-in-practice/P200000001390

Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7th ed.). O’Reilly Media. https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/

Hearst, M. A. (2009). *Search User Interfaces*. Cambridge University Press. https://www.cambridge.org/core/books/search-user-interfaces/58C3143B87202A2637ABF5E3A5C8A9DA

Interactive Advertising Bureau (IAB). (2022). *Guide to geotargeting in digital advertising*. IAB Tech Lab. https://iabtechlab.com

Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing* (3rd ed., draft). Stanford University. https://web.stanford.edu/~jurafsky/slp3/

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady, 10*(8), 707–710. https://www.mathnet.ru/eng/dan25590

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys, 33*(1), 31–88. https://doi.org/10.1145/375360.375365

Sebesta, R. W. (2016). *Concepts of Programming Languages* (11th ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/concepts-of-programming-languages/P200000001382

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education. https://www.db-book.com/db7/index.html

Microsoft / .NET / ASP.NET Core

https://learn.microsoft.com/dotnet/core/whats-new/dotnet-8

https://learn.microsoft.com/aspnet/core/?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/security/authentication/identity?view=aspnetcore-8.0

https://learn.microsoft.com/ef/core/

https://learn.microsoft.com/ef/core/performance/

https://learn.microsoft.com/aspnet/core/performance/rate-limit?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/performance/caching/output?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/mvc/views/tag-helpers/builtin-th/cache-tag-helper?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/host-and-deploy/health-checks?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/signalr/introduction?view=aspnetcore-8.0

https://learn.microsoft.com/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-8.0

https://learn.microsoft.com/dotnet/core/deploying/ready-to-run

https://learn.microsoft.com/dotnet/core/deploying/native-aot

https://learn.microsoft.com/aspnet/core/security/cors?view=aspnetcore-8.0

https://learn.microsoft.com/sql/sql-server/

https://learn.microsoft.com/sql/relational-databases/sql-server-index-design-guide

https://learn.microsoft.com/sql/relational-databases/search/full-text-search

Elasticsearch

https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-greek-analyzer.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#fuzziness

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-suggesters-completion.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-edgengram-tokenizer.html

https://www.elastic.co/guide/en/elasticsearch/reference/current/knn-search.html

https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/index.html

https://el.wikipedia.org/wiki/Elasticsearch

DevOps / Containers / Proxy / Observability

https://docs.docker.com/

https://docs.docker.com/compose/

https://docs.docker.com/build/buildx/

https://github.com/containrrr/watchtower

https://docs.gitea.com/

https://docs.gitea.com/usage/actions/quickstart

https://docs.gitea.com/runner/overview

https://doc.traefik.io/traefik/

https://doc.traefik.io/traefik/https/acme/

https://www.jaegertracing.io/docs/

https://opentelemetry.io/docs/languages/net/

https://serilog.net/

https://github.com/serilog-contrib/serilog-sinks-elasticsearch

Πρότυπα / Αρχιτεκτονική

https://www.w3.org/TR/WCAG21/

https://c4model.com/

https://spec.openapis.org/oas/latest.html

https://gomomentus.com/

Repo εργασίας (αναφορά υλοποίησης)

https://github.com/devblaze/PtixiakiReservations