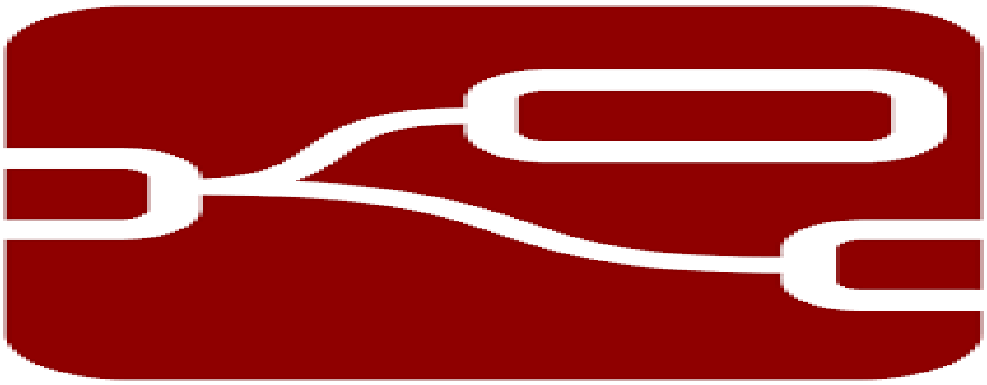
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

# ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

## «Ανάπτυξη πολυχρηστικού πίνακα ελέγχου για την πλατφόρμα Node-RED»




**Του φοιτητή**
**Πιπέρη Κωνσταντίνου**
**Αρ. Μητρώου: 154522**

**Επιβλέπων**
**Αντωνίου Ευστάθιος**
**Βαθμίδα Καθηγητής**


**12/09/2025**

Ανάπτυξη πολυχρηστικού πίνακα ελέγχου της πλατφόρμας Node-RED
23125
Πιπέρης Κωνσταντίνος
Αντωνίου Ευστάθιος
Ημερομηνία ανάληψης Δ.Ε. 8 Μαρτίου 2023
Ημερομηνία περάτωσης Δ.Ε. 12 Σεπτεμβρίου 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 διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.*

*Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Πιπέρη Κωνσταντίνου που την εκπόνησε/αν. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.*

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.

*«Σε όσους με στηρίζουν»*

# Πρόλογος

Επέλεξα αυτό το θέμα γιατί με ενδιέφερε να δουλέψω πάνω σε κάτι πρακτικό που να συνδυάζει τον προγραμματισμό με την επίλυση πραγματικών προβλημάτων. Το Node-RED είναι μια πλατφόρμα που χρησιμοποιείται αρκετά, αλλά δεν υποστηρίζει πολλούς χρήστες ταυτόχρονα. Αυτό αποτέλεσε πρόκληση και αφορμή για την ανάπτυξη μιας λύσης που να καλύπτει αυτό το κενό.

Κατά τη διάρκεια της πτυχιακής γνώρισα καλύτερα τεχνολογίες όπως το Node.js και τη χρήση βάσεων δεδομένων, ενώ παράλληλα έμαθα πώς να οργανώνω και να υλοποιώ μια εφαρμογή από την αρχή. Η διαδικασία δεν ήταν πάντα εύκολη, όμως με βοήθησε να μάθω μέσα από τα προβλήματα και να γίνω πιο δημιουργικός στην αντιμετώπισή τους.

Το μεγαλύτερο κέρδος ήταν ότι απέκτησα εμπειρία σε μια ολοκληρωμένη υλοποίηση, κάτι που με έκανε πιο σίγουρο για τις δυνατότητές μου και πιο έτοιμο για μελλοντικές προκλήσεις.

# Περίληψη

Η παρούσα πτυχιακή εργασία ασχολείται με την ανάπτυξη ενός πολυχρηστικού περιβάλλοντος βασισμένου στο Node-RED, με στόχο την ασφαλή ταυτοποίηση χρηστών και την απομόνωση των ροών τους. Η υλοποίηση επιδιώκει να καλύψει το κενό που υπάρχει στη βασική πλατφόρμα, η οποία δεν έχει σχεδιαστεί εξαρχής για χρήση από πολλούς χρήστες. Μέσω της προτεινόμενης προσέγγισης, κάθε χρήστης αποκτά το δικό του αυτόνομο Node-RED instance, το οποίο εκκινείται δυναμικά από τον κεντρικό server και εκτελείται σε ξεχωριστό port. Με τον τρόπο αυτό διασφαλίζεται η ανεξαρτησία των flows και η προστασία των δεδομένων.

Η ταυτοποίηση των χρηστών πραγματοποιείται με τη χρήση JSON Web Tokens (JWT), ενώ τα στοιχεία τους αποθηκεύονται σε μη σχεσιακή βάση δεδομένων MongoDB. Ο συνδυασμός αυτός εξασφαλίζει τόσο την ασφάλεια της πρόσβασης όσο και την ευελιξία στη διαχείριση των δεδομένων. Παράλληλα, η χρήση του Node.js και του Express.js για την ανάπτυξη του server επιτρέπει την εύκολη ενσωμάτωση επιπλέον λειτουργιών, καθώς και την απλή επικοινωνία μεταξύ των επιμέρους τμημάτων του συστήματος.

Κατά τη διάρκεια της ανάπτυξης δημιουργήθηκε ένα περιβάλλον το οποίο δεν απαιτεί από τον τελικό χρήστη τεχνικές γνώσεις για την εκκίνηση ή τη διαχείριση των υπηρεσιών, αφού όλα τα απαραίτητα βήματα εκτελούνται αυτόματα από τον server. Η λύση αυτή καθιστά την πλατφόρμα περισσότερο φιλική και εύχρηστη, χωρίς να θυσιάζει την ασφάλεια και την επεκτασιμότητα.

Η εργασία ολοκληρώνεται με την αξιολόγηση της προτεινόμενης μεθοδολογίας και την ανάδειξη των πλεονεκτημάτων της σε σχέση με το βασικό Node-RED. Τέλος, προτείνονται μελλοντικές κατευθύνσεις, όπως η κεντρική αποθήκευση flows, η χρήση container τεχνολογιών και η ενσωμάτωση εναλλακτικών μεθόδων ταυτοποίησης, ώστε η πλατφόρμα να μπορέσει να εξελιχθεί σε μια πλήρως πολυχρηστική λύση για ένα ευρύ φάσμα εφαρμογών.

# «Development of a Multi-User Dashboard for the Node-RED Platform»

«Konstantinos Piperis»

## Abstract

This thesis focuses on the development of a multi-user environment based on Node-RED, aiming at secure user authentication and the isolation of individual flows. The implementation addresses a limitation of the original platform, which was not designed to support multiple users simultaneously. Through the proposed approach, each user is assigned an independent Node-RED instance, launched dynamically by the central server and executed on a separate port. This ensures the independence of flows and the protection of user data.

User authentication is handled using JSON Web Tokens (JWT), while user information is stored in a non-relational MongoDB database. This combination provides both secure access control and flexibility in data management. In parallel, Node.js and Express.js are employed for server development, enabling straightforward integration of additional features and facilitating communication among system components.

During the implementation, a user-friendly environment was created, which does not require technical knowledge from the end user for launching or managing services, as all necessary processes are performed automatically by the server. This approach makes the platform more accessible and easier to use, without compromising security or scalability.

The thesis concludes with an evaluation of the proposed methodology and highlights its advantages compared to the core Node-RED platform. Furthermore, future directions are suggested, such as centralized flow storage, the adoption of container technologies, and the integration of alternative authentication methods. These enhancements could enable Node-RED to evolve into a fully multi-user solution suitable for a wide range of applications.

# Πίνακας Περιεχομένων

# Πίνακας Εικόνων



# Πίνακας Πινάκων

# Συντομογραφίες

| | |
|---|---|
| API | Application Programming Interface |
| GUI | Graphical User Interface |
| IoT | Internet of Things |
| HTML | HyperText Markup Language |
| JWT | JSON Web Token |
| JSON | JavaScript Object Notation |
| DB | DataBase |
| MQTT | Message Queuing Telemetry Transport |
| SSL/HTTPS | Secure Socket Layer / HyperText Transfer Protocol Secure |
| Npm | Node Package Manager |
| UI | User Interface |
| FBP | Flow-Based Programming |

# Κεφάλαιο 1: Εισαγωγή

## 1.1 Αντικείμενο της Πτυχιακής Εργασίας

Το αντικείμενο της παρούσας πτυχιακής εργασίας επικεντρώνεται στην ανάπτυξη και την τεχνική επέκταση του **Node-RED dashboard**, με στόχο την ενσωμάτωση μηχανισμών ταυτοποίησης χρηστών μέσω **JSON Web Tokens (JWT)** και τη διαχείριση δεδομένων συνεδρίας. Η εργασία στοχεύει να καλύψει ένα κενό στην υπάρχουσα λειτουργικότητα του Node-RED dashboard, επιτρέποντας τη δημιουργία εξατομικευμένων flows για κάθε χρήστη και τη διασφάλιση της ασφάλειας των δεδομένων μέσω σύγχρονων μεθόδων κρυπτογράφησης. Το πρόσθετο που υλοποιήθηκε προσφέρει δυνατότητες που ενισχύουν τη χρηστικότητα και την επεκτασιμότητα του Node-RED σε περιβάλλοντα παραγωγής, όπου απαιτείται αυστηρή διαχείριση πρόσβασης και εξατομικευμένα δεδομένα για κάθε χρήστη.

Επιπλέον, η εργασία παρέχει έναν οδηγό για τη δημιουργία ενός πακέτου εγκατάστασης του εκτεταμένου προσθέτου, ώστε να είναι εύκολα διαθέσιμο και επαναχρησιμοποιήσιμο από άλλους χρήστες. Μέσα από την ανάπτυξη του προσθέτου και τη χρήση εργαλείων όπως η MongoDB για τη διαχείριση δεδομένων συνεδρίας και χρηστών, παρέχονται προηγμένες λειτουργίες που διευκολύνουν τη διαχείριση ροών στο **Node-RED dashboard** και επεκτείνουν τις δυνατότητες της πλατφόρμας.

## 1.2. Σκοπός και Στόχοι

Ο κύριος σκοπός της παρούσας εργασίας είναι η ανάπτυξη ενός εκτεταμένου προσθέτου για το Node-RED dashboard, το οποίο θα προσφέρει δυνατότητες ασφαλούς ταυτοποίησης χρηστών και εξατομίκευσης των flows που δημιουργούνται ανά συνεδρία. Οι στόχοι περιλαμβάνουν την ενσωμάτωση της υποδομής JWT για τη διαχείριση της ταυτότητας των χρηστών, τη διατήρηση δεδομένων συνεδρίας σε μια μη σχεσιακή βάση δεδομένων όπως η **MongoDB**, καθώς και την ανάπτυξη ενός πακέτου εγκατάστασης για εύκολη υιοθέτηση από άλλους χρήστες.

Ένας σημαντικός στόχος είναι επίσης η βελτίωση της εμπειρίας του χρήστη μέσα από την ομαλή διαχείριση flows και η διασφάλιση της ασφάλειας των δεδομένων με τη χρήση σύγχρονων μεθόδων κρυπτογράφησης. Η εργασία επιδιώκει επίσης την τεκμηρίωση και την παροχή οδηγιών για την ανάπτυξη και την τοπική εκτέλεση του προσθέτου, ώστε να υποστηριχθεί ευρύτερα η κοινότητα των χρηστών του **Node-RED**.

## 1.3. Μεθοδολογία

Η μεθοδολογία της εργασίας στηρίζεται σε μία βήμα προς βήμα προσέγγιση ανάπτυξης λογισμικού, αρχικά με την ανάλυση των υπαρχόντων προβλημάτων και περιορισμών του Node-RED dashboard. Έπειτα, ακολουθεί η σχεδίαση και υλοποίηση της τεχνικής επέκτασης, εστιάζοντας στην ενσωμάτωση της υποδομής JWT και της MongoDB για την αποθήκευση δεδομένων συνεδρίας. Κατά τη διάρκεια της ανάπτυξης, χρησιμοποιήθηκαν σύγχρονα εργαλεία όπως το **Express.js** και το **Node.js** για τη διαχείριση των αιτημάτων και τη δημιουργία του backend της εφαρμογής.

Η διαδικασία ανάπτυξης ολοκληρώθηκε με τη δημιουργία ενός πακέτου εγκατάστασης που περιλαμβάνει όλα τα απαραίτητα αρχεία κώδικα και ρυθμίσεις, επιτρέποντας την εύκολη υλοποίηση του συστήματος σε άλλα περιβάλλοντα. Η μεθοδολογία περιλαμβάνει επίσης την υλοποίηση και δοκιμή των διαφόρων λειτουργιών, προκειμένου να διασφαλιστεί η ορθή λειτουργία και η επίτευξη των στόχων της εργασίας.

## 1.4. Δομή της Πτυχιακής Εργασίας

Η εργασία είναι οργανωμένη σε έξι κεφάλαια, καθένα από τα οποία καλύπτει διαφορετικές πτυχές της υλοποίησης και ανάλυσης του προσθέτου. Στο πρώτο κεφάλαιο, παρουσιάζεται η εισαγωγή στο θέμα, οι στόχοι και η μεθοδολογία. Το δεύτερο κεφάλαιο ασχολείται με τη θεωρία του ασύγχρονου προγραμματισμού στο Node.js, παρέχοντας τη βάση για την κατανόηση της υλοποίησης. Το τρίτο κεφάλαιο εστιάζει στην παρουσίαση του Node-RED, περιγράφοντας τις δυνατότητες του συστήματος και τη χρήση του σε διάφορες εφαρμογές.

Το τέταρτο κεφάλαιο αναλύει τη σχεδίαση της τεχνικής επέκτασης και την ενσωμάτωση του JWT, ενώ το πέμπτο κεφάλαιο επικεντρώνεται στην πρακτική υλοποίηση του προσθέτου και τη δημιουργία του πακέτου εγκατάστασης. Τέλος, το έκτο κεφάλαιο περιλαμβάνει τα συμπεράσματα, την αξιολόγηση της υλοποίησης και τις προτάσεις για μελλοντική επέκταση, προσφέροντας μια ολοκληρωμένη εικόνα της συνεισφοράς της εργασίας.

# Κεφάλαιο 2: Το Μοντέλο Ασύγχρονου Προγραμματισμού στο Node.js

Το Κεφάλαιο 2 της πτυχιακής εργασίας θα εξετάσει σε βάθος το μοντέλο ασύγχρονου προγραμματισμού που αποτελεί τον πυρήνα του Node.js. Αρχικά, γίνεται μια εισαγωγή στο Node.js, αναλύοντας την αρχιτεκτονική και τα βασικά χαρακτηριστικά του. Στόχος είναι να κατανοήσουμε πώς το Node.js εκμεταλλεύεται το JavaScript runtime και το event-driven μοντέλο για να υποστηρίξει την ανάπτυξη γρήγορων, επεκτάσιμων δικτυακών εφαρμογών.

Στη συνέχεια, το κεφάλαιο εμβαθύνει στις αρχές του ασύγχρονου προγραμματισμού, παρουσιάζοντας τα θεμελιώδη στοιχεία που καθιστούν δυνατή την εκτέλεση πολλαπλών λειτουργιών ταυτόχρονα χωρίς να μπλοκάρεται η λειτουργία της εφαρμογής. Εξετάζονται οι βασικές έννοιες όπως τα callbacks, τα promises, και το async/await, προσφέροντας μια ολοκληρωμένη κατανόηση του πώς λειτουργεί ο ασύγχρονος προγραμματισμός στο Node.js.

Το κεφάλαιο συνεχίζει με την ανάλυση των μηχανισμών χειρισμού συμβάντων (event-driven programming), οι οποίοι αποτελούν τον ακρογωνιαίο λίθο της απόδοσης και ευελιξίας του Node.js. Εδώ, περιγράφεται πώς το Node.js διαχειρίζεται τα συμβάντα και πώς αυτό επιτρέπει την ανάπτυξη εφαρμογών που μπορούν να ανταποκριθούν άμεσα σε εισερχόμενες αιτήσεις ή δεδομένα.

Επίσης, γίνεται διερεύνηση της χρήσης του Node.js σε διάφορες πλατφόρμες, από ενσωματωμένα συστήματα μέχρι cloud υποδομές. Αυτή η ενότητα δείχνει πώς η φορητότητα και η ευελιξία του Node.js το καθιστούν ιδανικό για διαφορετικά περιβάλλοντα ανάπτυξης.

Τέλος, το κεφάλαιο ολοκληρώνεται με την παρουσίαση των πλεονεκτημάτων και μειονεκτημάτων του ασύγχρονου προγραμματισμού στο Node.js. Αναλύονται οι επιπτώσεις της ασύγχρονης προσέγγισης στην απόδοση και την ευχρηστία του κώδικα, καθώς και οι προκλήσεις που ενδέχεται να προκύψουν κατά την ανάπτυξη εφαρμογών. Προτείνονται επίσης πιθανές βελτιώσεις για το Node.js, βασισμένες σε σύγχρονες έρευνες και τεχνολογικές εξελίξεις.

## 2.1. Εισαγωγή στο Node.js

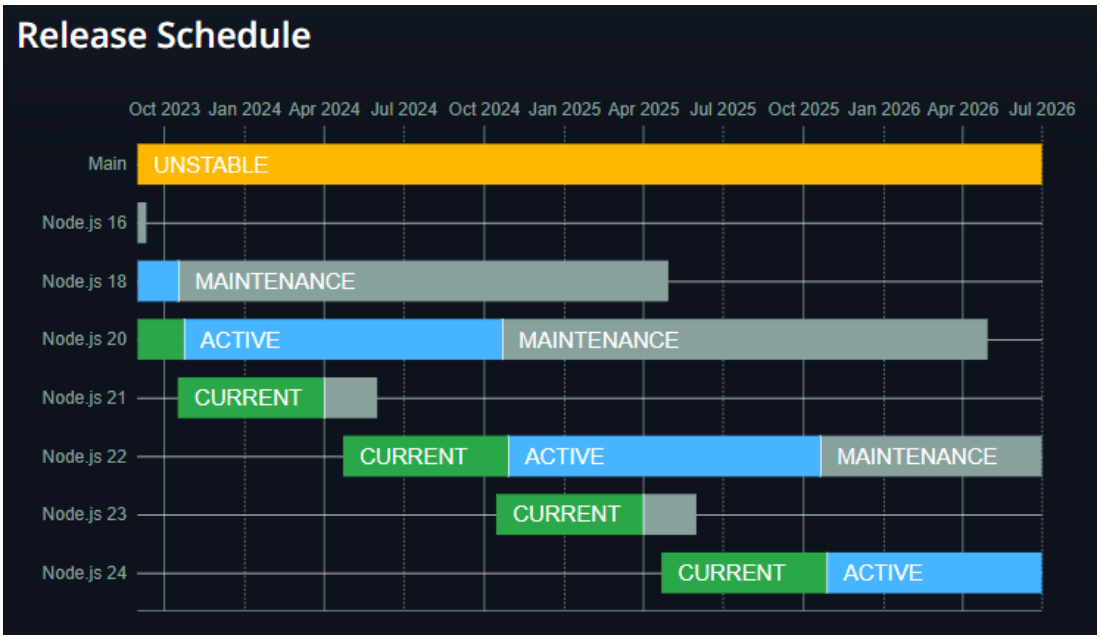
Το Node.js είναι ένα περιβάλλον εκτέλεσης JavaScript που επιτρέπει την ανάπτυξη server-side εφαρμογών χρησιμοποιώντας τη γλώσσα προγραμματισμού JavaScript. Αναπτύχθηκε από τον Ryan Dahl το 2009, με στόχο την παροχή ενός πιο αποδοτικού τρόπου διαχείρισης των ταυτόχρονων συνδέσεων σε έναν διακομιστή, ειδικά σε εφαρμογές που απαιτούν υψηλή απόδοση και ευελιξία. Χρησιμοποιεί τη μηχανή V8 της Google, η οποία μεταφράζει τον κώδικα JavaScript σε γλώσσα μηχανής, προσφέροντας ταχύτητα και αποδοτικότητα. Αυτή η αρχιτεκτονική επιτρέπει στον Node.js να διαχειρίζεται πολλές ταυτόχρονες συνδέσεις με χαμηλή καθυστέρηση, καθιστώντας τον ιδανικό για ανάπτυξη εφαρμογών που απαιτούν υψηλή ταυτόχρονη απόδοση, όπως εφαρμογές δικτύου και συστήματα πραγματικού χρόνου.

Ένα από τα κύρια χαρακτηριστικά του Node.js είναι η ασύγχρονη, event-driven αρχιτεκτονική του, η οποία επιτρέπει την εκτέλεση λειτουργιών εισόδου-εξόδου χωρίς να εμποδίζεται η κύρια ροή του προγράμματος. Αυτή η προσέγγιση εξασφαλίζει ότι οι λειτουργίες εισόδου-εξόδου, όπως η ανάγνωση αρχείων ή η επικοινωνία με βάσεις δεδομένων, δεν μπλοκάρουν την εκτέλεση άλλων τμημάτων του προγράμματος. Αντίθετα, οι λειτουργίες αυτές εκτελούνται στο παρασκήνιο και, μόλις ολοκληρωθούν, πυροδοτούν ένα event που επιτρέπει την επεξεργασία του αποτελέσματος. Η ασύγχρονη φύση του Node.js επιτρέπει στους προγραμματιστές να δημιουργούν πιο αποδοτικές και κλιμακούμενες εφαρμογές σε σύγκριση με τις παραδοσιακές server-side πλατφόρμες.

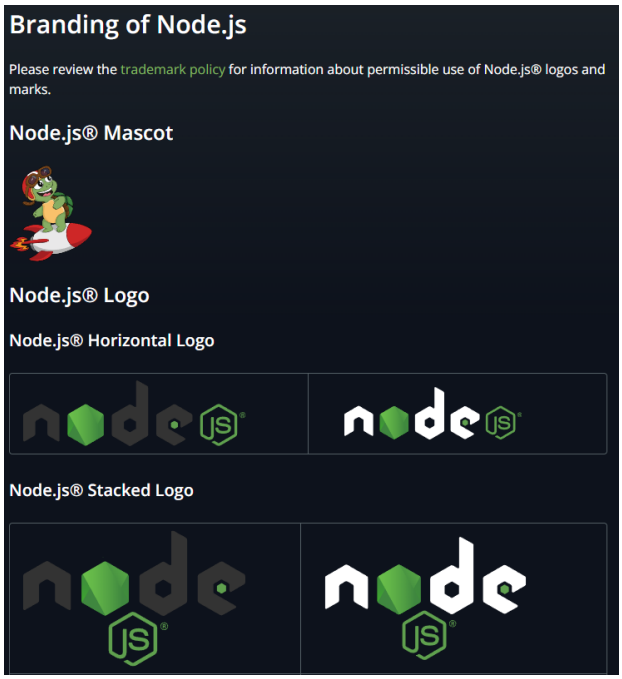
Ένα άλλο σημαντικό πλεονέκτημα του Node.js είναι η ευελιξία του να χρησιμοποιείται σε διαφορετικές πλατφόρμες και περιβάλλοντα. Από την αρχή, το framework Node.js σχεδιάστηκε για να είναι ελαφρύ και φορητό, επιτρέποντας την ανάπτυξή του σε διάφορα περιβάλλοντα, όπως προσωπικοί υπολογιστές, ενσωματωμένα συστήματα και cloud υποδομές. Αυτό καθιστά το Node.js κατάλληλο για ένα ευρύ φάσμα εφαρμογών, από μικρές, single-page εφαρμογές έως μεγάλες, κατανεμημένες εφαρμογές. Επιπλέον, η μεγάλη και ενεργή κοινότητα προγραμματιστών που υποστηρίζει το Node.js έχει δημιουργήσει έναν εκτενή κατάλογο πακέτων και βιβλιοθηκών, διαθέσιμων μέσω του npm (Node Package Manager), που επιτρέπουν την εύκολη προσθήκη λειτουργικότητας στις εφαρμογές.

Επίσης, η μοντελοποίηση του Node.js είναι προσανατολισμένη στη χρήση του JavaScript παντού, τόσο στο frontend όσο και στο backend. Αυτό επιτρέπει στους προγραμματιστές να χρησιμοποιούν την ίδια γλώσσα και εργαλεία σε όλο το φάσμα ανάπτυξης μιας εφαρμογής, από το client-side μέχρι το server-side. Αυτός το ενοποιημένο προγραμματιστικό μοντέλο μειώνει την πολυπλοκότητα και επιτρέπει μια πιο συνεκτική και αποδοτική διαδικασία ανάπτυξης. Με τη χρήση του JavaScript σε όλη τη στοίβα, οι ομάδες ανάπτυξης μπορούν να εστιάσουν σε μία μόνο γλώσσα προγραμματισμού, μειώνοντας έτσι το χρόνο εκμάθησης και την ανάγκη για διαχείριση πολλαπλών γλωσσών.

Τέλος, το Node.js έχει κερδίσει δημοτικότητα λόγω της εξαιρετικής του απόδοσης και της κοινότητας που το υποστηρίζει. Πολλές μεγάλες εταιρείες, όπως η Netflix, η LinkedIn και η PayPal, έχουν υιοθετήσει το Node.js για τις βασικές τους εφαρμογές, εκμεταλλευόμενες την ταχύτητα, την ευελιξία και την επεκτασιμότητά του. Οι εφαρμογές που αναπτύσσονται με το Node.js μπορούν να χειριστούν εκατομμύρια ταυτόχρονους χρήστες με χαμηλό κόστος συντήρησης και ανάπτυξης. Αυτό έχει καταστήσει το Node.js μία από τις πιο δημοφιλείς επιλογές για την ανάπτυξη μοντέρνων, αποδοτικών και κλιμακούμενων διαδικτυακών εφαρμογών.




**Εικόνα 2.1** Απεικόνιση του προγράμματος των εκδόσεων του Node.js framework ανά μήνα




**Εικόνα 2.2** Απεικόνιση του logo του framework

## 2.2. Αρχές του Ασύγχρονου Προγραμματισμού

Ο ασύγχρονος προγραμματισμός αποτελεί μια θεμελιώδη προσέγγιση στην ανάπτυξη λογισμικού, ιδιαίτερα σε περιβάλλοντα όπου η διαχείριση πολλαπλών εισόδων και εξόδων γίνεται ταυτόχρονα. Στον παραδοσιακό, συγχρονικό προγραμματισμό, οι λειτουργίες εκτελούνται σειριακά, με κάθε λειτουργία να αναμένει την ολοκλήρωση της προηγούμενης προτού ξεκινήσει. Αυτή η προσέγγιση μπορεί να οδηγήσει σε καθυστερήσεις, ειδικά όταν η διαδικασία περιλαμβάνει ενέργειες όπως η ανάγνωση αρχείων, η επικοινωνία με βάσεις δεδομένων ή η πρόσβαση σε διαδικτυακές υπηρεσίες. Ο ασύγχρονος προγραμματισμός λύνει αυτό το πρόβλημα επιτρέποντας τη μη μπλοκαρισμένη εκτέλεση λειτουργιών. Όταν μια ασύγχρονη λειτουργία καλείται, δεν εμποδίζει την εκτέλεση άλλων λειτουργιών. Αντίθετα, η λειτουργία αυτή συνεχίζει στο παρασκήνιο και, μόλις ολοκληρωθεί, πυροδοτεί ένα event ή καλεί μια καθορισμένη συνάρτηση (callback), που χειρίζεται το αποτέλεσμα της λειτουργίας [1].

Μια από τις βασικές αρχές του ασύγχρονου προγραμματισμού είναι η χρήση των callbacks, δηλαδή συναρτήσεων που καλούνται αφού ολοκληρωθεί μια ασύγχρονη λειτουργία. Το Node.js, που βασίζεται σε ένα event-driven μοντέλο, χρησιμοποιεί αυτήν την προσέγγιση για να επιτρέπει την ταυτόχρονη επεξεργασία πολλαπλών αιτημάτων χωρίς να μπλοκάρει την κύρια ροή του προγράμματος. Αν και οι συναρτήσεις callback είναι εξαιρετικά αποδοτικές, η εκτεταμένη χρήση τους μπορεί να οδηγήσει σε περίπλοκο κώδικα, γνωστό ως "callback hell", όπου ο κώδικας γίνεται δύσκολα αναγνώσιμος και συντηρήσιμος λόγω των πολλαπλών επιπέδων αναμονής για την ολοκλήρωση ασύγχρονων λειτουργιών. Για την αντιμετώπιση αυτού του ζητήματος, έχουν αναπτυχθεί εναλλακτικές προσεγγίσεις όπως τα Promises και οι async/await συναρτήσεις, οι οποίες απλοποιούν τη διαχείριση ασύγχρονων λειτουργιών και καθιστούν τον κώδικα πιο γραμμικό και ευανάγνωστο.

Τα Promises είναι μια πιο σύγχρονη μέθοδος διαχείρισης ασύγχρονων λειτουργιών, που εισήχθη στο πρότυπο ECMAScript 2015. Ένα Promise αντιπροσωπεύει την τελική ολοκλήρωση (ή αποτυχία) μιας ασύγχρονης λειτουργίας και την τιμή που αυτή θα επιστρέψει στο μέλλον. Η χρήση των Promises επιτρέπει στους προγραμματιστές να συνδυάζουν ασύγχρονες λειτουργίες με τρόπο που αποφεύγει την πολυπλοκότητα των callback chains. Τα Promises διαθέτουν τις μεθόδους .then() και .catch() που επιτρέπουν τον καθορισμό ενεργειών που θα εκτελεστούν όταν το Promise ολοκληρωθεί επιτυχώς ή αποτύχει, αντίστοιχα. Επιπλέον, οι JavaScript Promises μπορούν να συνδυαστούν και να αλυσοδεθούν, διευκολύνοντας τη σύνθεση πολλών ασύγχρονων ενεργειών σε μία λογική ακολουθία.

Η εισαγωγή των async/await στην ECMAScript 2017, έφερε μια επαναστατική αλλαγή στην διαχείριση ασύγχρονου κώδικα. Οι async συναρτήσεις επιστρέφουν πάντα ένα Promise και επιτρέπουν τη χρήση της εντολής await μέσα στον κώδικα τους, η οποία αναμένει την ολοκλήρωση του Promise χωρίς να μπλοκάρει την εκτέλεση άλλων λειτουργιών. Αυτή η προσέγγιση καθιστά τον ασύγχρονο κώδικα πιο γραμμικό και ευανάγνωστο, προσφέροντας ένα στυλ προγραμματισμού που είναι πιο κοντά στον παραδοσιακό συγχρονικό προγραμματισμό. Η χρήση του async/await έχει μειώσει σημαντικά την πολυπλοκότητα στη διαχείριση ασύγχρονων λειτουργιών και έχει γίνει μια από τις προτιμώμενες μεθόδους για την ανάπτυξη σύγχρονων JavaScript εφαρμογών.

Εκτός από την JavaScript, ο ασύγχρονος προγραμματισμός είναι ένα χαρακτηριστικό που υπάρχει σε πολλές άλλες γλώσσες και περιβάλλοντα προγραμματισμού. Ωστόσο, η προσέγγιση του Node.js, που

επικεντρώνεται στη μη μπλοκαρισμένη εκτέλεση λειτουργιών και την αποδοτική διαχείριση των εισόδων και εξόδων, τον καθιστά ιδανικό για την ανάπτυξη διακομιστών υψηλής απόδοσης και εφαρμογών πραγματικού χρόνου. Η αυξημένη χρήση του Node.js στην ανάπτυξη σύγχρονων διαδικτυακών εφαρμογών αποδεικνύει την ισχύ και την αποδοτικότητα του ασύγχρονου προγραμματισμού, ειδικά όταν εφαρμόζεται σε κατανεμημένα συστήματα και περιβάλλοντα cloud. Η κατανόηση των αρχών του ασύγχρονου προγραμματισμού και η ορθή εφαρμογή τους είναι κρίσιμη για την ανάπτυξη αποδοτικών και κλιμακούμενων εφαρμογών, ιδιαίτερα σε έναν κόσμο που απαιτεί συνεχώς αυξημένη απόδοση και ταυτόχρονη επεξεργασία πολλαπλών αιτημάτων [2].

CJS MJS

```
1  const { createServer } = require('node:http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
```

JavaScript Copy to clipboard

**Εικόνα 2.3** Απεικόνιση ενός ασύγχρονου προγράμματος εκτέλεσης JavaScript με βάση τα συμβάντα. Στο ακόλουθο παράδειγμα «hello world», πολλές συνδέσεις μπορούν να αντιμετωπιστούν ταυτόχρονα. Σε κάθε σύνδεση, η επανάκληση πυροδοτείται, αλλά αν δεν υπάρχει εργασία που πρέπει *[3]* να γίνει, το Node.js θα αδρανήσει.

## 2.3. Μηχανισμοί Χειρισμού Συμβάντων (Event-driven Programming)

Ο προγραμματισμός που βασίζεται στα συμβάντα (event-driven programming) αποτελεί έναν από τους πιο θεμελιώδεις και διαδεδομένα προγραμματιστικά υποδείγματα, ιδιαίτερα για την ανάπτυξη συστημάτων που απαιτούν απόκριση σε εξωτερικά ερεθίσματα σε πραγματικό χρόνο. Στην καρδιά του event-driven προγραμματισμού βρίσκεται η ιδέα ότι η ροή ενός προγράμματος καθορίζεται από συμβάντα, όπως κλικ του χρήστη, εισαγωγή δεδομένων, ή ακόμα και μηνύματα που φτάνουν σε έναν διακομιστή. Το σύστημα αναμένει την εμφάνιση τέτοιων συμβάντων και, όταν αυτά εμφανιστούν, ενεργοποιεί συγκεκριμένους χειριστές (event handlers) που είναι υπεύθυνοι για την επεξεργασία τους. Αυτή η προσέγγιση καθιστά τα

προγράμματα πιο δυναμικά και ευέλικτα, καθώς μπορούν να προσαρμόζονται και να ανταποκρίνονται σε διαφορετικές καταστάσεις και ανάγκες των χρηστών.

Ένα από τα κύρια πλεονεκτήματα του event-driven προγραμματισμού είναι η ικανότητά του να διαχειρίζεται ταυτόχρονα πολλά γεγονότα χωρίς να χρειάζεται η εκτέλεση αυστηρά σειριακών διαδικασιών. Αυτό καθίσταται ιδιαίτερα χρήσιμο σε περιβάλλοντα όπου ο χρόνος απόκρισης είναι κρίσιμος, όπως στις διαδικτυακές εφαρμογές και στις εφαρμογές πραγματικού χρόνου. Σε αντίθεση με τον παραδοσιακό προγραμματισμό, όπου οι εντολές εκτελούνται με σειρά και κάθε ενέργεια περιμένει την ολοκλήρωση της προηγούμενης, ο event-driven προγραμματισμός επιτρέπει την εκτέλεση πολλαπλών συμβάντων ανεξάρτητα και ασύγχρονα. Αυτός ο μηχανισμός καθιστά δυνατή την αποτελεσματική διαχείριση εισόδων και εξόδων, τη βελτίωση της απόδοσης, και την αύξηση της ταυτόχρονης εκτέλεσης πολλών εργασιών [4].

Η υλοποίηση ενός event-driven μοντέλου απαιτεί την ύπαρξη ενός μηχανισμού για την αναγνώριση, την καταγραφή και την προώθηση των συμβάντων. Στο Node.js, αυτός ο μηχανισμός βασίζεται στον ενσωματωμένο EventEmitter, μια βιβλιοθήκη που παρέχει τις απαραίτητες λειτουργίες για τη δημιουργία και τον χειρισμό συμβάντων. Με τον EventEmitter, οι προγραμματιστές μπορούν να δημιουργούν προσαρμοσμένα συμβάντα, να προσθέτουν ή να αφαιρούν χειριστές για αυτά τα συμβάντα, και να καθορίζουν τη συμπεριφορά της εφαρμογής όταν τα συμβάντα αυτά πυροδοτούνται. Επιπλέον, ο EventEmitter επιτρέπει την εκπομπή και τη μετάδοση μηνυμάτων μεταξύ διαφορετικών τμημάτων μιας εφαρμογής, διευκολύνοντας έτσι την επικοινωνία μεταξύ των διαφόρων λειτουργιών και ενοτήτων του συστήματος. Αυτός ο μηχανισμός είναι κρίσιμος για την ανάπτυξη πολύπλοκων, διαδραστικών εφαρμογών που πρέπει να ανταποκρίνονται σε μια σειρά από διαφορετικά γεγονότα [5].

Η χρήση του event-driven προγραμματισμού δεν περιορίζεται μόνο στις διαδικτυακές εφαρμογές. Στην ανάπτυξη λογισμικού, το μοντέλο αυτό χρησιμοποιείται ευρέως και σε άλλους τομείς, όπως οι εφαρμογές GUI (Graphical User Interface), όπου οι χρήστες αλληλεπιδρούν με το λογισμικό μέσω συμβάντων, όπως το πάτημα πλήκτρων ή το κλικ σε κουμπιά. Επίσης, στον τομέα των ενσωματωμένων συστημάτων, το event-driven μοντέλο είναι ιδιαίτερα χρήσιμο για την παρακολούθηση και τον έλεγχο συσκευών που λειτουργούν σε πραγματικό χρόνο. Με την ενσωμάτωση αισθητήρων και ενεργοποιητών, τα ενσωματωμένα συστήματα μπορούν να ανταποκρίνονται άμεσα σε αλλαγές στο περιβάλλον, ενεργοποιώντας συμβάντα που καθοδηγούν την επεξεργασία και την αντίδραση του συστήματος.

Τέλος, ο event-driven προγραμματισμός είναι μια προσέγγιση που συμβάλλει στη δημιουργία πιο αποκεντρωμένων και αποκρίσιμων συστημάτων. Καθώς τα συστήματα γίνονται όλο και πιο πολύπλοκα και αποκεντρωμένα, η ανάγκη για μοντέλα που μπορούν να διαχειριστούν ταυτόχρονα και με ασφάλεια την επεξεργασία πολλών διαφορετικών συμβάντων αυξάνεται. Ο event-driven προγραμματισμός όχι μόνο καλύπτει αυτή την ανάγκη, αλλά προσφέρει και μια ευέλικτη βάση για τη δημιουργία κλιμακούμενων συστημάτων που μπορούν να ανταποκριθούν στις σύγχρονες απαιτήσεις. Η αποτελεσματική εφαρμογή αυτού του μοντέλου μπορεί να βελτιώσει σημαντικά την απόδοση, την αποδοτικότητα και την εμπειρία χρήστη σε μια ευρεία γκάμα εφαρμογών, από μικρές εφαρμογές έως μεγάλες κατανεμημένες πλατφόρμες [6].

## 2.4. Χρήση του Node.js σε Διαφορετικές Πλατφόρμες

Το Node.js είναι ένα εξαιρετικά ευέλικτο περιβάλλον εκτέλεσης JavaScript, το οποίο μπορεί να χρησιμοποιηθεί σε διάφορες πλατφόρμες, από ενσωματωμένα συστήματα έως μεγάλες υποδομές cloud. Η ευελιξία αυτή οφείλεται στην αρχιτεκτονική του Node.js, που βασίζεται στη μη μπλοκαριστική (non-blocking) λειτουργία εισόδου/εξόδου και στη χρήση του event-driven προγραμματισμού. Στις πλατφόρμες cloud, όπως το Amazon Web Services (AWS) και το Microsoft Azure, το Node.js είναι ιδιαίτερα δημοφιλές για την ανάπτυξη κλιμακούμενων μικροϋπηρεσιών και API, χάρη στην ικανότητά του να χειρίζεται μεγάλο όγκο αιτημάτων ταυτόχρονα με χαμηλό κόστος σε πόρους. Οι υποδομές cloud προσφέρουν τη δυνατότητα αυτόματης κλιμάκωσης και ευελιξίας, που συνδυάζονται ιδανικά με τις δυνατότητες του Node.js, καθιστώντας το ιδανικό για την ανάπτυξη εφαρμογών που απαιτούν υψηλή διαθεσιμότητα και αξιοπιστία.

Στις πλατφόρμες ενσωματωμένων συστημάτων, όπως το Raspberry Pi, το Node.js βρίσκει επίσης εξαιρετική εφαρμογή. Τα ενσωματωμένα συστήματα, που συχνά διαθέτουν περιορισμένους πόρους σε επεξεργαστική ισχύ και μνήμη, επωφελούνται από την αποδοτική διαχείριση πόρων που προσφέρει το Node.js. Οι προγραμματιστές μπορούν να χρησιμοποιούν το Node.js για τη δημιουργία εφαρμογών που αλληλεπιδρούν με το υλικό του συστήματος, όπως αισθητήρες και ενεργοποιητές, μέσω πρωτοκόλλων όπως το GPIO, I2C και SPI. Η υποστήριξη για Node.js στο Raspberry Pi έχει οδηγήσει στη δημιουργία πολλών έργων που αφορούν το IoT (Internet of Things), όπου οι συσκευές μπορούν να επικοινωνούν μεταξύ τους ή με το cloud, προσφέροντας λειτουργικότητα όπως η παρακολούθηση περιβάλλοντος και ο απομακρυσμένος έλεγχος συστημάτων [3].

Η χρήση του Node.js στην ανάπτυξη διαδικτυακών εφαρμογών είναι άλλη μια ισχυρή πτυχή της πλατφόρμας. Τα σύγχρονα περιβάλλοντα ανάπτυξης διαδικτυακών εφαρμογών απαιτούν ταχύτητα και απόκριση σε πραγματικό χρόνο, ιδιότητες που είναι απόλυτα ενσωματωμένες στη φιλοσοφία του Node.js. Χάρη στο μη μπλοκαριστικό του μοντέλο και τη χρήση του V8 JavaScript engine της Google, το Node.js επιτρέπει την ανάπτυξη διαδραστικών ιστοσελίδων και εφαρμογών με ταχύτατη απόκριση, ακόμη και υπό υψηλό φορτίο χρηστών. Εφαρμογές όπως το chat σε πραγματικό χρόνο, οι συνεργατικές πλατφόρμες και οι εφαρμογές streaming πολυμέσων, χρησιμοποιούν το Node.js για την παροχή εμπειριών χρήστη υψηλής ποιότητας. Επίσης, η ενσωμάτωση του Node.js με τεχνολογίες όπως το WebSockets διευκολύνει την υλοποίηση λειτουργιών που απαιτούν συνεχή και αμφίδρομη επικοινωνία μεταξύ του διακομιστή και των πελατών.

Η πλατφόρμα Node.js έχει επίσης εισχωρήσει στις παραδοσιακές επιχειρηματικές εφαρμογές και στις εφαρμογές γραφείου, ειδικά μέσω εργαλείων όπως το Electron. Το Electron είναι ένα πλαίσιο που επιτρέπει τη δημιουργία επιτραπέζιων εφαρμογών χρησιμοποιώντας web τεχνολογίες, με το Node.js να αποτελεί τη βάση για την αλληλεπίδραση με το σύστημα αρχείων, τα συστήματα δικτύου, και άλλες λειτουργίες χαμηλού επιπέδου. Μέσω του Electron, το Node.js έχει καταστεί βασικό εργαλείο για την ανάπτυξη επιτραπέζιων εφαρμογών που είναι ταυτόχρονα ισχυρές και εύκολα διανεμήσιμες σε πολλαπλές πλατφόρμες, συμπεριλαμβανομένων των Windows, macOS και Linux. Παραδείγματα εφαρμογών που έχουν αναπτυχθεί με Electron και Node.js περιλαμβάνουν το Visual Studio Code και το Slack, εφαρμογές

που χρησιμοποιούνται ευρέως παγκοσμίως και αποδεικνύουν την ισχύ και την ευελιξία αυτής της τεχνολογίας [3].

Τέλος, το Node.js χρησιμοποιείται και στον τομέα της ανάπτυξης κινητών εφαρμογών. Μέσω της χρήσης εργαλείων όπως το React Native, το Node.js επιτρέπει τη δημιουργία υβριδικών εφαρμογών που λειτουργούν τόσο σε iOS όσο και σε Android, χρησιμοποιώντας μια ενιαία βάση κώδικα. Η αποδοτική διαχείριση της εισόδου/εξόδου, η δυνατότητα επαναχρησιμοποίησης κώδικα και η ευκολία στη δημιουργία API backend υπηρεσιών καθιστούν το Node.js ιδανικό για τον σχεδιασμό και την υλοποίηση εφαρμογών που απαιτούν ταχύτητα, ασφάλεια και συνδεσιμότητα. Ο συνδυασμός της ισχύος του Node.js με τις δυνατότητες των mobile πλατφορμών επιτρέπει στους προγραμματιστές να αναπτύσσουν εφαρμογές που εκμεταλλεύονται στο έπακρο τις δυνατότητες των κινητών συσκευών, ενώ διατηρούν τη βέλτιστη εμπειρία χρήστη.

## 2.5. Πλεονεκτήματα και Μειονεκτήματα του Ασύγχρονου Προγραμματισμού

**Πλεονεκτήματα του Ασύγχρονου Προγραμματισμού**

Ο ασύγχρονος προγραμματισμός προσφέρει σημαντικά πλεονεκτήματα, ιδιαίτερα σε περιβάλλοντα που απαιτούν υψηλή απόδοση και κλιμάκωση. Ένα από τα πιο σημαντικά πλεονεκτήματα είναι η ικανότητα διαχείρισης πολλαπλών εργασιών ταυτόχρονα χωρίς να εμποδίζεται η εκτέλεση του προγράμματος. Στο Node.js, αυτό επιτυγχάνεται μέσω του event loop, το οποίο επιτρέπει στο σύστημα να εκτελεί άλλες ενέργειες ενώ αναμένει την ολοκλήρωση μιας ασύγχρονης λειτουργίας, όπως η ανάκτηση δεδομένων από μια βάση ή η αποστολή ενός αιτήματος σε έναν διακομιστή. Αυτό το χαρακτηριστικό βελτιώνει την απόδοση, καθώς ο επεξεργαστής δεν μένει αδρανής κατά τη διάρκεια αναμονής, και έτσι οι εφαρμογές μπορούν να εξυπηρετούν περισσότερα αιτήματα ανά δευτερόλεπτο σε σύγκριση με άλλα μοντέλα προγραμματισμού.

Ένα άλλο σημαντικό πλεονέκτημα είναι η μείωση της πολυπλοκότητας στη διαχείριση των πόρων του συστήματος. Ο ασύγχρονος προγραμματισμός, λόγω της φύσης του να μην μπλοκάρει την εκτέλεση κατά τη διάρκεια των εισόδων/εξόδων (I/O operations), επιτρέπει την καλύτερη χρήση της διαθέσιμης μνήμης και της CPU. Ειδικά σε εφαρμογές που επεξεργάζονται μεγάλο όγκο δεδομένων ή σε περιβάλλοντα με περιορισμένους πόρους, η αποδοτική διαχείριση των I/O επιτυγχάνεται πιο αποτελεσματικά με το ασύγχρονο μοντέλο. Επιπλέον, η χρήση μηχανισμών όπως τα Promises και οι async/await εντολές στο Node.js βελτιώνουν περαιτέρω την ευανάγνωστη κωδικοποίηση και τη διατήρηση κώδικα, καθιστώντας τον πιο ευανάγνωστο και ευκολότερο στη συντήρηση.

Το ασύγχρονο μοντέλο προγραμματισμού ενισχύει επίσης τη συνολική εμπειρία χρήστη σε εφαρμογές που απαιτούν γρήγορες αντιδράσεις, όπως οι διαδραστικές ιστοσελίδες και οι εφαρμογές πραγματικού χρόνου.

Οι χρήστες δεν χρειάζεται να περιμένουν για μεγάλες περιόδους όταν το σύστημα εκτελεί λειτουργίες στο παρασκήνιο, καθώς αυτές δεν επηρεάζουν άμεσα την απόκριση του συστήματος. Οι εφαρμογές που είναι κατασκευασμένες με ασύγχρονο προγραμματισμό είναι συχνά πιο γρήγορες και αποκρίσιμες, βελτιώνοντας την ικανοποίηση των χρηστών και ενισχύοντας την αλληλεπίδρασή τους με το λογισμικό.

**Μειονεκτήματα του Ασύγχρονου Προγραμματισμού**

Παρά τα προφανή πλεονεκτήματα, ο ασύγχρονος προγραμματισμός συνοδεύεται από μια σειρά προκλήσεων και μειονεκτημάτων. Ένα από τα κυριότερα είναι η πολυπλοκότητα στην κατανόηση και τη διαχείριση του κώδικα. Η ασύγχρονη φύση του προγραμματισμού μπορεί να οδηγήσει σε κώδικα που είναι δύσκολο να παρακολουθηθεί, ειδικά όταν ο κώδικας περιέχει πολλά nested callbacks, μια κατάσταση γνωστή ως "callback hell". Αυτό καθιστά τη συντήρηση και την αντιμετώπιση προβλημάτων πιο δύσκολη, ειδικά σε μεγάλες εφαρμογές, καθώς η λογική της ροής του προγράμματος δεν είναι πάντα άμεσα εμφανής. Παρά την εισαγωγή των Promises και της async/await, η οποία μειώνει αυτό το πρόβλημα, η ασύγχρονη κωδικοποίηση μπορεί ακόμα να περιπλέξει την ανάπτυξη, απαιτώντας από τους προγραμματιστές να έχουν βαθιά κατανόηση των μηχανισμών διαχείρισης της ροής του προγράμματος.

Ένα άλλο σοβαρό μειονέκτημα είναι το ζήτημα της διαχείρισης σφαλμάτων (error handling) σε ασύγχρονα περιβάλλοντα. Σε παραδοσιακό, συγχρονισμένο προγραμματισμό, τα σφάλματα μπορούν να εντοπιστούν και να αντιμετωπιστούν σε μια ενιαία ροή εκτέλεσης, αλλά σε ασύγχρονα περιβάλλοντα, τα σφάλματα μπορεί να εμφανιστούν σε διαφορετικές χρονικές στιγμές και σε διαφορετικά μέρη του προγράμματος. Αυτό μπορεί να δυσκολέψει τον εντοπισμό της πηγής του σφάλματος και την επίλυσή του. Επιπλέον, ο κώδικας μπορεί να διακοπεί σε απρόβλεπτα σημεία, καθιστώντας την ανάλυση και την αντιμετώπιση προβλημάτων πιο περίπλοκη, ειδικά σε εφαρμογές που περιλαμβάνουν πολλαπλές ασύγχρονες κλήσεις και διασυνδέσεις.

Ένα ακόμα μειονέκτημα είναι η πιθανή επιβάρυνση στην απόδοση, όταν ο ασύγχρονος προγραμματισμός δεν χρησιμοποιείται σωστά. Αν και το μοντέλο αυτό αποσκοπεί στην αύξηση της απόδοσης, σε περιπτώσεις όπου οι ασύγχρονες λειτουργίες χρησιμοποιούνται χωρίς προσεκτικό σχεδιασμό, μπορεί να οδηγήσει σε "race conditions" και άλλες παγίδες που επηρεάζουν αρνητικά την αποδοτικότητα της εφαρμογής. Επιπλέον, η υπερβολική εξάρτηση από εξωτερικές κλήσεις μπορεί να αυξήσει τον χρόνο απόκρισης μιας εφαρμογής, όταν οι κλήσεις αυτές δεν διαχειρίζονται σωστά. Αυτές οι προκλήσεις απαιτούν από τους προγραμματιστές να σχεδιάζουν με προσοχή την αρχιτεκτονική των εφαρμογών, προκειμένου να αποφύγουν αυτά τα προβλήματα.

**Προτάσεις Βελτίωσης του Node.js**

Για τη βελτίωση του Node.js και τη μείωση των μειονεκτημάτων του ασύγχρονου προγραμματισμού, μια σειρά από προσεγγίσεις και βελτιώσεις μπορούν να υιοθετηθούν. Μία από αυτές είναι η ενίσχυση των εργαλείων και των βιβλιοθηκών για την καλύτερη διαχείριση των ασύγχρονων λειτουργιών και τη μείωση της πολυπλοκότητας. Η χρήση βιβλιοθηκών όπως το Async.js, που παρέχει διάφορα εργαλεία για τη διαχείριση ασύγχρονων λειτουργιών, μπορεί να βελτιώσει σημαντικά τη διαχείριση της ροής του

προγράμματος. Επιπλέον, η χρήση προτύπων σχεδίασης (design patterns) όπως το Promise.all και το Promise.race μπορεί να βοηθήσει στην αποτελεσματική διαχείριση πολλαπλών ασύγχρονων εργασιών ταυτόχρονα, ενώ παράλληλα διατηρείται η αναγνωσιμότητα και η ευκολία στη συντήρηση του κώδικα.

Άλλες προτάσεις βελτίωσης αφορούν την ενσωμάτωση καλύτερων μηχανισμών για την αντιμετώπιση σφαλμάτων σε ασύγχρονες εφαρμογές. Η ανάπτυξη βελτιωμένων εργαλείων παρακολούθησης και εντοπισμού σφαλμάτων, που μπορούν να εντοπίζουν σφάλματα σε πραγματικό χρόνο και να παρέχουν άμεσα πληροφορίες για τη θέση και τη φύση του σφάλματος, είναι κρίσιμη για την αύξηση της αξιοπιστίας των εφαρμογών. Επιπλέον, η προώθηση της χρήσης των async/await μεθόδων μπορεί να απλοποιήσει τον κώδικα και να μειώσει την εμφάνιση του callback hell, βελτιώνοντας τη συνολική εμπειρία προγραμματισμού στο Node.js.

# Κεφάλαιο 3: Παρουσίαση του Node-RED και των Δυνατοτήτων του

Το Κεφάλαιο 3 της πτυχιακής εργασίας εστιάζει στην ανάλυση του Node-RED, μιας πλατφόρμας που βασίζεται σε Node.js και προσφέρει ένα γραφικό περιβάλλον για τη δημιουργία ροών δεδομένων και την αυτοματοποίηση συστημάτων. Αρχικά, παρουσιάζεται μια εισαγωγή στο Node-RED, περιγράφοντας την εξέλιξή του, τη φιλοσοφία πίσω από τον σχεδιασμό του, και τους λόγους που το καθιστούν ένα από τα πιο δημοφιλή εργαλεία για την ανάπτυξη εφαρμογών IoT και αυτοματισμών. Η εισαγωγή αυτή θέτει τις βάσεις για την κατανόηση των επόμενων τμημάτων του κεφαλαίου, που εξετάζουν λεπτομερέστερα τις δυνατότητες και τις εφαρμογές του Node-RED.

Στη συνέχεια, το κεφάλαιο εμβαθύνει στις αρχές λειτουργίας του Node-RED και στη διαχείριση ροών (flow-based programming). Εδώ, αναλύεται πώς το Node-RED υιοθετεί την προσέγγιση του προγραμματισμού που βασίζεται σε ροές, επιτρέποντας στους χρήστες να συνδέουν συσκευές, APIs, και υπηρεσίες μέσα από ένα διαισθητικό περιβάλλον. Αυτό το τμήμα εξηγεί τη σημασία της διαχείρισης των ροών και πώς αυτή η προσέγγιση διευκολύνει την ανάπτυξη εφαρμογών με σύνθετη λογική, χωρίς την ανάγκη γραφής κώδικα σε παραδοσιακές γλώσσες προγραμματισμού.

Τέλος, στο κεφάλαιο αυτό θα εξεταστεί το οικοσύστημα του Node-RED, περιγράφοντας τις δυνατότητες, τα χαρακτηριστικά και τα παραδείγματα εφαρμογών του. Αναλύεται το Node-RED ecosystem, το οποίο περιλαμβάνει πλήθος από προσθέτα (nodes) που επεκτείνουν τις δυνατότητες της πλατφόρμας και επιτρέπουν την ενσωμάτωση με άλλες τεχνολογίες και υπηρεσίες. Επιπλέον, παρατίθενται παραδείγματα πραγματικών εφαρμογών που χρησιμοποιούν το Node-RED, αναδεικνύοντας την πρακτική του αξία σε διάφορους τομείς, από το IoT και τις έξυπνες πόλεις μέχρι τη βιομηχανία και την εκπαίδευση. Αυτή η παρουσίαση βοηθά στην κατανόηση του πώς το Node-RED μπορεί να προσαρμοστεί σε διαφορετικά περιβάλλοντα και να συμβάλει στην επίλυση πολύπλοκων προβλημάτων αυτοματοποίησης και ολοκλήρωσης συστημάτων.

## 3.1. Εισαγωγή στο Node-RED

Το Node-RED είναι ένα εργαλείο ανάπτυξης ανοιχτού κώδικα, το οποίο σχεδιάστηκε από την IBM το 2013 με σκοπό να διευκολύνει τη δημιουργία εφαρμογών που συνδέουν υλικό, προγραμματιστικές διεπαφές (APIs) και διαδικτυακές υπηρεσίες. Η πλατφόρμα αυτή βασίζεται στην ιδέα του flow-based programming, όπου οι χρήστες μπορούν να δημιουργήσουν ροές εργασιών συνδέοντας μεταξύ τους προκαθορισμένους κόμβους (nodes) με τη βοήθεια ενός γραφικού περιβάλλοντος [7]. Το Node-RED διακρίνεται για την ευκολία χρήσης του, καθώς επιτρέπει την ανάπτυξη πολύπλοκων εφαρμογών χωρίς την ανάγκη εκτεταμένης γνώσης προγραμματισμού, κάτι που το καθιστά ιδανικό εργαλείο για αρχάριους προγραμματιστές αλλά και για επαγγελματίες που επιδιώκουν γρήγορη ανάπτυξη πρωτοτύπων και

εφαρμογών IoT. Το περιβάλλον του Node-RED βασίζεται σε ένα drag-and-drop interface, όπου οι χρήστες μπορούν εύκολα να δημιουργούν και να συνδέουν κόμβους για την επεξεργασία δεδομένων, την επικοινωνία με εξωτερικές υπηρεσίες, και τον έλεγχο συσκευών υλικού [8].

Ένα από τα βασικά πλεονεκτήματα του Node-RED είναι η ευελιξία του στην ενσωμάτωση και επικοινωνία με διάφορες τεχνολογίες και πλατφόρμες. Χάρη στο γεγονός ότι είναι κατασκευασμένο πάνω από το Node.js, μπορεί να αξιοποιήσει πλήρως τη δύναμη και την επεκτασιμότητα του JavaScript οικοσυστήματος, ενώ ταυτόχρονα παρέχει πρόσβαση σε μια μεγάλη ποικιλία έτοιμων κόμβων, οι οποίοι μπορούν να εγκατασταθούν εύκολα από το Node-RED library. Οι χρήστες μπορούν να επεκτείνουν τη λειτουργικότητα της πλατφόρμας δημιουργώντας και ενσωματώνοντας δικούς τους κόμβους, χρησιμοποιώντας είτε JavaScript είτε άλλες γλώσσες προγραμματισμού. Το Node-RED υποστηρίζει πλήρως τη χρήση και τη διαχείριση των περισσότερων δημοφιλών APIs και πρωτοκόλλων όπως HTTP, MQTT, WebSockets και πολλά άλλα, καθιστώντας το ένα ισχυρό εργαλείο για την ανάπτυξη ολοκληρωμένων λύσεων στον τομέα του IoT και της αυτοματοποίησης [9].

Η δημοφιλία του Node-RED έχει αυξηθεί ραγδαία τα τελευταία χρόνια, καθώς οι δυνατότητες και η ευκολία χρήσης του έχουν κερδίσει την εμπιστοσύνη ενός ευρέος φάσματος χρηστών, από ερασιτέχνες μέχρι μεγάλες επιχειρήσεις. Η πλατφόρμα έχει υιοθετηθεί σε πολλές βιομηχανίες, όπως η ενέργεια, η γεωργία, η υγεία και η αυτοκινητοβιομηχανία, όπου οι απαιτήσεις για γρήγορη ανάπτυξη και διασύνδεση συστημάτων είναι υψηλές. Επιπλέον, το Node-RED έχει εξελιχθεί σε ένα βασικό εργαλείο για την ανάπτυξη εφαρμογών που σχετίζονται με το Διαδίκτυο των Πραγμάτων (IoT), καθώς επιτρέπει την εύκολη σύνδεση και διαχείριση αισθητήρων, actuators και άλλων συσκευών μέσω διαφόρων πρωτοκόλλων επικοινωνίας. Η κοινότητα του Node-RED είναι εξαιρετικά ενεργή, προσφέροντας συνεχή υποστήριξη, νέα πρόσθετα και εκτενή τεκμηρίωση για τους χρήστες του [8].

Παρά την αρχική του σχεδίαση για την υποστήριξη IoT εφαρμογών, το Node-RED έχει επεκταθεί και σε άλλους τομείς χρήσης, όπως η ανάπτυξη εφαρμογών cloud και microservices. Μέσω της ενσωμάτωσης με πλατφόρμες όπως το AWS, το Azure και το IBM Cloud, το Node-RED επιτρέπει στους προγραμματιστές να δημιουργούν και να διαχειρίζονται πολύπλοκες cloud-based εφαρμογές, εκμεταλλευόμενο την ευκολία του flow-based programming. Επιπλέον, οι δυνατότητες του Node-RED για επεξεργασία δεδομένων σε πραγματικό χρόνο το καθιστούν ιδανικό για την ανάπτυξη εφαρμογών που απαιτούν γρήγορη απόκριση και ευελιξία στην επεξεργασία μεγάλων όγκων δεδομένων [8].

Τέλος, ένα από τα σημαντικότερα πλεονεκτήματα του Node-RED είναι η κοινότητα και το οικοσύστημα που έχει αναπτυχθεί γύρω από αυτό. Η πλατφόρμα υποστηρίζεται ενεργά από την κοινότητα ανοιχτού κώδικα, με συνεχή βελτιώσεις και προσθήκες που διατίθενται μέσω του Node-RED library. Οι χρήστες μπορούν να μοιράζονται τις δικές τους δημιουργίες και ροές, διευκολύνοντας έτσι την επαναχρησιμοποίηση κώδικα και την ταχύτερη ανάπτυξη λύσεων. Η υποστήριξη από την κοινότητα περιλαμβάνει επίσης εκπαιδευτικά προγράμματα, σεμινάρια και εκδηλώσεις που συμβάλλουν στην εκμάθηση και την κατανόηση των δυνατοτήτων του Node-RED, κάνοντάς το ένα πολύτιμο εργαλείο για προγραμματιστές σε όλο τον κόσμο [7].

## 3.2. Αρχές Λειτουργίας και Διαχείριση Ροών (Flow-Based Programming)

Το Flow-Based Programming (FBP) είναι ένα υπόδειγμα προγραμματισμού που βασίζεται στη διαχείριση ροών δεδομένων μέσα από διασυνδεδεμένες διαδικασίες, ή "κόμβους", που ανταλλάσσουν πληροφορίες μέσω προκαθορισμένων συνδέσεων. Στο πλαίσιο του Node-RED, η ροή αυτή υλοποιείται μέσα από το γραφικό περιβάλλον, όπου οι χρήστες μπορούν να σύρουν και να συνδέσουν κόμβους για τη δημιουργία αλυσίδων δεδομένων και διαδικασιών. Κάθε κόμβος αντιπροσωπεύει μια λειτουργία, είτε είναι μια απλή αριθμητική πράξη είτε μια περίπλοκη αλληλεπίδραση με εξωτερικές υπηρεσίες. Η βασική αρχή του FBP είναι ότι οι ροές αυτές μπορούν να δημιουργηθούν και να αναδιαμορφωθούν εύκολα, προσφέροντας ευελιξία και επαναχρησιμοποίηση των διαδικασιών χωρίς να απαιτείται αλλαγή στον κώδικα. Ένα σημαντικό χαρακτηριστικό του FBP είναι η ανεξαρτησία των κόμβων, που επιτρέπει τη διαχείριση τους ως ανεξάρτητα "μαύρα κουτιά", ενισχύοντας την αρθρωτότητα και τη διαλειτουργικότητα του συστήματος [10].

Η διαχείριση των ροών στο Node-RED ακολουθεί την αρχή ότι κάθε κόμβος έχει συγκεκριμένη είσοδο και έξοδο, και κάθε ροή αποτελείται από μια αλληλουχία αυτών των κόμβων που εκτελούν συγκεκριμένες ενέργειες πάνω στα δεδομένα που διακινούνται. Οι ροές αυτές μπορεί να είναι είτε σύγχρονες είτε ασύγχρονες, ανάλογα με τις ανάγκες της εφαρμογής, με τους κόμβους να ανταλλάσσουν μηνύματα που περιέχουν δεδομένα ή οδηγίες για την εκτέλεση ενεργειών. Αυτή η προσέγγιση επιτρέπει την ανάπτυξη σύνθετων συστημάτων με χαμηλό επίπεδο κώδικα, καθιστώντας το Node-RED ιδανικό για εφαρμογές όπου η ταχύτητα ανάπτυξης και η απλότητα είναι κρίσιμοι παράγοντες. Η δυνατότητα επεξεργασίας δεδομένων σε πραγματικό χρόνο ενισχύει περαιτέρω την αποτελεσματικότητα των ροών, καθιστώντας το Node-RED μια ισχυρή επιλογή για εφαρμογές IoT και συστήματα αυτοματοποίησης [10].

Μια από τις σημαντικότερες λειτουργίες του FBP στο Node-RED είναι η δυνατότητα της πλατφόρμας να ενσωματώνει διαφορετικούς τύπους δεδομένων και να τους διαχειρίζεται μέσα από μια ενιαία ροή. Αυτό επιτυγχάνεται μέσω της χρήσης "ειδικών κόμβων" που είναι προσαρμοσμένοι για την αλληλεπίδραση με συγκεκριμένες πλατφόρμες ή υπηρεσίες, όπως βάσεις δεδομένων, APIs ή συστήματα αισθητήρων. Αυτοί οι κόμβοι λειτουργούν ως διαμεσολαβητές μεταξύ των διαφορετικών τμημάτων του συστήματος, επιτρέποντας την απρόσκοπτη μεταφορά και επεξεργασία δεδομένων. Το αποτέλεσμα είναι ένα αρθρωτό σύστημα που μπορεί να επεκταθεί και να προσαρμοστεί εύκολα σε νέες απαιτήσεις, χωρίς να επηρεάζεται η λειτουργικότητα των υφιστάμενων κόμβων ή ροών. Η χρήση του FBP στο Node-RED όχι μόνο απλοποιεί την ανάπτυξη σύνθετων εφαρμογών αλλά και προάγει την επαναχρησιμοποίηση κώδικα, διευκολύνοντας έτσι τη διαχείριση της πολυπλοκότητας σε μεγάλα συστήματα.

Η αρχιτεκτονική του Node-RED βασίζεται σε μια "event-driven" προσέγγιση, όπου οι ροές εκτελούνται ως αντίδραση σε γεγονότα (events) που πυροδοτούνται από εξωτερικές ή εσωτερικές πηγές. Αυτή η προσέγγιση επιτρέπει στο σύστημα να λειτουργεί με υψηλή αποδοτικότητα, καθώς οι κόμβοι εκτελούν τις εργασίες τους μόνο όταν αυτό είναι απαραίτητο, μειώνοντας έτσι την κατανάλωση πόρων. Επιπλέον, η δυνατότητα ενσωμάτωσης μηχανισμών όπως η διαχείριση συνεδριών και η ταυτοποίηση χρηστών μέσω

JSON Web Tokens (JWT) καθιστά το Node-RED κατάλληλο για εφαρμογές όπου η ασφάλεια και η διαχείριση χρηστών είναι κρίσιμες. Αυτή η αρχιτεκτονική επιτρέπει επίσης την εύκολη κλιμάκωση των εφαρμογών, καθώς οι ροές μπορούν να εκτελούνται παράλληλα, αξιοποιώντας πλήρως τις δυνατότητες του υποκείμενου υλικού.

Τέλος, το Node-RED και το FBP έχουν αναγνωριστεί σε ακαδημαϊκές μελέτες για την ευελιξία και την αποδοτικότητα τους στην ανάπτυξη IoT εφαρμογών και άλλων σύνθετων συστημάτων. Η χρήση του FBP στο Node-RED μειώνει σημαντικά τον χρόνο ανάπτυξης και δοκιμής εφαρμογών, ενώ παράλληλα διευκολύνει τη συντήρηση και την επαναχρησιμοποίηση κώδικα. Αυτό είναι ιδιαίτερα σημαντικό σε περιβάλλοντα όπου οι απαιτήσεις αλλάζουν συχνά και η ταχύτητα προσαρμογής είναι κρίσιμη. Η έρευνα δείχνει επίσης ότι το Node-RED μπορεί να ενσωματωθεί αποτελεσματικά σε περιβάλλοντα cloud, επιτρέποντας την ανάπτυξη και διαχείριση κατανεμημένων συστημάτων με ελάχιστο κόστος και πολυπλοκότητα [11].

## 3.3. Δυνατότητες και Χαρακτηριστικά του Node-RED

Το Node-RED είναι ένα πανίσχυρο εργαλείο ροών δεδομένων που διακρίνεται για την ευκολία χρήσης και την ευελιξία του. Μία από τις κυριότερες δυνατότητες του είναι το γραφικό περιβάλλον προγραμματισμού που παρέχει, το οποίο επιτρέπει τη δημιουργία, σύνθεση και διαχείριση ροών χωρίς την ανάγκη για παραδοσιακή γραφή κώδικα. Οι ροές, οι οποίες απαρτίζονται από κόμβους (nodes) που αντιπροσωπεύουν διαφορετικές λειτουργίες ή ενέργειες, μπορούν εύκολα να συνδεθούν μεταξύ τους με τη μέθοδο του drag-and-drop. Αυτή η προσέγγιση μειώνει το χρόνο ανάπτυξης και δοκιμής εφαρμογών, καθώς επιτρέπει στους χρήστες να βλέπουν άμεσα το αποτέλεσμα των αλλαγών τους και να επαναπροσδιορίζουν τις ροές τους σε πραγματικό χρόνο. Επιπλέον, η ενσωματωμένη δυνατότητα επαναχρησιμοποίησης ροών επιτρέπει στους χρήστες να αποθηκεύουν και να μοιράζονται ροές, κάτι που είναι ιδιαίτερα χρήσιμο σε μεγάλα έργα ή όταν εργάζονται σε ομάδες [8].

Μια άλλη σημαντική δυνατότητα του Node-RED είναι η ενσωμάτωση μιας μεγάλης ποικιλίας έτοιμων κόμβων, οι οποίοι επιτρέπουν την αλληλεπίδραση με διαφορετικές πλατφόρμες, APIs και υπηρεσίες. Αυτοί οι κόμβοι, οι οποίοι περιλαμβάνουν λειτουργίες όπως HTTP requests, σύνδεση με βάσεις δεδομένων, χειρισμό WebSocket και επικοινωνία με συσκευές IoT, καθιστούν το Node-RED εξαιρετικά προσαρμόσιμο. Οι χρήστες μπορούν να επεκτείνουν τις δυνατότητες του Node-RED με τη δημιουργία των δικών τους προσαρμοσμένων κόμβων ή με την εγκατάσταση πρόσθετων (plugins) από την ευρύτερη κοινότητα του Node-RED [11]. Αυτή η επεκτασιμότητα και η δυνατότητα σύνδεσης με πληθώρα εξωτερικών πηγών και υπηρεσιών ενισχύουν την ικανότητα του Node-RED να διαχειρίζεται πολύπλοκα συστήματα και να διευκολύνει την ενσωμάτωση διαφορετικών τεχνολογιών.

Το Node-RED επίσης διακρίνεται για την ισχυρή του υποστήριξη σε ασύγχρονες λειτουργίες και τη διαχείριση συμβάντων (event-driven programming). Κάθε κόμβος μπορεί να λειτουργεί ανεξάρτητα και να

αντιδρά σε γεγονότα, επιτρέποντας έτσι την ταυτόχρονη εκτέλεση πολλαπλών ροών χωρίς να προκαλείται συμφόρηση στο σύστημα. Αυτή η δυνατότητα είναι ιδιαίτερα χρήσιμη σε εφαρμογές που απαιτούν την επεξεργασία δεδομένων σε πραγματικό χρόνο, όπως σε περιβάλλοντα IoT ή σε εφαρμογές που βασίζονται σε συνεχείς ροές δεδομένων από αισθητήρες και άλλες πηγές. Η ικανότητα αυτή ενισχύεται περαιτέρω από την ενσωμάτωση του Node.js, που προσφέρει ένα εξαιρετικά αποδοτικό περιβάλλον εκτέλεσης JavaScript, το οποίο έχει σχεδιαστεί για να διαχειρίζεται πολλές ασύγχρονες διεργασίες με ελάχιστη κατανάλωση πόρων.

Επιπλέον, το Node-RED προσφέρει ισχυρές δυνατότητες διαχείρισης δεδομένων και ελέγχου ροών. Οι χρήστες μπορούν να χρησιμοποιούν λογικούς κόμβους για να ελέγχουν την κατεύθυνση της ροής δεδομένων, να εφαρμόζουν συνθήκες και να διαχειρίζονται την εκτέλεση συγκεκριμένων ενεργειών ανάλογα με τα δεδομένα που λαμβάνονται. Οι δυνατότητες αυτές ενισχύουν την ευελιξία και τη διαλειτουργικότητα των συστημάτων που δημιουργούνται με το Node-RED, επιτρέποντας τη δημιουργία σύνθετων διαδικασιών που μπορούν να προσαρμοστούν εύκολα στις ανάγκες της εφαρμογής. Επιπλέον, το Node-RED υποστηρίζει τη διαχείριση σφαλμάτων και την καταγραφή δεδομένων (logging), επιτρέποντας έτσι τη διαγνωστική ανάλυση και τη βελτιστοποίηση της λειτουργίας των ροών [12].

Τέλος, το Node-RED διαθέτει μια ενεργή κοινότητα χρηστών και προγραμματιστών, που συμβάλλει στη συνεχή ανάπτυξη και βελτίωση του εργαλείου. Μέσω της κοινότητας, οι χρήστες μπορούν να έχουν πρόσβαση σε μια μεγάλη βιβλιοθήκη πρόσθετων, να μοιράζονται ροές και να λαμβάνουν υποστήριξη για την ανάπτυξη των εφαρμογών τους. Η συνεργασία αυτή ενισχύει τη δυναμική του Node-RED και καθιστά το εργαλείο ένα από τα πιο δημοφιλή και ευρέως χρησιμοποιούμενα στη διαχείριση ροών δεδομένων και την ανάπτυξη εφαρμογών IoT.

## 3.4. Ανάλυση του Οικοσυστήματος του Node-RED

Το οικοσύστημα του Node-RED αποτελείται από μια σειρά εργαλείων, βιβλιοθηκών, και κοινοτικών συνεισφορών που επεκτείνουν τις βασικές δυνατότητες της πλατφόρμας, καθιστώντας την κατάλληλη για την ανάπτυξη ποικίλων εφαρμογών. Ένα από τα βασικότερα χαρακτηριστικά του οικοσυστήματος είναι η μεγάλη συλλογή διαθέσιμων κόμβων (nodes), οι οποίοι μπορούν να εγκατασταθούν και να χρησιμοποιηθούν μέσα από τη βιβλιοθήκη του Node-RED. Αυτοί οι κόμβοι καλύπτουν ένα ευρύ φάσμα λειτουργιών, όπως διασύνδεση με εξωτερικές βάσεις δεδομένων, επικοινωνία με APIs, επεξεργασία δεδομένων σε πραγματικό χρόνο και σύνδεση με διάφορες πλατφόρμες IoT. Ο πλούτος των κόμβων, που συνεχώς επεκτείνεται χάρη στις συνεισφορές της κοινότητας, επιτρέπει στους χρήστες να διαμορφώσουν τις ροές τους με βάση τις συγκεκριμένες απαιτήσεις του έργου τους, προσφέροντας έτσι μεγάλη ευελιξία και δυνατότητα προσαρμογής.

Μια σημαντική διάσταση του οικοσυστήματος του Node-RED είναι η ισχυρή και ενεργή κοινότητα χρηστών και προγραμματιστών που έχει αναπτυχθεί γύρω από την πλατφόρμα. Η κοινότητα αυτή συμβάλλει στη συνεχή βελτίωση του εργαλείου, προσφέροντας νέους κόμβους, πρόσθετα, και βελτιώσεις στον πυρήνα της πλατφόρμας. Επιπλέον, μέσω της κοινότητας, οι χρήστες έχουν πρόσβαση σε πληθώρα εκπαιδευτικών πόρων, όπως τεκμηρίωση, tutorials, και παραδείγματα χρήσης που διευκολύνουν την εκμάθηση και την υλοποίηση εφαρμογών με το Node-RED. Η συνεργασία αυτή προωθεί την καινοτομία και τη διάδοση του Node-RED, καθιστώντας το ένα από τα πιο διαδεδομένα εργαλεία στην ανάπτυξη εφαρμογών για IoT και διασυνδεδεμένα συστήματα [11].

Το Node-RED ecosystem δεν περιορίζεται μόνο στο software, αλλά επεκτείνεται και στο hardware. Η πλατφόρμα έχει σχεδιαστεί ώστε να είναι συμβατή με πολλές δημοφιλείς συσκευές και πλατφόρμες hardware, όπως το Raspberry Pi, Arduino, και διάφορα άλλα ενσωματωμένα συστήματα. Η δυνατότητα αυτή επιτρέπει την εύκολη ενσωμάτωση του Node-RED σε περιβάλλοντα IoT, όπου συσκευές και αισθητήρες επικοινωνούν μεταξύ τους και με το cloud. Το Node-RED παρέχει την υποδομή για την ανάπτυξη εφαρμογών που μπορούν να παρακολουθούν και να ελέγχουν αυτές τις συσκευές σε πραγματικό χρόνο, καθιστώντας το ιδανικό για βιομηχανικές εφαρμογές, έξυπνα σπίτια, και άλλες περιπτώσεις χρήσης όπου απαιτείται αλληλεπίδραση μεταξύ πολλών διαφορετικών συστημάτων [11].

Στο Node-RED ecosystem περιλαμβάνονται επίσης εργαλεία διαχείρισης και ανάπτυξης, τα οποία διευκολύνουν τη συνεργασία και την επαναχρησιμοποίηση κώδικα. Ένα από αυτά τα εργαλεία είναι το Node-RED Projects, που επιτρέπει στους χρήστες να διαχειρίζονται και να μοιράζονται τα projects τους μέσω του Git. Αυτό προωθεί την ομαδική εργασία και την ευκολία στη διαχείριση εκδόσεων, ιδιαίτερα σε μεγάλα έργα. Επίσης, η υποστήριξη για περιβάλλοντα συνεχούς ολοκλήρωσης και ανάπτυξης (CI/CD) επιτρέπει την αυτοματοποίηση της ανάπτυξης και της δοκιμής εφαρμογών, κάνοντας το Node-RED ένα εργαλείο που ενσωματώνεται ομαλά στις σύγχρονες διαδικασίες ανάπτυξης λογισμικού.

Τέλος, το Node-RED ecosystem περιλαμβάνει και εμπορικές επεκτάσεις, οι οποίες προσφέρουν πρόσθετα χαρακτηριστικά και υπηρεσίες που μπορούν να υποστηρίξουν την ανάπτυξη επαγγελματικών εφαρμογών. Πολλές εταιρείες έχουν αναπτύξει και προσφέρουν κόμβους και πρόσθετα που επεκτείνουν τις δυνατότητες του Node-RED σε συγκεκριμένες βιομηχανίες ή χρήσεις. Αυτές οι εμπορικές λύσεις συχνά παρέχουν υποστήριξη, εγγυήσεις, και πρόσθετες υπηρεσίες, που είναι κρίσιμες για επιχειρήσεις που βασίζονται στην τεχνολογία για την παροχή κρίσιμων υπηρεσιών. Το Node-RED, μέσω αυτού του εκτεταμένου οικοσυστήματος, έχει καταφέρει να εδραιωθεί ως μια αξιόπιστη και ευέλικτη λύση για την ανάπτυξη και την ενσωμάτωση συστημάτων IoT και όχι μόνο [8].

## 3.5. Παραδείγματα Εφαρμογών και Χρήσεων του Node-RED

Το Node-RED έχει βρει εφαρμογή σε ένα ευρύ φάσμα βιομηχανικών και καταναλωτικών τομέων, λόγω της ευελιξίας και της ικανότητάς του να διασυνδέει διαφορετικά συστήματα και πλατφόρμες. Μία από τις

πιο κοινές χρήσεις του Node-RED είναι στον τομέα του Διαδικτύου των Πραγμάτων (IoT), όπου χρησιμοποιείται για τη συλλογή, επεξεργασία και προβολή δεδομένων από αισθητήρες και συσκευές σε πραγματικό χρόνο. Για παράδειγμα, σε έξυπνα σπίτια, το Node-RED μπορεί να διασυνδέσει συσκευές όπως φώτα, θερμοστάτες, και συστήματα ασφαλείας, επιτρέποντας στους χρήστες να δημιουργήσουν αυτοματισμούς που βελτιώνουν την ενεργειακή αποδοτικότητα και την άνεση. Η ικανότητα του Node-RED να συνδέεται με δημοφιλείς πλατφόρμες IoT όπως το MQTT και το HTTP, το καθιστά ιδιαίτερα δημοφιλές για την υλοποίηση έξυπνων οικιακών συστημάτων.

Στον τομέα της βιομηχανίας, το Node-RED χρησιμοποιείται για τη διαχείριση και παρακολούθηση διαδικασιών παραγωγής. Σε περιβάλλοντα Industry 4.0, όπου η αυτοματοποίηση και η διασύνδεση των μηχανών είναι κρίσιμες, το Node-RED επιτρέπει τη δημιουργία προσαρμοσμένων εφαρμογών που μπορούν να συλλέγουν δεδομένα από διάφορες πηγές και να τα αναλύουν σε πραγματικό χρόνο. Ένα παράδειγμα τέτοιας χρήσης είναι η παρακολούθηση των μηχανημάτων σε ένα εργοστάσιο, όπου το Node-RED συγκεντρώνει δεδομένα όπως θερμοκρασία, ταχύτητα και κατάσταση λειτουργίας από αισθητήρες που είναι εγκατεστημένοι στις μηχανές. Αυτά τα δεδομένα μπορούν να χρησιμοποιηθούν για την πρόβλεψη πιθανών βλαβών και τη λήψη αποφάσεων για τη συντήρηση, μειώνοντας έτσι το χρόνο διακοπής λειτουργίας και αυξάνοντας την παραγωγικότητα [7].

Μία άλλη εφαρμογή του Node-RED είναι στον τομέα της υγείας, όπου η πλατφόρμα χρησιμοποιείται για την παρακολούθηση της κατάστασης των ασθενών και την παροχή απομακρυσμένης φροντίδας. Για παράδειγμα, το Node-RED μπορεί να συνδεθεί με φορητές συσκευές όπως οι αισθητήρες καρδιακού ρυθμού ή τα έξυπνα ρολόγια, συλλέγοντας δεδομένα σε πραγματικό χρόνο για την υγεία του ασθενούς. Αυτά τα δεδομένα μπορούν να αναλυθούν και να αποσταλούν σε ιατρούς για την παρακολούθηση της κατάστασης του ασθενούς ή ακόμα και για την αυτόματη ειδοποίηση σε περίπτωση ανωμαλιών. Αυτή η εφαρμογή του Node-RED είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου οι ασθενείς χρειάζονται συνεχή παρακολούθηση, αλλά δεν μπορούν να βρίσκονται συνεχώς σε ένα νοσοκομείο [13].

Το Node-RED χρησιμοποιείται επίσης στην ανάπτυξη εφαρμογών για έξυπνες πόλεις, όπου η συλλογή και ανάλυση δεδομένων σε πραγματικό χρόνο είναι απαραίτητη για τη βελτίωση της ποιότητας ζωής των κατοίκων. Για παράδειγμα, το Node-RED μπορεί να χρησιμοποιηθεί για την παρακολούθηση της κυκλοφορίας σε πραγματικό χρόνο, συγκεντρώνοντας δεδομένα από αισθητήρες που είναι εγκατεστημένοι στους δρόμους και προτείνοντας βελτιώσεις για τη ροή της κυκλοφορίας. Επιπλέον, μπορεί να χρησιμοποιηθεί για την παρακολούθηση της ποιότητας του αέρα, συλλέγοντας δεδομένα από αισθητήρες που μετρούν τη μόλυνση σε διάφορα σημεία της πόλης. Αυτά τα δεδομένα μπορούν να βοηθήσουν τις τοπικές αρχές να λάβουν αποφάσεις για τη βελτίωση της ποιότητας του αέρα και τη μείωση της ρύπανσης [14].

Τέλος, το Node-RED έχει εφαρμογές στον τομέα της εκπαίδευσης και της έρευνας, όπου χρησιμοποιείται ως εργαλείο για την εκμάθηση προγραμματισμού και για την ανάπτυξη πρωτοτύπων. Χάρη στο γραφικό του περιβάλλον, το Node-RED επιτρέπει στους εκπαιδευτικούς και τους ερευνητές να δημιουργούν και να δοκιμάζουν γρήγορα νέες ιδέες, χωρίς να απαιτούνται εκτενείς γνώσεις προγραμματισμού. Σε πανεπιστήμια και ερευνητικά κέντρα, το Node-RED χρησιμοποιείται για την ανάπτυξη πρωτοτύπων εφαρμογών IoT, όπου οι φοιτητές και οι ερευνητές μπορούν να δημιουργήσουν και να πειραματιστούν με διαφορετικές τεχνολογίες και να δουν τα αποτελέσματα σε πραγματικό χρόνο. Αυτή η δυνατότητα καθιστά

το Node-RED ένα πολύτιμο εργαλείο για την εκπαίδευση και την έρευνα σε ένα ευρύ φάσμα τεχνολογικών τομέων [13].

## 3.6. Οδηγίες Εγκατάστασης του Περιβάλλοντος Node-RED

Η εγκατάσταση του Node-RED είναι μια διαδικασία που μπορεί να πραγματοποιηθεί εύκολα σε διάφορες πλατφόρμες, όπως Linux, Windows, macOS, και ενσωματωμένα συστήματα όπως το Raspberry Pi. Στην πρώτη παράγραφο, παρουσιάζονται οι γενικές απαιτήσεις για την εγκατάσταση του Node-RED. Το Node-RED απαιτεί την ύπαρξη του Node.js στην έκδοση 14.x ή νεότερη, καθώς το περιβάλλον αυτό στηρίζεται στην πλατφόρμα Node.js για τη λειτουργία του. Είναι σημαντικό να διασφαλιστεί ότι το σύστημα στο οποίο θα εγκατασταθεί το Node-RED διαθέτει επίσης το npm (Node Package Manager), το οποίο περιλαμβάνεται αυτόματα με την εγκατάσταση του Node.js. Ο χρήστης μπορεί να επιβεβαιώσει την εγκατάσταση του Node.js και του npm εκτελώντας τις εντολές node -v και npm -v στο τερματικό [8].

Στην παρούσα παράγραφο, περιγράφεται η διαδικασία εγκατάστασης του Node-RED σε συστήματα Linux και macOS. Αφού επιβεβαιωθεί ότι το Node.js είναι εγκατεστημένο, ο χρήστης μπορεί να εγκαταστήσει το Node-RED παγκοσμίως χρησιμοποιώντας το npm. Αυτό επιτυγχάνεται με την εκτέλεση της εντολής sudo npm install -g --unsafe-perm node-red. Η παράμετρος --unsafe-perm χρησιμοποιείται για να διασφαλίσει ότι η εγκατάσταση θα ολοκληρωθεί χωρίς προβλήματα δικαιωμάτων, ειδικά όταν ο npm χρησιμοποιείται με το sudo. Μετά την ολοκλήρωση της εγκατάστασης, το Node-RED μπορεί να ξεκινήσει με την εντολή node-red στο τερματικό. Το περιβάλλον είναι πλέον έτοιμο για χρήση μέσω του προγράμματος περιήγησης στη διεύθυνση http://localhost:1880.

Η διαδικασία εγκατάστασης του Node-RED σε συστήματα Windows είναι παρόμοια με αυτή των άλλων λειτουργικών συστημάτων, με τη βασική διαφορά ότι δεν απαιτείται η χρήση του sudo. Ο χρήστης πρέπει πρώτα να ανοίξει μια γραμμή εντολών (Power shell) με δικαιώματα διαχειριστή και έπειτα να ακολουθήσει τις παρακάτω οδηγίες:

1.Αρχικά ο χρήστης θα πρέπει να ξεκινήσει την εγκατάσταση των πακέτων του Node.js από τον παρακάτω σύνδεσμο εδώ. Στη συνέχεια ο χρήστης θα πατήσει παντού next προκειμένου να εγκατασταθεί στον υπολογιστή του το Node.js .

2. Στη συνέχεια θα πρέπει ο χρήστης να εγκαταστήσει τα πακέτα του προσθέτου Node-RED

τα οποία θα εγκατασταθούν ακολουθώντας τις παρακάτω οδηγίες:

- Εκτέλεση της εντολής npm install -g --unsafe-perm node-red

στο power shell των windows προκειμένου να γίνει η εγκατάσταση του Node-RED ως global module και να προστεθεί η εντολή node-red στη διαδρομή του συστήματος του χρήστη.

3. Εκτέλεση του Node-RED

Εκτέλεση σε περιβάλλον Windows

Μόλις εγκατασταθεί, ο απλός τρόπος για να εκτελεστεί το Node-RED είναι να χρησιμοποιηθεί η εντολή node-red σε μια γραμμή εντολών: Εάν έχει εγκαταστήσει το Node-RED ως ένα παγκόσμιο πακέτο npm, μπορεί να χρησιμοποιήσετε η εντολή node-red σε οποιοδήποτε directory:

**C:>node-red**

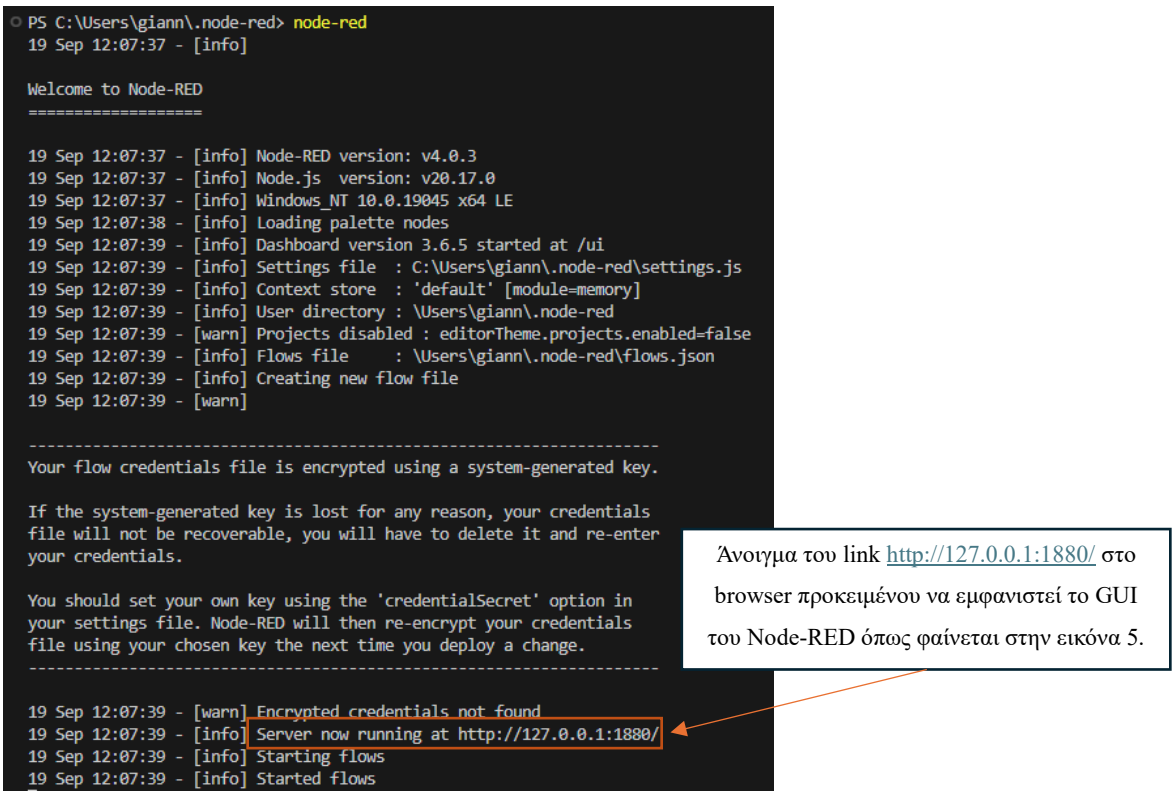
Αυτό θα δώσει το αρχείο καταγραφής του Node-RED στο τερματικό. Ο χρήστης θα πρέπει να διατηρήσει το τερματικό ανοιχτό για να συνεχίσει να εκτελεί το Node-RED.

Σημειώστε ότι η εκτέλεση του Node-RED θα δημιουργήσει ένα νέο φάκελο στο φάκελο %HOMEPATH% με το όνομα .node-red. Αυτός είναι ο φάκελος userDir, σκεφτείτε τον ως τον αρχικό φάκελο για τις ρυθμίσεις του Node-RED για τον τρέχοντα χρήστη. Συχνά θα δείτε ότι στην τεκμηρίωση αναφέρεται ως ~/.node-red. Το ~ είναι συντομογραφία για τον αρχικό φάκελο του χρήστη σε συστήματα τύπου Unix. Μπορείτε να χρησιμοποιήσετε την ίδια αναφορά αν χρησιμοποιείτε το PowerShell ως γραμμή εντολών, όπως συνιστάται. Εάν χρησιμοποιείτε το παλαιότερο κέλυφος cmd, αυτό δεν θα λειτουργήσει.

Μετά την ολοκλήρωση της εγκατάστασης, το Node-RED μπορεί να ξεκινήσει με την εντολή node-red και το περιβάλλον θα είναι διαθέσιμο στον προεπιλεγμένο web browser στη διεύθυνση http://localhost:1880.

Στην τέταρτη παράγραφο, αναλύεται η διαδικασία εγκατάστασης του Node-RED σε ένα Raspberry Pi, μια από τις δημοφιλέστερες πλατφόρμες για την ανάπτυξη έργων IoT. Το Raspberry Pi συνήθως συνοδεύεται από την έκδοση Raspberry Pi OS, η οποία βασίζεται στο Debian. Εδώ, προτείνεται η χρήση ενός ειδικού σεναρίου (script) που παρέχεται από την κοινότητα του Node-RED, το οποίο αυτοματοποιεί την εγκατάσταση. Το σενάριο μπορεί να εκτελεστεί με την εντολή bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered). Αυτό το σενάριο εγκαθιστά το Node.js, το Node-RED, και διαμορφώνει το σύστημα για να εκκινεί αυτόματα το Node-RED κατά την εκκίνηση του Raspberry Pi.

Τέλος, παρουσιάζουμε τη διαδικασία για την εκτέλεση του Node-RED σε περιβάλλοντα Docker. Το Docker επιτρέπει την εκτέλεση του Node-RED σε απομονωμένα containers, προσφέροντας ευκολία στη διαχείριση και την ανάπτυξη. Η εγκατάσταση πραγματοποιείται με τη χρήση της εντολής docker run -it -p 1880:1880 --name mynodered nodered/node-red, η οποία κατεβάζει την τελευταία έκδοση του Node-RED από το Docker Hub και τη διαμορφώνει για να τρέχει στη θύρα 1880. Αυτή η προσέγγιση είναι ιδανική για όσους επιθυμούν να διαχειρίζονται πολλαπλά instances του Node-RED ή να το ενσωματώνουν σε μεγαλύτερα έργα με χρήση κοντέινερς.




**Εικόνα 3.1** Απεικόνιση της εκκίνησης του server του Node-RED με την εντολή node-red


**Εικόνα 3.2** Απεικόνιση αρχικού GUI το οποίο χρησιμοποιείται για την δημιουργία των ροών(flows) πάνω στο Node RED framework

# Κεφάλαιο 4: Ανάλυση και Σχεδίαση της Τεχνικής Επέκτασης του Προσθέτου Node-RED-dashboard

Στο κεφάλαιο αυτό παρουσιάζεται η ανάλυση και η σχεδίαση της τεχνικής επέκτασης που αναπτύχθηκε πάνω στο Node-RED Dashboard, με στόχο την ασφαλή υποστήριξη πολλαπλών χρηστών και την πλήρη απομόνωση του περιβάλλοντος κάθε χρήστη. Ξεκινάμε από τον ορισμό του προβλήματος και τις απαιτήσεις, και προχωράμε στη σχεδιαστική πρόταση.’Ενας κεντρικός Express server με ταυτοποίηση μέσω JSON Web Tokens (JWT), εκτέλεση ξεχωριστού Node-RED instance ανά χρήστη σε δική του θύρα και έλεγχο πρόσβασης τόσο στα API όσο και στα per-user instances. Παράλληλα, περιγράφεται η ελαφριά διαχείριση συνεδρίας μέσω functionGlobalContext, καθώς και η ροή πλοήγησης (login/register). Εικόνες και πίνακες συμπληρώνουν την τεκμηρίωση της αρχιτεκτονικής και των βασικών ροών.

## 4.1. Ορισμός και Ανάλυση του Προβλήματος

Το υπάρχον πρόσθετο Node-RED-dashboard αντιμετωπίζει ένα σημαντικό πρόβλημα, καθώς όλοι οι χρήστες μοιράζονται την ίδια συνεδρία και την ίδια εσωτερική κατάσταση. Αυτό σημαίνει ότι οι αλλαγές που κάνει ένας χρήστης επηρεάζουν όλους τους άλλους χρήστες, κάτι που δεν είναι επιθυμητό σε πολλές περιπτώσεις. Για να λυθεί αυτό το πρόβλημα, είναι απαραίτητο να ενσωματωθεί ταυτοποίηση χρηστών και να υπάρχει τρόπος ώστε κάθε χρήστης να μπορεί να δουλεύει με την δική του απομονωμένη κατάσταση. Η ταυτοποίηση χρηστών θα επιτρέπει την αναγνώριση κάθε χρήστη ξεχωριστά, ενώ η εκτέλεση ξεχωριστού Node-RED instance ανά χρήστη διασφαλίζει ότι οι ροές και τα dashboards ενός χρήστη δεν επηρεάζουν και τα αντίστοιχα των υπολοίπων.

Για την ταυτοποίηση χρηστών, υλοποιήθηκε η χρήση JSON Web Tokens (JWT). Τα JWT αποτελούν έναν ασφαλή τρόπο αναγνώρισης, καθώς επιτρέπουν την υπογεγραμμένη ταυτοποίηση. Αποθηκεύονται σε cookie στον browser μετά το επιτυχές login, το οποίο ελέγχεται σε κάθε αίτημα. Έτσι εξασφαλίζεται ότι μόνο ταυτοποιημένοι χρήστες έχουν πρόσβαση στα dashboards τους, ενώ αποτρέπεται η χρήση παλιών ή άκυρων tokens.

Η διαχείριση των δεδομένων κάθε συνεδρίας υλοποιείται σε απλή μορφή. Κάθε φορά που συνδέεται ένας χρήστης , δημιουργείται ένα αντικείμενο **sessionData** στο global context του Node-RED, το οποίο περιλαμβάνει στοιχεία όπως χρόνος σύνδεσης και ένα μοναδικά αναγνωριστικό συνεδρίας. Στο global context υπάρχουν επίσης και τα προσωπικά στοιχεία του χρήστη. Τα προσωπικά αυτά στοιχεία είναι διαθέσιμα μέσω και ενός api (profile). Τα δεδομένα αυτά είναι διαθέσιμα μέσα από το dashboard. Με αυτόν τον τρόπο παρέχεται ένας βασικός μηχανισμός παρακολούθησης της εκάστοτε ενεργής συνεδρίας , χωρίς ωστόσο να γίνεται μόνιμη αποθήκευση των δεδομένων.

## 4.2. Υφιστάμενα Προβλήματα και Περιορισμοί του node-red-dashboard

Το node-red-dashboard, αν και είναι ένα ισχυρό εργαλείο για τη δημιουργία διαδραστικών διεπαφών χρήστη για εφαρμογές IoT και άλλες εφαρμογές, παρουσιάζει ορισμένα σημαντικά προβλήματα και περιορισμούς. Ένα από τα κύρια προβλήματα είναι ότι όλοι οι χρήστες μοιράζονται την ίδια κατάσταση. Αυτό σημαίνει ότι οποιαδήποτε αλλαγή γίνεται από έναν χρήστη, όπως η ενημέρωση ενός γραφήματος ή η αλλαγή μιας τιμής, επηρεάζει όλους τους άλλους χρήστες που χρησιμοποιούν το dashboard. Αυτό μπορεί να είναι ιδιαίτερα προβληματικό σε περιπτώσεις όπου πολλοί χρήστες χρειάζονται να έχουν ξεχωριστές και ανεξάρτητες προβολές των δεδομένων τους. Η έλλειψη αυτής της δυνατότητας μπορεί να περιορίσει τη χρηστικότητα του node-red-dashboard σε περιβάλλοντα όπου απαιτείται εξατομικευμένη εμπειρία χρήστη [15].

Ένας άλλος περιορισμός του node-red-dashboard είναι η έλλειψη ενσωματωμένης ταυτοποίησης χρηστών. Χωρίς τη δυνατότητα να αναγνωρίζονται και να διαχειρίζονται οι χρήστες, είναι δύσκολο να εφαρμοστούν πολιτικές ασφαλείας και ελέγχου πρόσβασης. Αυτό σημαίνει ότι οποιοσδήποτε έχει πρόσβαση στο dashboard μπορεί να δει και να αλληλεπιδράσει με όλα τα δεδομένα, κάτι που μπορεί να είναι ανεπιθύμητο σε πολλές περιπτώσεις. Η έλλειψη ταυτοποίησης χρηστών επίσης καθιστά δύσκολη την παρακολούθηση της δραστηριότητας των χρηστών και την εφαρμογή εξατομικευμένων ρυθμίσεων και προτιμήσεων.

Η διαχείριση δεδομένων συνεδρίας είναι επίσης ένας τομέας όπου το node-red-dashboard παρουσιάζει περιορισμούς. Χωρίς τη δυνατότητα να αποθηκεύονται και να διαχειρίζονται δεδομένα συνεδρίας για κάθε χρήστη ξεχωριστά, είναι δύσκολο να διατηρηθεί η συνέπεια και η ακεραιότητα των δεδομένων. Αυτό μπορεί να οδηγήσει σε προβλήματα όταν οι χρήστες αποσυνδέονται και επανασυνδέονται, καθώς μπορεί να χάσουν τις προσωρινές ρυθμίσεις και τα δεδομένα τους. Η έλλειψη αυτής της δυνατότητας μπορεί επίσης να περιορίσει τη δυνατότητα του node-red-dashboard να υποστηρίζει πιο σύνθετες και απαιτητικές εφαρμογές.

Ένα άλλο πρόβλημα είναι η περιορισμένη δυνατότητα προσαρμογής της διεπαφής χρήστη. Αν και το node-red-dashboard παρέχει μια σειρά από προκαθορισμένα widgets και στοιχεία, οι δυνατότητες προσαρμογής είναι περιορισμένες. Αυτό σημαίνει ότι οι χρήστες μπορεί να δυσκολευτούν να δημιουργήσουν διεπαφές που να ανταποκρίνονται πλήρως στις ανάγκες και τις προτιμήσεις τους. Η έλλειψη ευελιξίας στη σχεδίαση της διεπαφής μπορεί να περιορίσει τη χρηστικότητα του εργαλείου σε πιο εξειδικευμένες εφαρμογές.

Τέλος, η απόδοση του node-red-dashboard μπορεί να αποτελέσει πρόβλημα σε περιπτώσεις όπου υπάρχουν πολλοί χρήστες ή μεγάλος όγκος δεδομένων. Η απόδοση μπορεί να επηρεαστεί αρνητικά όταν το dashboard πρέπει να διαχειριστεί πολλές ταυτόχρονες συνδέσεις ή να επεξεργαστεί μεγάλα σύνολα δεδομένων σε πραγματικό χρόνο. Αυτό μπορεί να οδηγήσει σε καθυστερήσεις και προβλήματα απόκρισης, κάτι που μπορεί να είναι ιδιαίτερα προβληματικό σε εφαρμογές όπου η ταχύτητα και η αξιοπιστία είναι κρίσιμες. Η βελτίωση της απόδοσης και η διαχείριση των πόρων είναι σημαντικοί τομείς που πρέπει να ληφθούν υπόψη για την αναβάθμιση του Node-RED-Dashboard [16].

Στα επόμενα τμήματα παρουσιάζεται μία προσέγγιση που αντιμετωπίζει τα ζητήματα ταυτοποίησης και απομόνωσης, αξιοποιώντας JWT σε cookie και εκτέλεση ξεχωριστού Node-RED instance ανά χρήστη. Με

αυτόν τον τρόπο κάθε χρήστης διαθέτει το δικό του περιβάλλον εργασίας, απομονωμένο από τους υπόλοιπους.

## 4.3. Απαιτήσεις Χρηστών και Προδιαγραφές Σχεδιασμού

Οι απαιτήσεις χρηστών για το Node-RED Dashboard είναι ποικίλες και εξαρτώνται από τις συγκεκριμένες ανάγκες και τις εφαρμογές που χρησιμοποιούν. Μία από τις βασικές απαιτήσεις είναι η δυνατότητα εξατομίκευσης της διεπαφής χρήστη. Οι χρήστες θέλουν να μπορούν να προσαρμόζουν τα widgets και τα στοιχεία του dashboard ώστε να ανταποκρίνονται στις δικές τους ανάγκες και προτιμήσεις. Αυτό περιλαμβάνει τη δυνατότητα αλλαγής των χρωμάτων, των γραμματοσειρών, και της διάταξης των στοιχείων, καθώς και την αποθήκευση αυτών των ρυθμίσεων σε προσωπικό επίπεδο. Η εξατομίκευση πρέπει να γίνεται ανά χρήστη, έτσι ώστε κάθε χρήστης να έχει το δικό του απομονωμένο περιβάλλον εργασίας, χωρίς να επηρεάζει ή να επηρεάζεται από τις αλλαγές των άλλων.

**Πίνακας 4.1:** Αντιστοίχιση απαιτήσεων με την παρούσα υλοποίηση

| Απαίτηση | Συγκεκριμένη ανάγκη | Σχεδιαστική απόκριση |
|---|---|---|
| Εξατομίκευση και απομόνωση | Κάθε χρήστης να έχει δικό του UI/flows χωρίς να επηρεάζει άλλους. | Εκκίνηση per-user Node-RED instance σε μοναδικό port, προσωπικός φάκελος με δικό του settings.js και flows.json. |
| Ασφάλεια/ταυτοποίηση | Έλεγχος πρόσβασης σε API, editor,dashboard. | JWT σε httpOnly cookie. Στο κεντρικό Express προστασία API με express-jwt. Στο κάθε instance ένα admin και ένα node middleware επαληθεύουν το JWT. |
| Μία ενεργή συνεδρία ανά browser | Να μη γίνεται ταυτόχρονη σύνδεση με άλλο όνομα χρήστη στον ίδιο browser. | Έλεγχος στο /login: αν υπάρχει έγκυρο cookie άλλου χρήστη, μπλοκάρεται το νέο login |
| Συνέπεια βασικών δεδομένων συνεδρίας | Γρήγορη πρόσβαση σε στοιχεία χρήστη & session κατά τη διάρκεια της σύνδεσης. | functionGlobalContext στο per-user settings.js με user και sessionData (π.χ. loginTime, sessionId). Προβάσιμα από flows/dashboard. Εφήμερα—δεν αποθηκεύονται μόνιμα. |
| Προστασία μετά από restart | Παλιό cookie να μη δίνει πρόσβαση όταν αλλάζει το secret. | Σε κάθε εκκίνηση, παράγεται νέο secret. Κεντρικό middleware στο Express καθαρίζει άκυρα/παλιά cookies στο πρώτο αίτημα. |

| | | |
|---|---|---|
| Απόδοση & καθαρότητα κατάστασης | Να μην «μπερδεύεται» το state μεταξύ χρηστών. | Η απομόνωση μέσω ξεχωριστού port/instance εξαλείφει κοινό state στο dashboard και αποφεύγει συγκρούσεις. |
| Διαχείριση χρηστών | Ασφαλής εγγραφή/αποθήκευση στοιχείων. | MongoDB/Mongoose για μοντέλο User, bcrypt για hash κωδικού, multer για photo upload, getAvailablePort για αυτόματη ανάθεση θύρας. |

Μια άλλη σημαντική απαίτηση είναι η ασφάλεια και η ταυτοποίηση χρηστών. Οι χρήστες θέλουν να είναι βέβαιοι ότι τα δεδομένα τους είναι ασφαλή και ότι μόνο εξουσιοδοτημένα άτομα έχουν πρόσβαση στο dashboard. Αυτό σημαίνει ότι πρέπει να ενσωματωθούν μηχανισμοί ταυτοποίησης, όπως τα JSON Web Tokens (JWT), που επιτρέπουν την ασφαλή αναγνώριση των χρηστών. Στην συγκεκριμένη υλοποίηση, η ταυτοποίηση βασίζεται σε συνδυασμό MongoDB για την αποθήκευση στοιχείων χρηστών, bcrypt για την ασφαλή αποθήκευση κωδικών πρόσβασης και JWT σε cookies για την επαλήθευση κάθε αιτήματος. Με αυτόν τον τρόπο διασφαλίζεται ότι κάθε χρήστη μπορεί να έχει πρόσβαση μόνο στο δικό του Node-RED instance και στο δικό του dashboard.

Η διαχείριση δεδομένων συνεδρίας είναι επίσης μια σημαντική απαίτηση για τους χρήστες του Node-RED Dashboard. Οι χρήστες θέλουν να μπορούν να διατηρούν βασικές πληροφορίες συνεδρίας, ώστε να υπάρχει συνέπεια και ακεραιότητα στα δεδομένα τους κατά την διάρκεια της σύνδεσης. Στην παρούσα εργασία αυτό επιτυγχάνεται με την αποθήκευση δεδομένων συνεδρίας στο global context του Node-RED, όπου καταγράφονται στοιχεία όπως η ώρα σύνδεσης και ένα μοναδικό αναγνωριστικό συνεδρίας. Παράλληλα, στο global context είναι διαθέσιμα και τα προσωπικά στοιχεία του χρήστη, τα οποία είναι διαθέσιμα και μέσω του προστατευμένου API /api/profile. Έτσι παρέχεται ένας βασικός μηχανισμός παρακολούθησης της προσωρινής κατάστασης κάθε χρήστη.

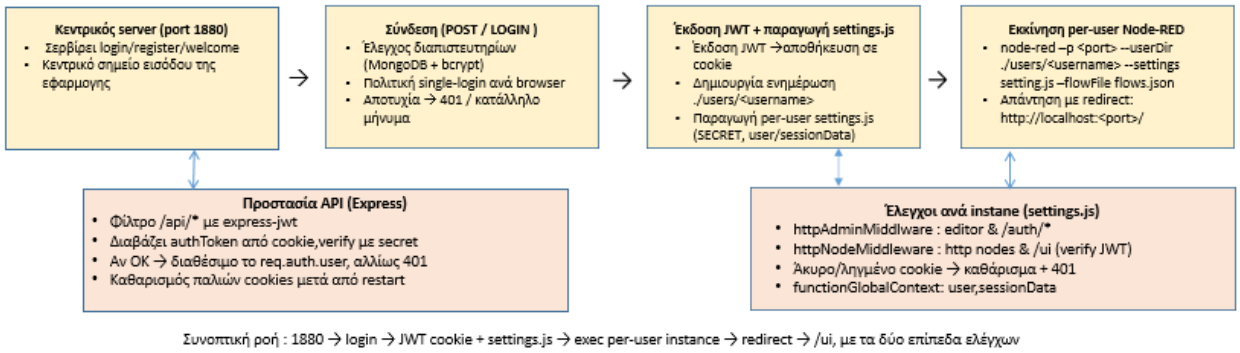
Η απόδοση και η αξιοπιστία είναι επίσης κρίσιμες απαιτήσεις για τους χρήστες του Node-RED Dashboard. Οι χρήστες θέλουν να είναι βέβαιοι ότι το σύστημα μπορεί να ανταποκριθεί με συνέπεια και χωρίς καθυστερήσεις. Στην παρούσα υλοποίηση η απόδοση διασφαλίζεται μέσα από την απομόνωση μέσω ξεχωριστού port για κάθε χρήστη. Παράλληλα, για λόγους ασφάλειας και συνέπειας επιτρέπεται μόνο μία ενεργή συνεδρία. Με αυτόν τον τρόπο τα δεδομένα και τα dashboards κάθε χρήστη παραμένουν απομονωμένα και προστατευμένα και δεν προκύπτουν συγκρούσεις συνεδριών.

Τέλος, η υποστήριξη και η τεκμηρίωση είναι σημαντικές απαιτήσεις για τους χρήστες του Node-RED Dashboard. Οι χρήστες θέλουν να έχουν πρόσβαση σε πλήρη και κατανοητή τεκμηρίωση που να εξηγεί πώς να χρησιμοποιούν το εργαλείο και να επιλύουν τυχόν προβλήματα που μπορεί να προκύψουν. Επιπλέον, η υποστήριξη από την κοινότητα και τους προγραμματιστές είναι κρίσιμη για την επίλυση προβλημάτων και την παροχή βοήθειας όταν χρειάζεται. Η ύπαρξη μιας ενεργής κοινότητας χρηστών και

προγραμματιστών μπορεί να βοηθήσει στην ανταλλαγή γνώσεων και την ανάπτυξη νέων λειτουργιών και βελτιώσεων για το εργαλείο.

## 4.4. Σχεδίαση της Τεχνικής Επέκτασης

Η σχεδίαση της τεχνικής επέκτασης του Node-RED dashboard βασίστηκε στη ανάγκη υποστήριξης πολλαπλών χρηστών με ασφαλή ταυτοποίηση και απομόνωση. Η κύρια επιδίωξη ήταν να διασφαλιστεί ότι κάθε χρήστης θα έχει το δικό του ανεξάρτητο περιβάλλον Node-RED, με προσωπικά flows και dashboards, τα οποία δεν θα είναι προσβάσιμα από άλλους χρήστες. Για τον σκοπό αυτό κρίθηκε αναγκαία η ενσωμάτωση μηχανισμών αυθεντικοποίησης, διαχείρισης συνεδριών και αυτόματης εκκίνησης ξεχωριστών Node-RED instances.




**Εικόνα 4.1** : Αρχιτεκτονική ροής

Η αρχιτεκτονική του συστήματος στηρίζεται σε έναν κεντρικό διακομιστή Express στο port 1880, ο οποίος είναι υπεύθυνος για την διαχείριση των αιτημάτων χρηστών και την εφαρμογή των πολιτικών ασφαλείας. Το Express είναι ένα ελαφρύ web framework για Node.js που απλοποιεί τη δημιουργία http εφαμοργών μέσω ευέλικτης δρομολόγησης αιτημάτων [17]. Η ταυτοποίηση των χρηστών γίνεται μέσω MongoDB, όπου αποθηκεύονται τα στοιχεία τους, και bcrypt το οποίο χρησιμοποιείται για την ασφαλή αποθήκευση κωδικών πρόσβασης. Μετά από την επιτυχή σύνδεση, δημιουργείται JSON Web Token (JWT), το οποίο αποθηκεύεται σε cookie στον browser και ελέγχεται σε κάθε αίτημα. Έτσι εξασφαλίζεται ότι μόνο ταυτοποιημένοι χρήστες έχουν πρόσβαση στο σύστημά τους.

Για την απομόνωση του περιβάλλοντος κάθε χρήστη, το σύστημα εκκινεί δυναμικά ένα νέο instance σε ξεχωριστό port μετά το login. Κατά την εγγραφή δημιουργείται αυτόματα ένας κατάλογος χρήστη που περιέχει προσωπικά αρχεία ρυθμίσεων και ροών. Στο αρχείο settings.js , το οποίο παράγεται ξεχωριστά για κάθε χρήστη, ενσωματώνονται μηχανισμοί ελέγχου JWT τόσο για τα admin endpoints όσο και για τα http node, ώστε να διασφαλίζεται ότι όλες οι λειτουργίες παραμένουν προστατευμένες.

Για την υποστήριξη βασικής διαχείρισης συνεδρίας και την πρόσβαση σε στοιχεία χρήστη μέσα από τις ροές, σε κάθε προσωπικό Node-RED instance καταχωρούνται δεδομένα στο functionGlobalContext. Εκεί, αποθηκεύονται τα βασικά στοιχεία του χρήστη καθώς και έναν αντικείμενο sessionData που με την ώρα σύνδεσης και ένα μοναδικό αναγνωριστικό συνεδρίας. Οι πληροφορίες αυτές είναι διαθέσιμες σε functions και dashboards, ενώ το προστατευμένο API /api/profile επιτρέπεται την ασφαλή ανάκτηση των προσωπικών στοιχείων.

Η παρούσα σχεδίαση επικεντρώνεται στην απομόνωση και την ασφάλεια, αποδίδοντας σε κάθε χρήστη το δικό του ανεξάρτητο Node-RED instance. Όλοι οι χρήστες έχουν κοινό ρόλο, γεγονός που απλοποιεί και ανταποκρίνεται στον στόχο της εργασίας. Ωστόσο, το σύστημα είναι επεκτάσιμο και μπορεί να υποστηρίξει διαφοροποίηση ρόλων και δικαιωμάτων, αλλά και ταυτόχρονες συνδέσεις.

Η υλοποίηση ολοκληρώνεται με τη δοκιμή και αξιολόγηση των μηχανισμών ταυτοποίησης και απομόνωσης. Οι δοκιμές εστιάζουν στην διαδικασία εγγραφής, σύνδεσης και εκκίνησης προσωπικών Node-RED instances, και της ανακατεύθυνσης των χρηστών στα δικά τους instances. Επιπλέον, αξιολογείται η λειτουργία του JWT ελέγχου και του API /api/profile. Η τεκμηρίωση της υλοποίησης περιλαμβάνει οδηγίες για την εγκατάσταση, την δημιουργία χρηστών και την χρήση της εφαρμογής, παρέχοντας στους χρήστες τα απαραίτητα εργαλεία για την αξιοποίηση του συστήματος. Τέλος, προβλέπεται η δημιουργία πακέτου εγκατάστασης, ώστε το σύστημα να μπορεί να αναπτυχθεί εύκολα σε διαφορετικά περιβάλλοντα χωρίς να απαιτείται πολύπλοκη διαδικασία ρύθμισες.

## 4.5. Χρήση JSON Web Tokens (JWT) για την Ταυτοποίηση Χρηστών

Προκειμένου να εφαρμοστεί η ταυτοποίηση των χρηστών χρειάζεται να γίνει πρώτα η χρήση των JSON Web Tokens. Στην παρούσα υλοποίηση, το token εκδίδεται κατά την σύνδεση του χρήστη, αποθηκεύεται σε cookie στον browser και χρησιμοποιείται για τις προστατευμένες λειτουργίες του συστήματος [18] .

Η επαλήθευση του JWT γίνεται σε δύο επίπεδα. Αρχικά στον κεντρικό server Express, ο οποίος ελέγχει κάθε αίτημα προς τα API endpoints μέσω του middleware που παρέχει το πακέτο express-jwt και σε κάθε προσωπικό Node-RED instance, όπου το δυναμικά παραγόμενο αρχείο settings.js περιλαμβάνει middleware για την προστασία των admin endpoints και των http nodes. Με αυτόν τον τρόπο μόνο οι ταυτοποιημένοι χρήστες έχουν πρόσβαση στα dashboards και τις ροές τους.

### 4.5.1. Προαπαιτούμενα Πακέτα

Για την υλοποίηση της ταυτοποίησης των χρηστών με JWT και την εκκίνηση ανεξάρτητων Node-RED instances απαιτούνται ορισμένα εξωτερικά πακέτα Node.js, εκτός από το ίδιο το Node-RED και το πρόσθετο Node-RED dashboard.

Συγκεκριμένα χρησιμοποιούνται τα εξής:

- **express:** για την υλοποίηση του κεντρικού server που διαχειρίζεται τα αιτήματα χρηστών.

- **mongoose:** για την αποθήκευση των χρηστών σε MongoDB.
- **bcrypt:** για την ασφαλή αποθήκευση και έλεγχο των κωδικών πρόσβασης.
- **jsonwebtoken:** για την δημιουργία και επαλήθευση των JWT.
- **express-jwt:** για τον αυτόματο έλεγχο των JWT στα API endpoints
- **cookie-parser:** για την ανάγνωση του token από το cookie του browser.
- **multer:** για την μεταφόρτωση και αποθήκευση φωτογραφίας κατά την εγγραφή.
- **open:** για το αυτόματο άνοιγμα σελίδας του login στον browser.

Η εγκατάσταση των εξωτερικών πακέτων γίνεται μέσω τερματικού με την εντολή :

```
npm install express mongoose bcrypt jsonwebtoken express-jwt cookie-parser multer open
```

Εκτός από αυτά, αξιοποιούνται και ενσωματωμένα πακέτα της Node.js, όπως http, path, fs, crypto και child_process. Τα modules αυτά δεν απαιτούν εγκατάσταση, καθώς περιλαμβάνονται στο περιβάλλον Node.js.

Η εισαγωγή όλων των παραπάνω πακέτων γίνεται στην αρχή του κώδικα με τις αντίστοιχες εντολές require, όπως φαίνεται στο ακόλουθο απόσπασμα:

```
const http = require('http');
const express = require('express');
const RED = require('node-red');
const path = require('path');
const jsonwebtoken = require('jsonwebtoken');
const mongoose = require('mongoose');
const { expressjwt: jwt } = require('express-jwt');
const multer = require('multer');
const { exec } = require("child_process");
const bcrypt = require('bcrypt');
const cookieParser = require('cookie-parser');
const fs = require('fs');
const crypto = require('crypto');
```

Με αυτόν τον τρόπο ο κώδικας προετοιμάζεται ώστε να χρησιμοποιεί τόσο τα built-in modules της Node.js όσο και τα εξωτερικά πακέτα που εγκαθίστανται μέσω npm.

### 4.5.2. Διαχείριση Χρηστών και Εγγραφή

Η διαχείριση χρηστών πραγματοποιείται μέσω MongoDB, χρησιμοποιώντας το πακέτο **mongoose.** Αρχικά, η εφαρμογή συνδέεται στη βάση δεδομένων:

```
mongoose.connect('mongodb://localhost:27017/Authentication', {
   useNewUrlParser: true,
   useUnifiedTopology: true
})
.then(() => console.log('Συνδέθηκε στη MongoDB'))
.catch(err => console.error('Σφάλμα σύνδεσης στη MongoDB:', err));
```

Στην συνέχεια ορίζεται το μοντέλο χρήστη. Το schema περιλαμβάνει τα βασικά πεδία ταυτοποίησης, προσωπικά στοιχεία καθώς και την port που ανέλαβε κατά την εγγραφή για το προσωπικό Node-RED instance:

```
const userSchema = new mongoose.Schema({
   username: { type: String, required: true, unique: true },
   password: { type: String, required: true },
   jwtToken: { type: String },
   firstname: String,
   lastname: String,
   university: String,
   email: String,
   photo: String,
   port: { type: Number, unique: true }
});
const User = mongoose.model('User', userSchema);
```

Κατά την εγγραφή ενός νέου χρήστη εκτελούνται τα παρακάτω βήματα:

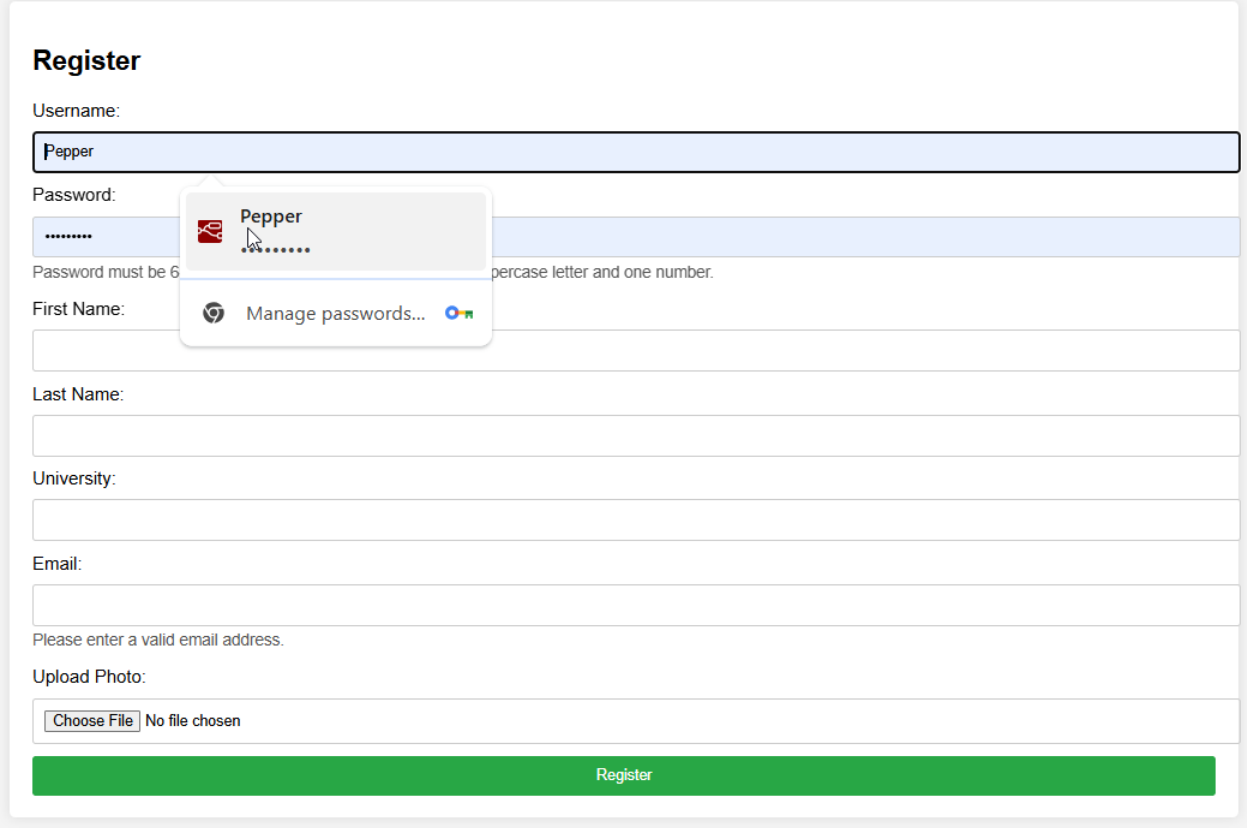
- Έλεγχος πληρότητας πεδίων και μοναδικότητας username.
- Ανάθεση διαθέσιμης θύρας μέσω της getAvailablePort() ώστε κάθε χρήστης να έχει το δικό του port.
- Hash κωδικός με bcrypt (δεν αποθηκεύονται ποτέ κωδικοί σε απλό κείμενο).
- Αποθήκευση στοιχείων στην MongoDB.
- Προαιρετική αποθήκευση φωτογραφίας μέσω multer.

- Δημιουργία προσωπικού καταλόγου χρήστη (./users/<username>) για μελλοντικά αρχεία ρυθμίσεων/ροών.

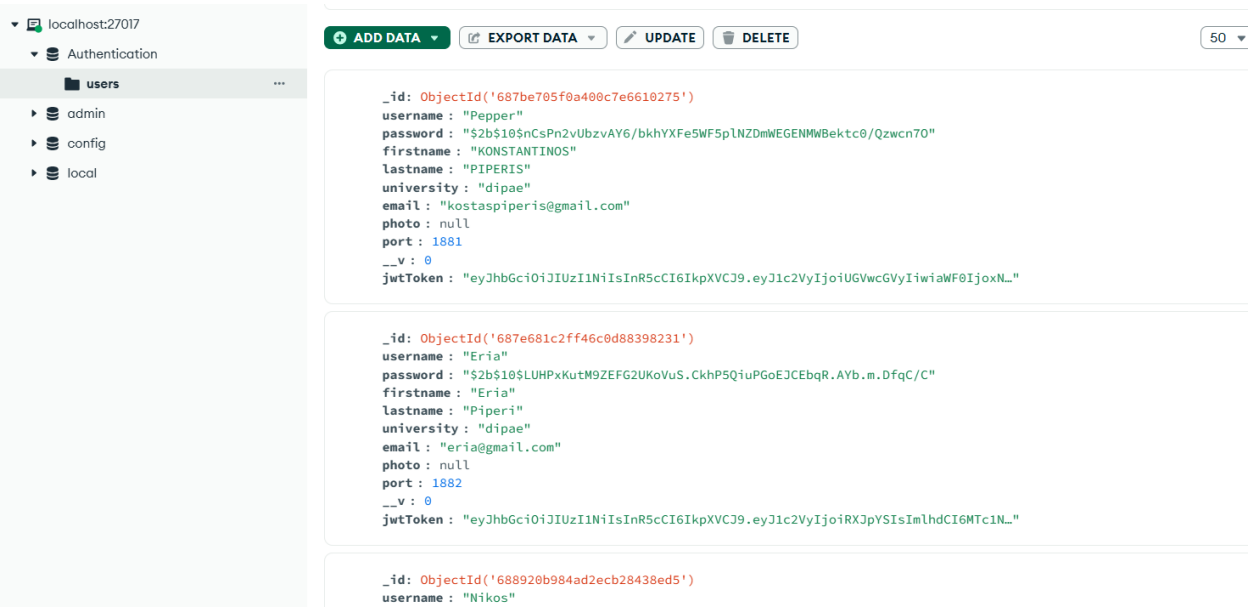
```
app.post('/register', upload.single('photo'), async (req, res) => {
  const { username, password, firstname, lastname, university, email } = req.body;
  const photoPath = req.file ? req.file.filename : null;

  if (!username || !password || !firstname || !lastname || !university || !email) {
    return res.status(400).json({ message: 'Όλα τα πεδία είναι υποχρεωτικά.' });
  }

  if (await User.findOne({ username })) {
    return res.status(400).json({ message: 'Το όνομα χρήστη υπάρχει ήδη.' });
  }

  const port = await getAvailablePort();
  const hashedPassword = await bcrypt.hash(password, 10);

  const newUser = new User({
    username,
    password: hashedPassword,
    firstname,
    lastname,
    university,
    email,
    photo: photoPath,
    port
  });

  await newUser.save();
  // Δημιουργία προσωπικού φακέλου, αν δεν υπάρχει
```

```
  const userDir = `./users/${username}`;
  if (!fs.existsSync(userDir)) {
      fs.mkdirSync(userDir, { recursive: true });
  }

  return res.json({ success: true, redirect: "/login.html" });
});
```

Στο στάδιο της εγγραφής δημιουργείται ο προσωπικός κατάλογος του χρήστη. Παράγεται όμως δυναμικά κατά την σύνδεση και περιλαμβάνει τα middleware ελέγχου JWT για το admin UI και τα http nodes, καθώς και το functionGlobalContext με τα στοιχεία χρήστη και δεδομένα συνεδρίας(βλ. 4.5.3 και 4.6).




**Εικόνα 4.2 :** Απεικόνιση της σελίδας register.html




**Εικόνα 4.3:** Αποθήκευση στοιχείων χρήστη στη MongoDB

### 4.5.3. Σύνδεση και Έκδοση JWT

Κατά τη διαδικασία σύνδεσης, μέσω της σελίδας login.html, ο κεντρικός server ελέγχει τα στοιχεία του χρήστη, εκδίδει JWT, το αποθηκεύει σε cookie και εκκινεί το προσωπικό Node-RED instance του χρήστη σε ξεχωριστή θύρα όπου γίνεται αυτόματο redirect. Επιπλέον, εφαρμόζεται έλεγχος ώστε να μην επιτρέπεται ταυτόχρονη σύνδεση σε διαφορετικό χρήστη στο ίδιο πρόγραμμα περιήγησης.

**Έλεγχος υπάρχοντος cookie (αποτροπή πολλαπλών logins)**

Πριν γίνει ο έλεγχος διαπιστευτηρίων, αν υπάρχει ήδη έγκυρο cookie authToken με διαφορετικό χρήστη, η σύνδεση απορρίπτεται :

```
const existing = req.cookies?.authToken;
 if (existing) {
   try {
     const decoded = jsonwebtoken.verify(existing, secret);
     if (decoded?.user && decoded.user !== username) {
       return res.status(409).json({
         success: false,
```

```
        message: `Είσαι ήδη συνδεδεμένος ως ${decoded.user}.`
      });
    }
  } catch (_) {
    // Άκυρο/ληγμένο token → συνεχίζουμε κανονικά στο login
  }
 }
 // ... συνεχίζει η ροή ελέγχου
});
```

**Έλεγχος διαπιστευτηρίων και έκδοση JWT σε cookie**

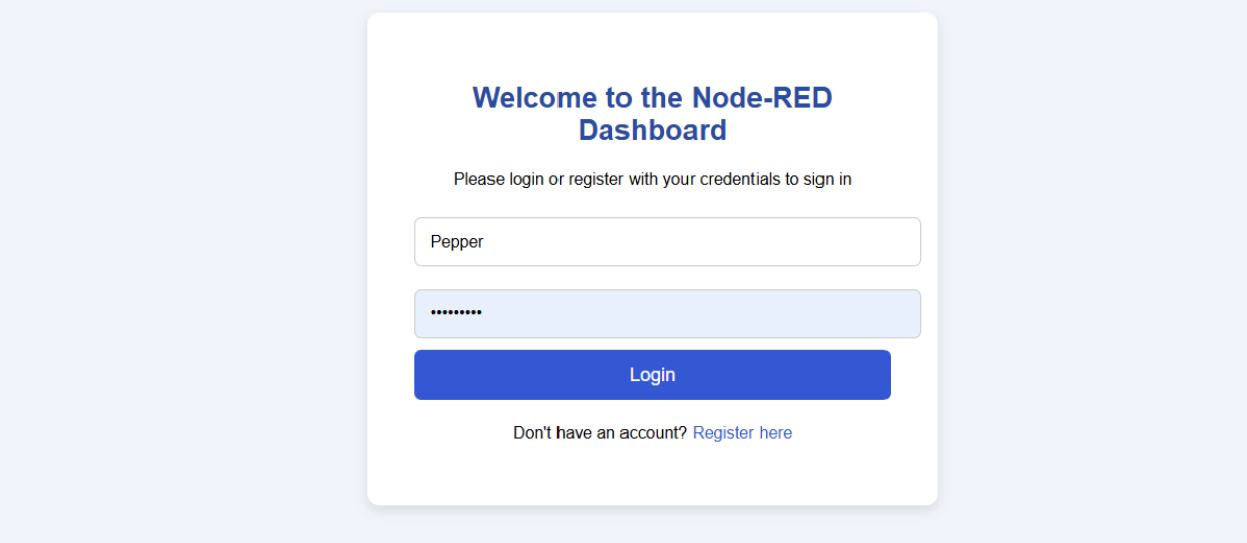
Αν τα στοιχεία είναι σωστά, εκδίδεται JWT και γράφεται σε httpOnly cookie ώστε να μην είναι προσβάσιμο από JavaScript και να συνοδεύει αυτόματα κάθε αίτημα προς τα http nodes/dashboard του Node-RED χωρίς πρόσθετο client-side κώδικα για ρύθμιση headers.

Το παρακάτω αποτελεί ενδεικτικό απόσπασμα :

```
const user = await User.findOne({ username });
if (!user || !(await bcrypt.compare(password, user.password))) {
   return res.status(401).json({ message: 'Λάθος όνομα χρήστη ή κωδικός.' });
}

const token = jsonwebtoken.sign({ user: username }, secret, { expiresIn: "1h" });
res.cookie("authToken", token, { httpOnly: true , path: '/', sameSite: 'Lax'});
```

**Δημιουργία προσωπικού φακέλου και παραγωγή settings.js**

Μετά το επιτυχές login, ενημερώνεται ο προσωπικός φάκελος του χρήστη ./users/<username> και παράγεται δυναμικά το προσωπικό settings.js, «δεμένο» με το τρέχον SECRET της εκτέλεσης του Express Server. Στο αρχείο αυτό υπάρχουν τα httpAdminMiddleware και httpNodeMiddleware για έλεγχο JWT στα admin endpoints και στα http nodes/dashboard. Επίσης θέτει στο αντικείμενο functionGlobalContext τα στοιχεία του χρήστη και session data.




**Εικόνα 4.4** : Απεικόνιση σελίδας σύνδεσης (login.html)

Το παρακάτω αποτελεί ενδεικτικό απόσπασμα :

```
// 1) Ορισμός προσωπικών διαδρομών & αρχείων χρήστη
const userDir = `./users/${username}`;              // φάκελος του χρήστη
const settingsPath = `${userDir}/settings.js`;      // per-user settings.js
const flowFile = 'flows.json';                      // όνομα αρχείου ροών

// 2) Δεδομένα χρήστη που θα περάσουν στο functionGlobalContext
const userData = {
    username: user.username,
    firstname: user.firstname,
    lastname: user.lastname,
    university: user.university,
    email: user.email,
    role: 'user',
    port: user.port
};

// 3) Δημιουργία φακέλου αν δεν υπάρχει
if (!fs.existsSync(userDir)) {
    fs.mkdirSync(userDir, { recursive: true });
}
```

### 4.5.4. Προστασία API

Για τον κεντρικό και ενιαίο έλεγχο πρόσβασης επιλέχθηκε η προστασία του namespace /api/* στον Express, ώστε τα public html (login,register και welcome) αρχεία να είναι προσβάσιμα ενώ κάθε κλήση σε API να απαιτεί έγκυρο JWT. Ο έλεγχος γίνεται με express-jwt, το οποίο διαβάζει το JWT από το httpOnly cookie, το ελέγχει με το ίδιο SECRET και αν είναι σωστό, το αίτημα προχωρά. Αν δεν είναι, ο server απορρίπτει το αίτημα ως μη εξουσιοδοτημένο (HTTP 401).

Το παρακάτω αποτελεί ενδεικτικό απόσπασμα :

```
// JWT μόνο για /api
app.use(
  '/api',
  jwt({
    secret,
    algorithms: ['HS256'],
    getToken: (req) => req.cookies?.authToken
  })
);
```

Επιπλέον, έγινε προσθήκη ενός /api/profile το οποίο φέρνει τα στοιχεία του χρήστη όταν κληθεί. Προστατεύεται μέσω του middleware express-jwt και κάνει εμφανή την σωστή λειτουργία των ελέγχων που έχουν χρησιμοποιηθεί. Αν δεν υπάρχει έγκυρο JWT τότε επιστρέφει μήνυμα λάθους.

```
// Προστατευμένο API: Προφίλ χρήστη
app.get('/api/profile', async (req, res) => {
  try {
    const username = req.auth.user;         // από το επαληθευμένο JWT
    const user = await User.findOne({ username }).select('-password -jwtToken');
    if (!user) return res.status(404).json({ success: false, message: 'User not found' });
    res.json({ success: true, user });
  } catch (err) {
    console.error('/api/profile error:', err);
    res.status(500).json({ success: false, message: 'Σφάλμα στον server.' });
  }});
```

### 4.5.5. Προστασία των Node-RED instances

Πέρα από το κεντρικό /api, κάθε προσωπικό Node-RED instance προστατεύεται από το settings.js που έχει αυτός ο χρήστης. Εκεί εφαρμόζονται οι έλεγχοι για τα admin endpoints και http nodes και dashboard από τα αντίστοιχα middleware (httpAdminMiddleware και httpNodeMiddleware) όπως είδαμε πιο πάνω στο 4.5.3. Και τα δύο διαβάζουν το JWT από το httpOnly cookie, και αν είναι έγκυρο, αφήνουν το αίτημα να προχωρήσει, διαφορετικά το μπλοκάρουν. Με αυτόν τον τρόπο έχουμε διπλή προστασία, τόσο για API όσο και σε επίπεδο χρήστη. Τέλος, μετά από επανεκκίνηση του server το μυστικό (SECRET) ανανεώνεται, έτσι το παλιό cookie γίνεται άκυρο και ένα κεντρικό middleware στον κεντρικό Express τα εντοπίζει και τα διαγράφει.

Το middleware για τη διαγραφή cookie κατά το restart:

```
// Διαγραφή cookie κατά το restart
app.use((req, res, next) => {
  const token = req.cookies?.authToken;
  if (!token) return next();
  try {
    jsonwebtoken.verify(token, secret); // έλεγχος με το TPEXON SECRET
    return next();
  } catch {
    // Άκυρο/παλιό token μετά από restart → καθάρισε το cookie
    res.clearCookie('authToken', { path: '/', sameSite: 'Lax' });
    return next();
  }
});
```

### 4.5.6 Διεπαφή χρήστη (login / register / welcome) και ροή πλοήγησης

Οι σελίδες login, register, welcome και profile σερβίρονται ως στατικά αρχεία από τον κεντρικό Express, ώστε το σημείο εισόδου να είναι πάντα το ίδιο (http://localhost:1880/). Με την εκκίνηση του server ανοίγει αυτόματα η σελίδα σύνδεσης (login.html) στον browser.

Μετά το επιτυχές login δεν γίνεται server-side HTTP redirect, ο server επιστρέφει JSON με το URL του προσωπικού instance. Το frontend αρχικά οδηγεί τον χρήστη στη σελίδα welcome**.**html, περνώντας ως παράμετρο το URL αυτό. Η welcome.html επιβεβαιώνει τη σύνδεση μέσω του endpoint /api/profile, εμφανίζει προσωποποιημένο μήνυμα Welcome <*username*> και, μετά από μικρή καθυστέρηση, πλοηγεί αυτόματα τον χρήστη στο προσωπικό του περιβάλλον (π.χ. http://localhost:1882/).

Ο έλεγχος πρόσβασης γίνεται αποκλειστικά στον server, στο Express middleware για τα endpoints /api και στο settings.js των per-user instances. Τα ίδια τα HTML αρχεία δεν χειρίζονται tokens, απλώς υποβάλλουν φόρμες ή εμφανίζουν δεδομένα που επιστρέφει το API.

```
// Σερβίρισμα στατικών αρχείων (login.html, register.html, welcome.html)
app.use(express.static(path.join(__dirname, 'public')));


// Εκκίνηση κεντρικού server & αυτόματο άνοιγμα login
server.listen(1880, async () => {
  console.log(' Server τρέχει στο http://localhost:1880/');
  const { default: open } = await import('open');
  open('http://localhost:1880/login.html');
});
```

Στην περίπτωση που θέλουμε τη δημιουργία νέου χρήστη, στη διεπαφή του login υπάρχει η επιλογή ανακατεύθυνσης στη σελίδα register**.**html. Εκεί ο νέος χρήστης συμπληρώνει τα απαραίτητα πεδία, αναλαμβάνει αυτόματα την επόμενη διαθέσιμη θύρα μέσω της getAvailablePort και αποθηκεύεται στη MongoDB. Μετά την επιτυχή εγγραφή του, ο χρήστης επιστρέφει στη σελίδα login για να συνδεθεί με τον νέο του λογαριασμό.

## 4.6. Διατήρηση Δεδομένων Συνεδρίας (SessionData)

Στο παραγόμενο per-user settings.js αξιοποιούμε το functionGlobalContext για να γίνουν προσβάσιμα μέσω από τα flows και το dashboard δύο μορφές δεδομένων. Αρχικά τα βασικά στοιχεία του χρήστη και δεύτερον δεδομένα συνεδρίας όπως η ώρα σύνδεσης και ένα sessionId. Κάθε αίτημα φέρνει μόνο του τα στοιχεία ταυτοποίησης μέσα στο cookie και προσφέρει άμεση πρόσβαση στα στοιχεία χωρίς κλήσεις στη βάση. Τα δεδομένα συνεδρίας είναι εφήμερα και δημιουργούνται κατά το login ενώ παύουν να ισχύουν με το restart του server.

Το αντικείμενο functionGlobalContext μέσα στο settings.js του χρήστη:

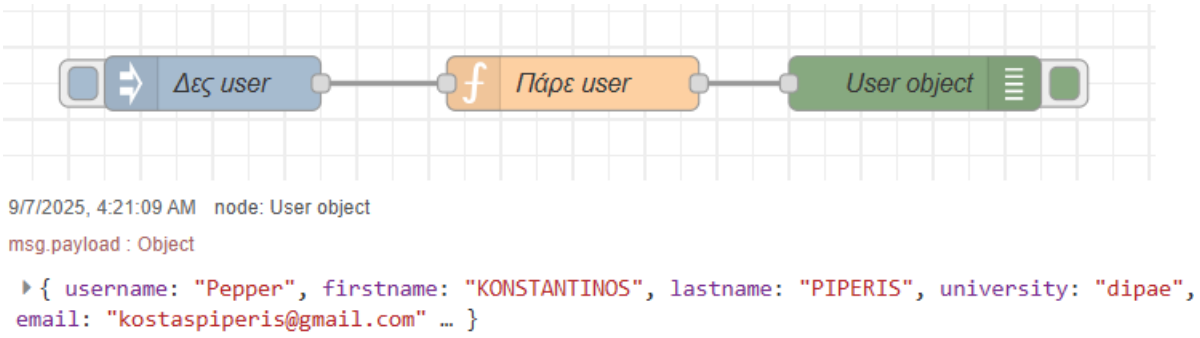
```
functionGlobalContext: {
  user: { /* ... στοιχεία χρήστη ... */ },
  sessionData: { /* loginTime, sessionId ... */ }
```

```
}
```

Με την δημιουργία δύο function nodes μέσα στο dashboard του χρήστη μπορούμε να έχουμε πρόσβαση σε αυτές τις πληροφορίες.

Για στοιχεία χρήστη:

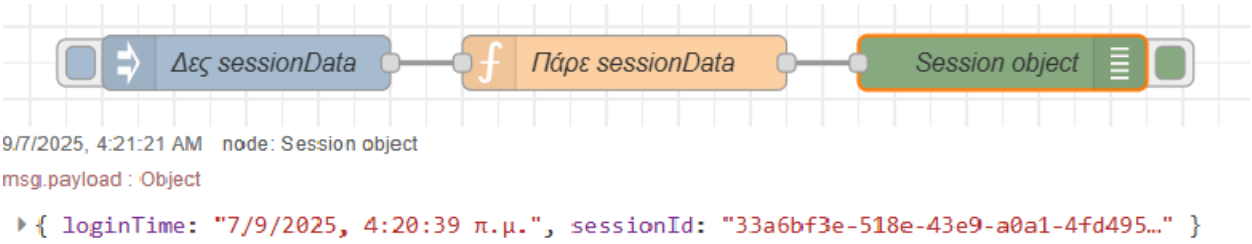
```
// Function node: "Get User"
const user = global.get('user');   // από το functionGlobalContext του settings.js
return { payload: user };
```




**Εικόνα 4.5:** Στοιχεία χρήστη διαθέσιμα μέσα από flow

Για δεδομένα συνεδρίας:

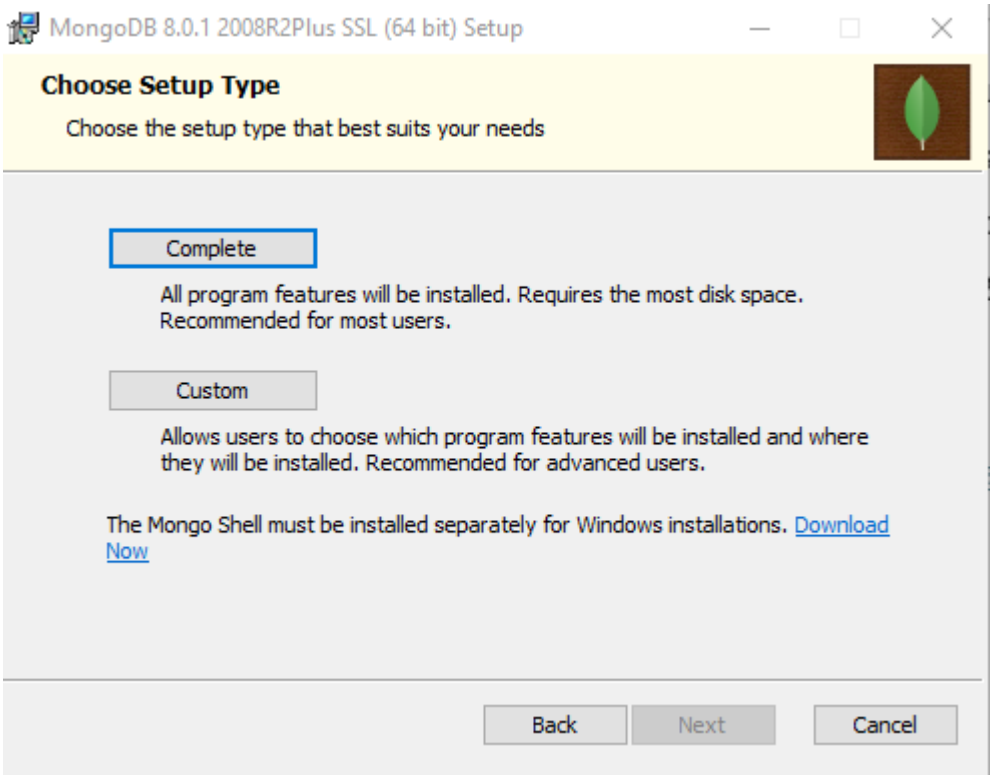
```
// Function node: "Get Session"
const session = global.get('sessionData');  // από το functionGlobalContext
return { payload: session };
```



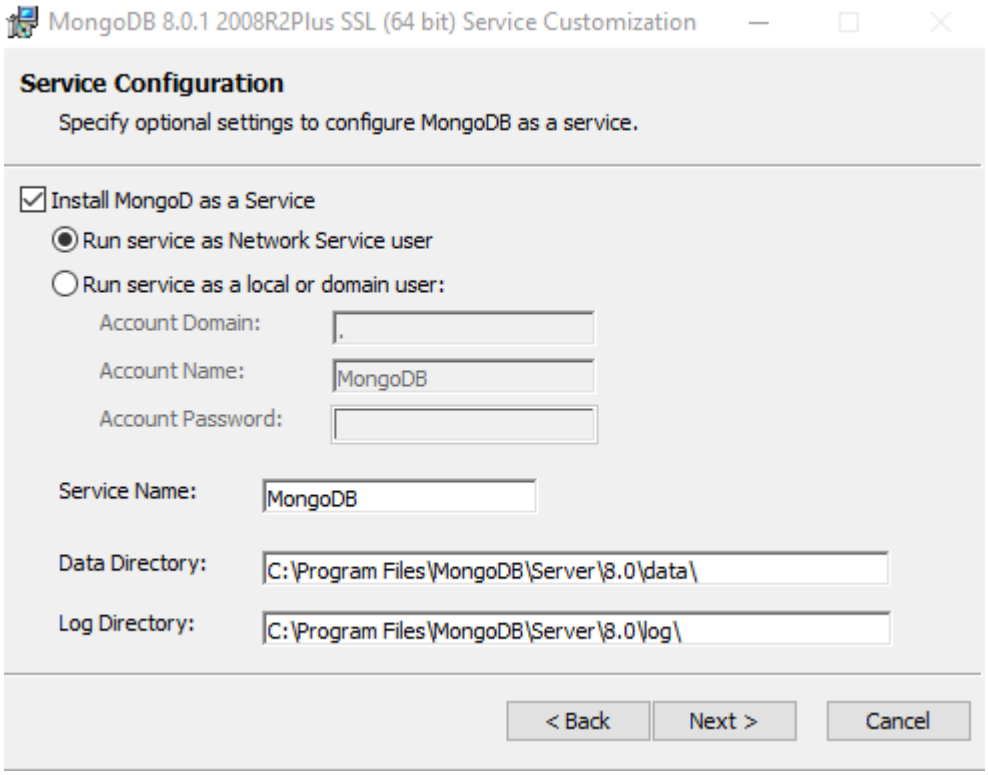

**Εικόνα 4.6:** Δεδομένα συνεδρίας μέσα από flow

## 4.7. Εγκατάσταση Βάσης MongoDB

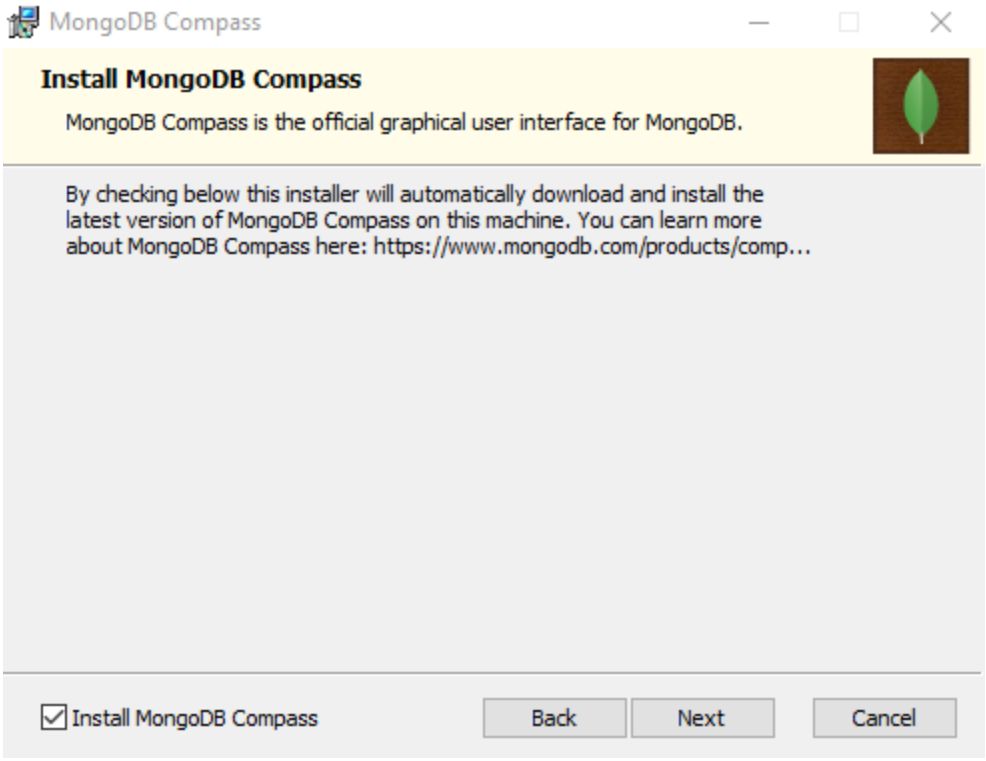
Αρχικά ο χρήστης θα πρέπει να εγκαταστήσει την επιλεγμένη βάση MongoDB για λειτουργικό σύστημα Windows 10+ από εδώ. Στη συνέχεια θα πάει στο φάκελο downloads του υπολογιστή του όπου και έχει εγκατασταθεί και θα πατήσει διπλό κλικ πάνω στο .msi αρχείο το οποίο θα έχει όνομα τύπου mongodb-windows-x86_64-8.0.1-signed.msi . Έπειτα θα χρειάζεται να πατήσει στον τύπο εγκατάστασης ολοκληρωμένη (complete) προκειμένου να εγκατασταθούν όλα τα πακέτα της mongodb καθώς και να επιλέξει να τρέξει την υπηρεσία σαν χρήστης δικτύου. Τέλος είναι βασικό να μπορεί να έχει ένα γραφικό περιβάλλον το οποίο να διαχειρίζεται τα δεδομένα της εφαρμογής που θα αναπτυχθεί, οπότε θα πρέπει να πατήσει τικ στο κουμπί της αυτόματης εγκατάστασης του mongo DB compass. Η εφαρμογή αυτή μπορεί να εγκατασταθεί και από εδώ.




**Εικόνα 4.7:** Επιλογή ολοκληρωμένης εγκατάστασης




**Εικόνα 4.8**: Επιλογή εκτέλεσης ως χρήστης δικτύου




**Εικόνα 4.9**: Επιλογή της αυτόματης εγκατάστασης

# Κεφάλαιο 5: Υλοποίηση του Εκτεταμένου Προσθέτου και Δημιουργία Πακέτου Εγκατάστασης

Στο προηγούμενο κεφάλαιο είδαμε την αρχιτεκτονική και τις βασικές επιλογές που κάναμε για να στηθεί η εφαρμογή. Τώρα περνάμε στο πρακτικό κομμάτι, πώς όλα αυτά γίνονται πράξη στον κώδικα. Στόχος είναι να δείξουμε αναλυτικά τα βήματα της υλοποίησης, ώστε να φανεί ξεκάθαρα πώς υλοποιείται η θεωρία που περιγράψαμε πριν.

Αρχικά θα δούμε τη δομή του κεντρικού server και πώς αυτός συνεργάζεται με τη βάση MongoDB, τα middleware και τα προσωπικά Node-RED instances. Έπειτα θα αναλύσουμε τη ροή εγγραφής και σύνδεσης ενός χρήστη, την παραγωγή και χρήση του JWT, καθώς και το πώς δημιουργείται για κάθε χρήστη το δικό του αρχείο ρυθμίσεων settings.js. Στη συνέχεια θα παρουσιάσουμε το frontend (login, register, welcome, profile) και τον ρόλο που έχει κάθε σελίδα στην εμπειρία του χρήστη.

Το κεφάλαιο θα κλείσει με την παρουσίαση της δημιουργίας ενός εγκαταστάσιμου πακέτου, ώστε η εφαρμογή να μπορεί να στηθεί εύκολα και σε άλλα συστήματα χωρίς πολύπλοκες ρυθμίσεις.

## 5.1. Γενική Αρχιτεκτονική και Αρχικοποίηση του node-red-server.js

Το node-red-server.js αποτελεί το κεντρικό εκτελέσιμο της εφαρμογής. Όλες οι ευθύνες είναι συγκεντρωμένες εκεί. Αρχικά εκκινεί τον web server στην θύρα 1880, σερβίρει την διεπαφή (login, register, welcome, profile), συνδέεται στην MongoDB και ορίζει το μοντέλο χρήστη. Διαμορφώνει τις πολιτικές ασφάλειας (JWT, cookies και καθαρισμός αυτών) και γεφυρώνει το login με την εκκίνηση του προσωπικού Node-RED instance για κάθε χρήστη. Έτσι η αυθεντικοποίηση παραμένει κεντρική και η εκτέλεση είναι απομονωμένη σε επίπεδο χρήστη.

**Κύκλος ζωής κατά την εκκίνηση**

Με την εκκίνηση του server, δημιουργείται ένα νέο secret για τα JWT. Το secret αυτό χρησιμοποιείται για την επαλήθευση tokens καθ'όλη την διάρκεια της συγκεκριμένης εκτέλεσης ενώ σε νέα εκτέλεση παράγεται και νέο secret. Πριν τα endpoints, ενεργοποιούνται τα βασικά middlewares (express.json() και cookie-parser) αλλά και το στατικό σερβίρισμα του φακέλου public που περιέχει τα html αρχεία, ώστε το σημείο εισόδου να είναι πάντα το http://localhost:1880/.

Ακολουθεί ένα global middleware καθαρισμού cookies ώστε να μην υπάρξουν ασυμφωνίες με παλιά ή άκυρα cookies μετά από επανεκκίνηση του server. Στην συνέχεια έχουμε την προστασία των /api με express-jwt. Με την προστασία των /api, μένουν προσβάσιμες οι σελίδες διεπαφής ενώ κάθε κλήση API υποχρεώνεται να συνοδεύεται από έγκυρο JWT.

Παράλληλα ενσωματώνεται ένας κεντρικός Node-RED runtime στον ίδιο server. Ο ρόλος του είναι υποστηρικτικός και δεν αντικαθιστά τα per-user instances, αλλά εξυπηρετεί τις βασικές λειτουργίες όντας απομονωμένος όμως από τους προσωπικούς καταλόγους των χρηστών. Τέλος, ο server αρχίζει να ακούει στη θύρα 1880 και ανοίγει αυτόματα τη σελίδα login.html στον προεπιλεγμένο browser.

**Αποθήκευση δεδομένων και δεδομένα συνεδρίας**

Η αποθήκευση των δεδομένων των χρηστών γίνονται στη MongoDB αλλά και στον δίσκο μέσα στο προσωπικό φάκελο ./users/<username>/. Στην βάση αποθηκεύονται τα στοιχεία του χρήστη, ο μοναδικός αριθμός θύρας που του αποδίδεται, το hash του κωδικού με bcrypt και προαιρετικά το όνομα του αρχείου της φωτογραφίας που αποθηκεύεται στον φάκελο uploads/ μέσω multer. Στον δίσκο, για κάθε χρήστη υπάρχει προσωπικός φάκελος ./users/<username>/ , όπου στο πρώτο επιτυχές login δημιουργείται το προσωπικό settings.js και φιλοξενούνται οι ροές του Node-RED, ώστε το περιβάλλον εργασίας του να διατηρείται ξεχωριστά από των υπολοίπων. Αντίθετα, ο μηχανισμός ταυτοποίησης είναι χωρίς server-side session. Το JWT αποστέλλεται σε κάθε αίτημα και ελέγχεται με τον τρέχον secret, ενώ στο functionGlobalContext κρατάμε μόνο ασφαλή πεδία όπως τα στοιχεία του χρήστη και τα sessionData (loginTime και sessionId) για άμεση χρήση από flows/dashboard. Έτσι, μετά από το restart, τυχόν παλιά cookies παύουν να ισχύουν και διαγράφονται αυτόματα από το κεντρικό middleware. Ο χρήστης κάνει νέα σύνδεση , παράγεται νέο settings.js, πάντα δεμένο με το τρέχον secret και εκκινεί το προσωπικό instance του χρήστη.

## 5.2. Διαχείριση Χρηστών

Η εφαρμογή υλοποιεί πλήρη ροή δημιουργίας και σύνδεσης χρήστη στον κεντρικό server. Κατά την εγγραφή, ο server ελέγχει τα υποχρεωτικά πεδία, αποτρέπει διπλή καταχώρηση του ίδιο ονόματος χρήστη, αναθέτει μοναδική θύρα στο νέο προφίλ, παράγει hash του κωδικού με bcrypt και αποθηκεύει τα στοιχεία στη MongoDB. Η απόκριση της εγγραφής είναι JSON που καθοδηγεί το περιβάλλον να επιστρέψει στη σελίδα σύνδεσης.

Η ανάθεση θύρας γίνεται δυναμικά. Ο server ξεκινά από τη θύρα 1881 και προχωρά αυξάνοντας μέχρι να βρεθεί ελεύθερη θύρα, δηλαδή θύρα που δεν έχει καταληφθεί από άλλον χρήστη. Με αυτόν τον τρόπο κάθε προφίλ παίρνει αποκλειστικό port για το προσωπικό του Node-RED instance.

```
//  Ανάθεση θύρας για χρήστη
const getAvailablePort = async () => {
   let port = 1881;
   while (await User.findOne({ port })) {
      port++;
   }
   return port;
};
```

Το endpoint της εγγραφής δέχεται τα δεδομένα, ελέγχει τις προϋποθέσεις και δημιουργεί την εγγραφή. Αν το username υπάρχει ήδη, η ροή σταματά με κατάλληλο μήνυμα σφάλματος, διαφορετικά προχωρά με την δημιουργία του hash κωδικού και την ανάθεση θύρας, πριν αποστείλει την απόκριση για επιστροφή στο login.

```
// REGISTER
app.post('/register', upload.single('photo'), async (req, res) => {
  const { username, password, firstname, lastname, university, email } = req.body;
  const photoPath = req.file ? req.file.filename : null;

  // 1) Έλεγχος υποχρεωτικών πεδίων
  if (!username || !password || !firstname || !lastname || !university || !email) {
    return res.status(400).json({ message: 'Όλα τα πεδία είναι υποχρεωτικά.' });
  }

  try {
    // 2) Αποφυγή διπλής εγγραφής
    if (await User.findOne({ username })) {
      return res.status(400).json({ message: 'Το όνομα χρήστη υπάρχει ήδη.' });
    }

    // 3) Ανάθεση θύρας και hash κωδικού
```

```
    const port = await getAvailablePort();
    const hashedPassword = await bcrypt.hash(password, 10);

    // 4) Καταχώριση χρήστη στη MongoDB
    await new User({
      username, password: hashedPassword, firstname, lastname,
      university, email, photo: photoPath, port
    }).save();
```

Στην σύνδεση εφαρμόζεται πρώτα ο έλεγχος για μοναδικό login ανά browser. Αν ο browser κουβαλά ήδη έγκυρο JWT άλλου χρήστη, η νέα προσπάθεια μπλοκάρεται και ο server ενημερώνει ότι υπάρχει ήδη ενεργή σύνδεση. Αν δεν ισχύει κάτι τέτοιο, ο server αναζητά τον χρήστη στη βάση, συγκρίνει το hash του κωδικού, εκδίδει JWT με διάρκεια μίας ώρας σε httpOnly cookie και ενημερώνει το μοντέλο. Αμέσως μετά, δημιουργεί ή ενημερώνει τον προσωπικό φάκελο του χρήστη, μαζί και το settings.js που είναι δεμένο με το τρέχον secret, έτσι ώστε να προστατεύονται τα editor και http nodes και να είναι διαθέσιμα τα user και sessionData στο functionGlobalContext. Στην συνέχεια εκκινεί το προσωπικό Node-RED instance στη θύρα που έχει αναλάβει ο χρήστης και επιστρέφει στο frontend το URL ανακατεύθυνσης για να ολοκληρωθεί η πλοήγηση.

```
// LOGIN
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    // 1) Επαλήθευση διαπιστευτηρίων
    // 2) Έκδοση JWT σε httpOnly cookie (λήξη 1h)
    // 3) Προσωπικοί πόροι: φάκελος & settings.js
    // 3a) Δεδομένα που θα εισαχθούν στο functionGlobalContext
    };
// 3b) Παραγωγή per-user settings.js
    const settingsContent = `
const cookie = require("cookie");
const jwt = require("jsonwebtoken");
const SECRET = "${secret}";
```

```
module.exports = {
  // Admin endpoints (editor, /auth/*)
  httpAdminMiddleware: (req, res, next) => { /* έλεγχος JWT + logout */ },
  // http nodes & Dashboard (/ui)
  httpNodeMiddleware:  (req, res, next) => { /* έλεγχος JWT για /ui & http nodes */ },
  functionGlobalContext: {
    user: ${JSON.stringify(userData)},
    sessionData: { loginTime: ${Date.now()}, sessionId: "${crypto.randomUUID()}" }
  }
};`;
    fs.writeFileSync(settingsPath, settingsContent);

    // 4) Εκκίνηση προσωπικού Node-RED instance
    exec(`node-red -p ${user.port} --userDir ${userDir} --settings ${settingsPath} --
flowFile ${flowFile}`,
      (error) => {
        if (error) {
          console.error('Σφάλμα εκκίνησης Node-RED:', error);
          return res.status(500).json({ message: 'Σφάλμα εκκίνησης Node-RED' });
        }
        console.log(`Node-RED εκτελείται στη θύρα ${user.port} για ${username}`);
      });
```

Σε μελλοντική επέκταση, η λύση μπορεί να υποστηρίξει ανανέωση tokens, σταθερό secret και ρόλους πρόσβασης ανά χρήστη, χωρίς αλλαγές στην βασική αρχιτεκτονική.

## 5.3. Υλοποίηση ταυτοποίησης με JWT

Η ταυτοποίηση στη λύση μας βασίζεται σε JSON Web Tokens (JWT) που εκδίδονται κατά την σύνδεση, αποθηκεύονται σε httpOnly cookie και ελέγχονται αποκλειστικά από τον server. Δεν διατηρούμε server-

side session και κάθε αίτημα συνοδεύεται από την απόδειξη ταυτότητας ενώ ο server την επαληθεύει με το τρέχον secret. Έτσι το frontend παραμένει απλό, δεν διαχειρίζεται tokens, ενώ η πρόσβαση θωρακίζεται σε δύο επίπεδα, κεντρικά στον Express για τα /api και τοπικά μέσα σε κάθε προσωπικό Node-RED instance μέσω του παραγόμενου settings.js. Μετά από κάθε επανεκκίνηση του server παράγεται νέο secret, άρα παλιά cookies καθίστανται άκυρα και καθαρίζονται αυτόματα από ένα κεντρικό middleware. Ο χρήστης συνδέεται εκ νέου και παράγεται φρέσκο settings.js δεμένο με το νέο μυστικό.

### 5.3.1. Κεντρικός Έλεγχος στο /api (Express)

Ο κεντρικός server κλειδώνει το namespace των API με το express-jwt. Το middleware διαβάζει το token από το cookie, το επαληθεύει με το τρέχον secret και αν είναι έγκυρο βάζει τα στοιχεία του χρήστη στο req.auth. Έτσι τα endpoints τα βρίσκουν έτοιμα χωρίς να ξαναδιαβάζουν cookies ή headers. Αν ο έλεγχος αποτύχει, η κλήση απορρίπτεται ως μη εξουσιοδοτημένη (HTTP 401) με σαφές μήνυμα.

```
// Προστασία μόνο για /api
app.use(
 '/api',
 jwt({
   secret,
   algorithms: ['HS256'],
   getToken: (req) => req.cookies?.authToken
 }));
```

Η διαμόρφωση του cookie είναι κρίσιμη. Αρχικά δηλώνεται ως httpOnly cookie ώστε να μην μπορεί να το διαβάσει JavaScript στον browser, άρα δύσκολα κλέβεται με κακόβουλο script. Το samesite το κρατάμε ως ‘Lax’ ώστε το cookie να στέλνεται μέσα στον ιστότοπο αλλά όχι σε τρίτες σελίδες. Το /path δηλώνεται με αυτόν τον τρόπο με σκοπό να ισχύει για όλο το hostname.

Η έκδοση του token γίνεται στο login, με λήξη μίας ώρας. Το frontend δεν διαβάζει ποτέ το token, λαμβάνει μόνο το redirect URL του προσωπικού instance και πλοηγεί τον χρήστη εκεί. Στην υλοποίηση μας το τρέχον token αποθηκεύεται στο πεδίο jwtToken του χρήστη στη MongoDB, το οποίο όμως είναι βοηθητικό και δεν χρησιμοποιείται για σκοπούς εξουσιοδότησης.

```
const token = jsonwebtoken.sign({ user: username }, secret, { expiresIn: "1h" });
```

```
res.cookie("authToken", token, { httpOnly: true, path: '/', sameSite: 'Lax' });
```

Ένα επιπλέον στρώμα ανθεκτικότητας είναι το global middleware καθαρισμού. Το συγκεκριμένο middleware, μετά από restart ελέγχει το authToken με το νέο secret, και αν δεν ισχύει, το διαγράφει. Έτσι αποφεύγονται τυχόν cookies που έμειναν στον browser και ο χρήστης οδηγείται σε καθαρή ροή σύνδεσης.

### 5.3.2. Τοπικοί έλεγχοι στο per-user settings.js (editor, http nodes, dashboard)

Μετά την επιτυχή σύνδεση, παράγεται δυναμικά το προσωπικό settings.js του χρήστη. Σε αυτό εφαρμόζουμε το secret της τρέχουσας εκτέλεσης και ορίζουμε δύο middleware που λειτουργούν εντός του συγκεκριμένου Node-RED instance:

- **httpAdminMiddleware (editor και admin endpoints):** Κάθε αίτημα προς τον editor περνά από έλεγχο JWT. Επιπλέον, αν ο editor καλέσει logout (POST /auth/revoke), καθαρίζουμε συγχρονισμένα το authToken και τυχόν Node-RED cookies.
- **httpNodeMiddleware (http nodes και dashboard /ui):** Καλύπτει τα runtime endpoints του χρήστη, συμπεριλαμβανομένου του Dashboard. Το cookie διαβάζεται από τα headers, το token επαληθεύεται με το secret αυτού του instance και αν δεν είναι έγκυρο επιστρέφεται 401 και καθαρίζεται το cookie ώστε ο browser να μην στέλνει ξανά άκυρο token.

Αυτό το διπλό τοίχος προστασίας έχει συγκεκριμένα οφέλη:

- **Σαφή σημεία ελέγχου:** Ό,τι είναι API περνά από τον Express ενώ ότι είναι editor/UI περνά από το τοπικό middleware. Ο κώδικας εξουσιοδότησης είναι συγκεντρωμένος και ελεγχόμενος.
- **Απομόνωση ανά χρήστη:** Ακόμη και αν κάποιος προσπαθήσει να προσπελάσει κατευθείαν το /ui παρακάμπτοντας το κεντρικό /api, το προσωπικό instance απαιτεί έγκυρο JWT και μπλοκάρει κάθε άλλη προσπάθεια.
- **Σταθερή εμπειρία:** Το frontend δεν χρειάζεται να αποθηκεύσει ή μεταφέρει κάποιο token. Λαμβάνει μόνο το URL του προσωπικού instance και συνεχίζει σε αυτό. Τα flows και το dashboard αντλούν user και sessionData από το functionGlobalContext, χωρίς να εμπλέκονται με την ταυτοποίηση.

Τέλος, η πολιτική του single-login ανά browser που εφαρμόζουμε στο /login συμπληρώνει το παραπάνω σχήμα. Αν υπάρχει ήδη έγκυρο cookie άλλου χρήστη, εμποδίζεται η δεύτερη σύνδεση από την ίδια συνεδρία browser. Σε συνδυασμό με την ανανέωση του secret σε κάθε restart, εξασφαλίζεται ότι δεν παραμένουν παλιά, λειτουργικά tokens μετά από το restart και ότι κάθε προσωπικό instance ξεκινά πάντα με καθαρό περιβάλλον αυθεντικοποιήσης. Μελλοντικά, η ίδια αρχιτεκτονική επιτρέπει εύκολη προσθήκη ρόλων πρόσβασης μέσα στο payload του JWT και σταθερού secret για μακρόχρονη ισχύ tokens χωρίς αλλαγές στον βασικό κορμό των middlewares.

Ο συνδυασμός κεντρικού ελέγχου στο /api και τοπικών ελέγχων ανά instance διασφαλίζει ότι κάθε dashboard και ροή εκτελούνται μόνο υπό έγκυρο JWT, με σαφή σημεία των πολιτικών ασφαλείας και καθαρά όρια ανά χρήστη.

## 5.4. Ροή πλοήγησης και διεπαφή (login / register / welcome / profile )

Το frontend της εφαρμογής διατηρείται εσκεμμένα απλό. Οι σελίδες login.html, register.html, welcome.html και profile.html σερβίρονται ως στατικά αρχεία από τον κεντρικό Express (θύρα 1880), ενώ όλη η λογική πρόσβασης και ελέγχου ταυτότητας εφαρμόζεται στον server. Το JWT δεν το χειρίζεται το JavaScript του browser, εκδίδεται στο /login και αποθηκεύεται σε httpOnly cookie, άρα αποστέλλεται αυτόματα σε κάθε επόμενο αίτημα προς το ίδιο host (π.χ. localhost:1880 και localhost:<userPort>), χωρίς client-side ρυθμίσεις headers. Το profile.html μας βοηθά να επιβεβαιώσουμε την σωστή λειτουργία των πολιτικών ασφαλείας μέσω της κλήσης του /profile.

### 5.4.1. Σελίδα σύνδεσης (login.html)

Η σελίδα σύνδεσης αποτελεί το πρώτο σημείο επαφής του χρήστη με την εφαρμογή και έχει σχεδιαστεί να είναι όσο το δυνατόν πιο απλή και λειτουργική. Το περιβάλλον της είναι στατικό, ώστε να διατηρείται ελαφρύ και γρήγορο, και περιλαμβάνει μια φόρμα με δύο βασικά πεδία, τα username και password. Η υποβολή της φόρμας δεν γίνεται με κλασικό form action, αλλά μέσω fetch προς το endpoint /login του Express server. Έτσι η εμπειρία χρήσης είναι πιο ομαλή και η επικοινωνία με τον server γίνεται ασύγχρονα, χωρίς ανανέωση σελίδας.

Όταν ο χρήστης εισάγει σωστά τα διαπιστευτήριά του, ο server εκδίδει ένα JWT διάρκειας μίας ώρας. Το token αυτό δεν εμφανίζεται ποτέ στο frontend, αλλά αποθηκεύεται αυτόματα σε cookie με ιδιότητες httpOnly, sameSite: 'Lax' και path: '/'. Η επιλογή του httpOnly αποτρέπει την πρόσβαση στο token από JavaScript. Το sameSite: 'Lax' περιορίζει τη χρήση του cookie μόνο σε σχετικές πλοηγήσεις, ενώ με το path: '/' το cookie ισχύει για όλη την εφαρμογή. Με αυτόν τον τρόπο, η λογική ελέγχου ταυτότητας μένει αποκλειστικά στο backend.

Μετά την επιτυχή σύνδεση, το frontend δεν πλοηγεί τον χρήστη κατευθείαν στο προσωπικό του instance· τον οδηγεί πρώτα στη σελίδα welcome.html, περνώντας σε αυτήν το URL του instance. Η welcome.html εμφανίζει προσωποποιημένο μήνυμα και στη συνέχεια πραγματοποιεί αυτόματα την ανακατεύθυνση. Συνολικά, η σελίδα login.html και το αντίστοιχο backend endpoint συνεργάζονται ώστε η εμπειρία σύνδεσης να είναι απλή, ασφαλής και ομαλή για τον χρήστη.

```
// login.html – υποβολή φόρμας με fetch
```

```
// login.html – υποβολή φόρμας με fetch
document.getElementById("login-form").addEventListener("submit", async (e) => {
  e.preventDefault();
  const username = document.getElementById("username").value;
  const password = document.getElementById("password").value;

  try {
    const res = await fetch("/login", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ username, password })
    });

    const data = await res.json();
    if (data.success) {
      // δεν πάμε κατευθείαν στο instance
      // αλλά περνάμε πρώτα από το welcome.html
      const target = encodeURIComponent(data.redirect);
      window.location.href = `/welcome.html?to=${target}`;
    } else {
      alert("Σφάλμα: " + data.message);
    }
  } catch (err) {
    console.error("Error:", err);
    alert("Παρουσιάστηκε σφάλμα κατά τη σύνδεση.");
  }
});
```

### 5.4.2. Σελίδα εγγραφής (register..html)

Η σελίδα εγγραφής παραμένει στατική και λιτή, με μία συγκεντρωτική φόρμα στο κέντρο της οθόνης. Τα πεδία που συμπληρώνει ο χρήστης είναι username, password, firstname, lastname, university, email και προαιρετικά φωτογραφία. Τα required στα πεδία κειμένου και ο τύπος email επιβάλλουν βασικούς ελέγχους από τον browser, ενώ δίπλα στο password και στο email εμφανίζονται κατατοπιστικά hints για τον χρήστη. Το πεδίο φωτογραφίας δέχεται μόνο εικόνες μέσω accept="image/*", ώστε να ανέβει με ασφάλεια μαζί με τη φόρμα.

Η υποβολή δεν γίνεται με action αλλά μέσω JavaScript. Συλλέγεται ένα FormData απευθείας από το <form> και αποστέλλεται με fetch στο /register χωρίς χειροκίνητο ορισμό headers, ώστε ο browser να στείλει σωστά το multipart/form-data μαζί με το αρχείο. Αν η εγγραφή ολοκληρωθεί επιτυχώς, εμφανίζεται μήνυμα επιβεβαίωσης και ο χρήστης μεταφέρεται στη σελίδα σύνδεσης. Σε περίπτωση σφάλματος, εμφανίζεται αναλυτικό μήνυμα από την απόκριση του server. Το frontend δεν χειρίζεται tokens και δεν αποθηκεύει τίποτα στο localStorage.

Η σχεδίαση της φόρμας στο CSS είναι εστιασμένη στην ευχρηστία και την καθαρότητα. Καθαρή τυπογραφία, ουδέτερο γκρι φόντο σελίδας, σκιά και στρογγυλεμένες γωνίες στο πλαίσιο της φόρμας για να ξεχωρίζει από το υπόβαθρο. Το αποτέλεσμα είναι μία απλή αλλά λειτουργική διεπαφή, όπου ο χρήστης καταλαβαίνει αμέσως τι πρέπει να συμπληρώσει και πώς να προχωρήσει στην είσοδο.

### 5.4.3. Σελίδα καλωσορίσματος (welcome.html)

Η σελίδα καλωσορίσματος λειτουργεί ως ενδιάμεσο βήμα ανάμεσα στη σύνδεση και την πλοήγηση στο προσωπικό Node-RED instance. Μετά το επιτυχές login, ο server επιστρέφει στο frontend το URL του instance και το login.html ανακατευθύνει τον χρήστη στη welcome.html περνώντας το URL αυτό ως παράμετρο.

Η welcome.html δεν χειρίζεται tokens ούτε αποθηκεύει στοιχεία στον browser. Αντίθετα, ζητά από το endpoint /api/profile τα στοιχεία του τρέχοντα χρήστη, χρησιμοποιώντας το JWT cookie που έχει ήδη εκδοθεί κατά το login. Έτσι εμφανίζει προσωποποιημένο μήνυμα τύπου *Welcome <username>!*, επιβεβαιώνοντας ότι η σύνδεση ήταν επιτυχής.

Το περιβάλλον είναι λιτό, με μια κεντρική κάρτα που ενημερώνει τον χρήστη ότι η σύνδεση ολοκληρώθηκε. Μετά από μικρή καθυστέρηση η σελίδα εκτελεί αυτόματα ανακατεύθυνση προς το προσωπικό instance (π.χ. http://localhost:1882/). Αν για οποιονδήποτε λόγο το cookie δεν είναι έγκυρο, η σελίδα επιστρέφει τον χρήστη πίσω στη φόρμα σύνδεσης.

Με αυτόν τον τρόπο η welcome.html συνδυάζει επιβεβαίωση σύνδεσης και προσωποποιημένο μήνυμα, πριν ανακατευθύνει αυτόματα τον χρήστη στο δικό του περιβάλλον εργασίας.

### 5.4.4. Σελίδα προφίλ (profile.html)

Η σελίδα προφίλ παρέχει στον χρήστη μια απλή επισκόπηση των προσωπικών του στοιχείων όπως αυτά είναι αποθηκευμένα στη MongoDB. Σερβίρεται ως στατικό αρχείο από τον κεντρικό Express και περιλαμβάνει μόνο την απαραίτητη JavaScript για να ανακτήσει τα δεδομένα από το API.

Με την είσοδο στη σελίδα γίνεται αίτημα στο endpoint /api/profile. Το αίτημα συνοδεύεται αυτόματα από το JWT cookie που εκδόθηκε κατά το login. Ο server, μέσω του middleware express-jwt, ελέγχει την εγκυρότητα του cookie και, αν όλα είναι σωστά, επιστρέφει τα στοιχεία του χρήστη (χωρίς password ή token). Αν το cookie λείπει ή είναι άκυρο, η σελίδα ειδοποιεί τον χρήστη και τον επιστρέφει στη φόρμα σύνδεσης.

Η παρουσίαση είναι απλή, ένα κουτί με τα βασικά πεδία (username, όνομα, email, πανεπιστήμιο, θύρα Node-RED). Έτσι ο χρήστης μπορεί εύκολα να δει την ταυτότητά του μέσα στο σύστημα.

Η σελίδα profile.html αξιοποιεί έτσι τον μηχανισμό αυθεντικοποίησης χωρίς να χειρίζεται απευθείας tokens, διατηρώντας το frontend καθαρό και τον έλεγχο πρόσβασης αποκλειστικά στον server.

## 5.5. Διαχείριση Συνεδρίας

Στην εφαρμογή μας η έννοια της συνεδρίας δεν υλοποιείται με κλασικό server-side session, αλλά με έναν διαφορετικό τρόπο που ταιριάζει στη φιλοσοφία του Node-RED. Κατά το login δημιουργείται ένα αντικείμενο sessionData, το οποίο περιέχει πληροφορίες για τη συνεδρία του χρήστη, όπως την ώρα σύνδεσης (loginTime) και έναν μοναδικό αναγνωριστικό κωδικό (sessionId)(βλ. Εικόνα 4.5).

Το αντικείμενο αυτό εισάγεται στο functionGlobalContext του προσωπικού Node-RED instance μέσα στο παραγόμενο settings.js, μαζί με τα βασικά στοιχεία του χρήστη(βλ. Εικόνα 4.4). Με αυτόν τον τρόπο, οι ίδιες οι ροές του χρήστη έχουν άμεση πρόσβαση στη συνεδρία τους μέσα από το περιβάλλον σχεδίασης του Node-RED, χωρίς να χρειάζεται να αποθηκεύουμε ή να μεταφέρουμε tokens. Για παράδειγμα, ένας κόμβος function μπορεί να διαβάσει global.get("sessionData") και να χρησιμοποιήσει τον χρόνο σύνδεσης ή το sessionId στη λογική του flow.

```
functionGlobalContext: {
  user: ${JSON.stringify(userData, null, 2)},
```

```
  sessionData: {
    loginTime: ${Date.now()},
    sessionId: "${crypto.randomUUID()}"
  }
```

Έτσι η «διαχείριση συνεδρίας» στη λύση μας μεταφράζεται σε πρόσβαση σε δεδομένα συνεδρίας μέσα από το ίδιο το Node-RED, αντί για αποθήκευση κατάστασης στον server. Η προσέγγιση αυτή είναι ελαφριά, stateless και ταιριάζει απόλυτα στη λογική του Node-RED, αφού τα flows μπορούν να αξιοποιούν άμεσα στοιχεία για τον τρέχοντα χρήστη και τη συνεδρία του.

## 5.6. Δημιουργία Πακέτου Εγκατάστασης και Ανάπτυξη σε Νέο Περιβάλλον

Για την ολοκλήρωση της εφαρμογής κρίθηκε αναγκαία η δημιουργία ενός πακέτου εγκατάστασης, ώστε να είναι δυνατή η εύκολη αναπαραγωγή και λειτουργία της σε οποιοδήποτε σύστημα χωρίς να απαιτείται πολύπλοκη διαδικασία παραμετροποίησης. Στόχος ήταν η απλοποίηση των βημάτων που πρέπει να ακολουθήσει ο τελικός χρήστης, ώστε με ελάχιστες ενέργειες να μπορέσει να αξιοποιήσει την εφαρμογή.

Η διαδικασία περιλαμβάνει τα εξής ακόλουθα στάδια:

**1. Ορισμός εξαρτήσεων στο package.json**

Στο αρχείο package.json καταγράφηκαν όλες οι απαραίτητες βιβλιοθήκες που χρησιμοποιούνται στον κώδικα (express, jsonwebtoken, mongoose, bcrypt, cookie-parser, κ.ά.). Με τον τρόπο αυτό, όταν η εφαρμογή εγκαθίσταται σε νέο υπολογιστή, η εντολή:

```
npm install
```

εγκαθιστά αυτόματα όλα τα απαραίτητα πακέτα. Οι βιβλιοθήκες bcrypt και cookie-parser προστέθηκαν χειροκίνητα από εμάς μέσα στο package.json.

**2. Χρήση Docker για τη βάση δεδομένων και το περιβάλλον Node-RED**

Για να μην απαιτείται χειροκίνητη εγκατάσταση της MongoDB και του Node.js στο εκάστοτε σύστημα, αξιοποιήθηκε το Docker. Με τη βοήθεια του αρχείου docker-compose.yml ορίζονται δύο υπηρεσίες:

- mongo για την εκτέλεση της βάσης MongoDB,
- node-red-app για την εφαρμογή Node-RED και τον server που υλοποιήθηκε.

Η εκτέλεση της εντολής:

```
docker compose up
```

δημιουργεί αυτόματα τα απαραίτητα containers και εκκινεί την εφαρμογή.

**3. Δημιουργία αρχείου εκκίνησης (start.bat) και τερματισμού (stop.bat)**

Για την απλοποίηση της χρήσης δημιουργήθηκαν δύο αρχεία τύπου batch (.bat):

- Το αρχείο start.bat εκτελεί εσωτερικά την εντολή docker compose up και ανοίγει αυτόματα τον φυλλομετρητή στη σελίδα σύνδεσης της εφαρμογής. Με τον τρόπο αυτό ο τελικός χρήστης δεν χρειάζεται να ανοίγει τερματικό και να πληκτρολογεί εντολές.
- Το αρχείο stop.bat εκτελεί την εντολή docker compose down, η οποία τερματίζει τα containers της εφαρμογής και απελευθερώνει τις θύρες του συστήματος.

Η χρήση των δύο αυτών αρχείων εξασφαλίζει ότι ο χρήστης μπορεί να ξεκινήσει ή να σταματήσει την εφαρμογή εύκολα με διπλό κλικ, χωρίς περαιτέρω τεχνικές γνώσεις.

**4. Εγκατάσταση σε νέο υπολογιστή**

Η διαδικασία εγκατάστασης σε νέο σύστημα συνοψίζεται στα εξής βήματα:

1. Εγκατάσταση του Docker Desktop.
2. Αντιγραφή του φακέλου της εφαρμογής στον υπολογιστή.
3. Εκτέλεση της εντολής npm install στον φάκελο του project (μόνο την πρώτη φορά).
4. Διπλό κλικ στο αρχείο start.bat.

Με τον τρόπο αυτό ο χρήστης δεν χρειάζεται να γνωρίζει τεχνικές λεπτομέρειες, καθώς το περιβάλλον εκτέλεσης δημιουργείται αυτόματα.

**5. Λειτουργία εφαρμογής μετά την εγκατάσταση**

Μετά την επιτυχή εκτέλεση του πακέτου, ο server της εφαρμογής είναι διαθέσιμος στη διεύθυνση http://localhost:1880/. Ο χρήστης μπορεί να προχωρήσει σε εγγραφή νέου λογαριασμού και στη συνέχεια να συνδεθεί, ώστε να αποκτήσει πρόσβαση στο δικό του instance Node-RED με πλήρη λειτουργικότητα.

# Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Εργασία

Στο παρόν κεφάλαιο συνοψίζονται τα αποτελέσματα που προέκυψαν από την ανάπτυξη της εφαρμογής, καθώς και τα οφέλη που προσφέρει η προτεινόμενη υλοποίηση σε σχέση με τις αρχικές απαιτήσεις. Παρουσιάζονται τα βασικά συμπεράσματα που αναδείχθηκαν μέσα από τη διαδικασία σχεδίασης και υλοποίησης, ενώ ταυτόχρονα εξετάζονται οι δυνατότητες περαιτέρω βελτίωσης και εξέλιξης του συστήματος. Η ανάλυση που ακολουθεί στοχεύει αφενός στην αποτίμηση της παρούσας προσέγγισης και αφετέρου στη διατύπωση προτάσεων για μελλοντική έρευνα και ανάπτυξη, οι οποίες μπορούν να επεκτείνουν τη λειτουργικότητα και την αξιοπιστία της πλατφόρμας.

## 6.1. Συμπεράσματα

Κατά την πορεία της υλοποίησης, αποδείχθηκε ότι είναι εφικτή η δημιουργία ενός περιβάλλοντος στο οποίο κάθε χρήστης διαθέτει το δικό του αυτόνομο και απομονωμένο Node-RED instance. Αυτό επιτυγχάνεται μέσω του κεντρικού server (node-red-server.js), ο οποίος αναλαμβάνει τόσο την ταυτοποίηση των χρηστών με τη χρήση JSON Web Tokens όσο και την εκκίνηση νέων instances με δυναμική ανάθεση port. Με αυτόν τον τρόπο διασφαλίζεται ότι τα δεδομένα και οι ροές κάθε χρήστη παραμένουν ανεξάρτητα, χωρίς να επηρεάζονται από τις ενέργειες των υπολοίπων.

Η χρήση της MongoDB αποδείχθηκε καθοριστική για την ασφαλή αποθήκευση των στοιχείων χρηστών, επιτρέποντας την ευέλικτη και αξιόπιστη διαχείριση των δεδομένων ταυτοποίησης. Αντίθετα με αρχικές προσεγγίσεις που βασίζονταν στην αποθήκευση flows στη βάση, η τελική λύση κράτησε τη διαχείριση των flows στα εκάστοτε Node-RED instances. Η επιλογή αυτή μείωσε την πολυπλοκότητα, βελτίωσε την απομόνωση και κατέστησε το σύστημα πιο ανθεκτικό σε σφάλματα.

Το αποτέλεσμα είναι μια εφαρμογή που συνδυάζει απλότητα χρήσης και επεκτασιμότητα. Ο χρήστης δεν χρειάζεται να γνωρίζει λεπτομέρειες εκκίνησης υπηρεσιών, καθώς όλη η διαδικασία γίνεται αυτόματα από τον server. Η συμβολή της εργασίας έγκειται στο ότι απέδειξε, με πρακτικό τρόπο, πως το Node-RED μπορεί να εξελιχθεί σε πολυχρηστικό εργαλείο με σύγχρονους μηχανισμούς ασφάλειας, διατηρώντας ταυτόχρονα τη φιλικότητα που το χαρακτηρίζει.

## 6.2. Προτάσεις για Μελλοντική Επέκταση

Παρά τα θετικά αποτελέσματα της υλοποίησης, παραμένουν αρκετές προοπτικές για μελλοντική βελτίωση και εξέλιξη. Μία σημαντική κατεύθυνση αφορά την ενσωμάτωση κεντρικού μηχανισμού αποθήκευσης των flows σε βάση δεδομένων, ώστε οι χρήστες να μπορούν να μεταφέρουν τις ροές τους μεταξύ διαφορετικών instances ή περιβαλλόντων χωρίς να απαιτείται χειροκίνητη παρέμβαση. Μια τέτοια δυνατότητα θα καθιστούσε το σύστημα πιο ευέλικτο και θα το έφερνε πιο κοντά σε μοντέλα cloud-based υπηρεσιών.

Επιπλέον, η τρέχουσα διαχείριση των ports θα μπορούσε να επεκταθεί με πιο εξελιγμένους μηχανισμούς ελέγχου, όπως η δυναμική παραχώρηση και απελευθέρωση πόρων με βάση τη χρήση, ή ακόμη και η υιοθέτηση container-based τεχνολογιών, όπως το Docker, για την πλήρη απομόνωση των instances. Αυτό θα ενίσχυε τόσο την ασφάλεια όσο και την κλιμακωσιμότητα του συστήματος.

Σε επίπεδο ασφάλειας, προτείνεται η προσθήκη εναλλακτικών μεθόδων ταυτοποίησης, όπως το OAuth2 ή η διασύνδεση με LDAP, ώστε το σύστημα να μπορεί να υιοθετηθεί και σε περιβάλλοντα όπου απαιτείται ενοποίηση με υφιστάμενες εταιρικές υποδομές. Παράλληλα, θα μπορούσε να εξεταστεί η δημιουργία μηχανισμών παρακολούθησης και ελέγχου των instances σε πραγματικό χρόνο, ώστε να υπάρχει καλύτερη εικόνα για την κατάσταση του συστήματος και την κατανομή των πόρων.

Τέλος, ιδιαίτερο ενδιαφέρον παρουσιάζει η προοπτική αξιοποίησης της εργασίας σε πιο εξειδικευμένες εφαρμογές, όπως η εκπαίδευση, η βιομηχανία και το Internet of Things. Σε αυτά τα πεδία, η ύπαρξη εξατομικευμένων Node-RED περιβαλλόντων για κάθε χρήστη θα μπορούσε να διευκολύνει τη δημιουργία, την ανάπτυξη και τη δοκιμή πολύπλοκων ροών, χωρίς να απαιτείται προηγμένη τεχνική εξειδίκευση.

## 6.3. Συνολική Συνεισφορά

Η συμβολή της παρούσας πτυχιακής εργασίας έγκειται στο ότι απέδειξε τη δυνατότητα του Node-RED να εξελιχθεί σε μια πλατφόρμα που δεν περιορίζεται σε μονοχρηστικά περιβάλλοντα αλλά μπορεί να υποστηρίξει με ασφάλεια και αποδοτικότητα πολλαπλούς χρήστες. Η χρήση μηχανισμών ταυτοποίησης με JWT, η ενσωμάτωση βάσης δεδομένων για τη διαχείριση χρηστών και η εκκίνηση αυτόνομων Node-RED instances ανά χρήστη αποτελούν ένα πλαίσιο που ενισχύει τη λειτουργικότητα και ανοίγει τον δρόμο για νέες προσεγγίσεις.

Η εργασία δεν περιορίστηκε μόνο σε θεωρητικό επίπεδο, αλλά παρείχε και μια πλήρως λειτουργική υλοποίηση, η οποία μπορεί να αξιοποιηθεί από άλλους ερευνητές και προγραμματιστές. Με τον τρόπο αυτό, αποδεικνύεται ότι η πλατφόρμα Node-RED διαθέτει τη δυναμική να ενσωματωθεί σε πιο σύνθετες εφαρμογές, ενισχύοντας το εύρος και την αξία της. Η συνολική συνεισφορά της εργασίας βρίσκεται ακριβώς σε αυτή την απόδειξη: ότι μια πλατφόρμα με ανοιχτό κώδικα και αρχικά περιορισμένες δυνατότητες μπορεί να επεκταθεί και να προσαρμοστεί σε σύγχρονες ανάγκες, προσφέροντας λύσεις που είναι ταυτόχρονα πρακτικές, ασφαλείς και επαναχρησιμοποιήσιμες.

# Βιβλιογραφικές αναφορές


[1] M. C. Loring, M. Marron και D. Leijen, «Semantics of Asynchronous JavaScript,» *13th ACM SIGPLAN International Symposium on Dynamic Languages (DLS '17),* pp. 1-12, Oct 2017.

[2] M. Shumilov, «The Use of Asynchronous Programming in Node.js to Increase the Performance of Web Services,» *International Research Journal of Modernization in Engineering Technology and Science (IRJMETS),* Vol. 6, no. 10, pp. 5177-5180, Oct 2024.

[3] S. &. V. S. Tilkov, «Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing, 14(6), 80-83,» 2010.

[4] H. Cabane και K. Farias, «On the impact of event-driven architecture on performance,» *Future Generation Computer Systems,* Vol. 153, no. 3, pp. 52-69, April 2024.

[5] J. Davis, A. Thekumparampil και D. Lee, «Node.fz: Fuzzing the Server-Side Event-Driven Architecture,» *EuroSys '17: Twelfth EuroSys Conference 2017,* pp. 145-160, April 2017.

[6] O. &. F. P. Vermesan, «Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. River Publishers,» 2014.

[7] U. Onwuegbuzie, K. A. Kayode και O. A. Olowojebutu, «Node-RED and IoT Analytics: A Real-Time Data Processing and Visualization Platform,» *Tech Sphere Multidisciplinary International Journal,* τόμ. 1, αρ. 1, September 2024.

[8] N.-R. team, «Node-RED Documentation.,» [Ηλεκτρονικό]. Available: https://nodered.org/docs/.

[9] V. Khanna και S. Kaur, «Implementation and Comparison of MQTT, WebSocket, and HTTP Protocols for Smart Room IoT Application in Node-RED,» , *IoT for Sustainable Smart Cities and Society*, Springer, pp. 165-103.

[10] C. Orłowski και M. Adrych, «Model of IoT design decision-making processes in Flow Based Programming systems,» *JOURNAL OF INFORMATION AND TELECOMMUNICATION,* Vol. 8, pp. 315-324, April 2024.

[11] D. Clerissi, M. Leotta, G. Reggio και F. Ricca, «Towards an Approach for Developing and Testing Node-RED IoT Systems,» σε *Proceedings of the 4th International Workshop on Rapid and Continuous Software Engineering (RCoSE '18)*, Gothenburg, 2018.

[12] Node-RED, «Handling errors,» [Ηλεκτρονικό]. Available: https://nodered.org/docs/user-guide/handling-errors.

[13] J. Fiaidhi και S. Mohammed, «Virtual care for cyber–physical systems (VH_CPS): NODE-RED, community of,» *Computer Communications,* Vol. 170, pp. 84-94, 2021.

[14] A. Medina-Pérez, D. Sánchez-Rodríguez και I. Alonso-González, «AnInternet of Thing Architecture Based on Message Queuing,» *Smart Cities 2021,* Vol. 4, pp. 803-818, April 2021.

[15] Node-RED Community, «Node-RED Flow Library,» [Ηλεκτρονικό]. Available: https://flows.nodered.org/node/node-red-contrib-mdashboard.

[16] FlowFuse, «Scaling Node-RED with FlowFuse: Differences between a Device Instance and a FlowFuse Instance,» [Ηλεκτρονικό]. Available: https://flowfuse.com/blog/2024/03/scaling-node-red-devices-vs-flowfuse-instance/.

[17] «Express.js Official Documentation,» [Ηλεκτρονικό]. Available: https://expressjs.com/.

[18] «JSON Web Tokens (JWT) – Introduction to JWT.,» [Ηλεκτρονικό]. Available: https://jwt.io/introduction.

# ΠΑΡΑΡΤΗΜΑ Α: ΚΩΔΙΚΑΣ NODE.JS

***node-red-server.js:***

```
const http = require('http');
const express = require('express');
const RED = require('node-red');
const path = require('path');
const jsonwebtoken = require('jsonwebtoken');
const mongoose = require('mongoose');
const { expressjwt: jwt } = require('express-jwt');
const multer = require('multer');
const { exec } = require("child_process");
const bcrypt = require('bcrypt');
const cookieParser = require('cookie-parser');
const fs = require('fs');
const crypto = require('crypto');

const app = express();
const server = http.createServer(app);
const secret = crypto.randomBytes(32).toString('hex');



//  Σύνδεση MongoDB
mongoose.connect('mongodb://localhost:27017/Authentication', {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
.then(() => console.log(' Συνδέθηκε στη MongoDB'))
.catch(err => console.error(' Σφάλμα σύνδεσης στη MongoDB:', err));

//  Μοντέλο Χρήστη
const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    jwtToken: { type: String },
```

```
    firstname: String,
    lastname: String,
    university: String,
    email: String,
    photo: String,
    port: { type: Number, unique: true }
});
const User = mongoose.model('User', userSchema);

//  Αποθήκευση εικόνας
const storage = multer.diskStorage({
    destination: (req, file, cb) => cb(null, 'uploads/'),
    filename: (req, file, cb) => cb(null, Date.now() + '-' + file.originalname)
});
const upload = multer({ storage });

//  Middleware
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());
app.use(cookieParser());

// Διαγραφή cookie κατά το restart
app.use((req, res, next) => {
  const token = req.cookies?.authToken;
  if (!token) return next();
  try {
    jsonwebtoken.verify(token, secret);
    return next();
  } catch {
    // Άκυρο/παλιό token μετά από restart , καθάρισέ το
    res.clearCookie('authToken', { path: '/', sameSite: 'Lax' });
    return next();
  }
});
```

```
//  Ανάθεση θύρας για χρήστη
const getAvailablePort = async () => {
    let port = 1881;
    while (await User.findOne({ port })) {
        port++;
    }
    return port;
};

// JWT MONO για /api
app.use(
  '/api',
  jwt({
    secret,
    algorithms: ['HS256'],
    getToken: (req) => req.cookies?.authToken
  })
);

//  Εγγραφή
app.post('/register', upload.single('photo'), async (req, res) => {
    const { username, password, firstname, lastname, university, email } = req.body;
    const photoPath = req.file ? req.file.filename : null;

    if (!username || !password || !firstname || !lastname || !university || !email) {
        return res.status(400).json({ message: 'Όλα τα πεδία είναι υποχρεωτικά.' });
    }

    try {
        if (await User.findOne({ username })) {
            return res.status(400).json({ message: 'Το όνομα χρήστη υπάρχει ήδη.' });
        }

        const port = await getAvailablePort();
        const hashedPassword = await bcrypt.hash(password, 10);
```

```
        const newUser = new User({
            username,
            password: hashedPassword,
            firstname,
            lastname,
            university,
            email,
            photo: photoPath,
            port
        });

        await newUser.save();
        console.log(` Νέος χρήστης στη θύρα ${port}:`, newUser);
        res.json({ success: true, redirect: "/login.html" });

    } catch (error) {
        console.error(' Σφάλμα κατά την εγγραφή:', error);
        res.status(500).json({ message: 'Σφάλμα στον server.' });
    }
});

// Login & Εκκίνηση Node-RED
app.post('/login', async (req, res) => {
    const { username, password } = req.body;
    console.log(` Σύνδεση για: ${username}`);

    // Αν υπάρχει ήδη έγκυρο cookie, μην αφήνεις login άλλου χρήστη
const existing = req.cookies?.authToken;
if (existing) {
  try {
    const decoded = jsonwebtoken.verify(existing, secret);
    if (decoded?.user && decoded.user !== req.body.username) {
      return res.status(409).json({
        success: false,
        message: `Είσαι ήδη συνδεδεμένος ως ${decoded.user}.`
      });
```

```
    }
  } catch (_) {
    // Αν είναι άκυρο ή ληγμένο token, αγνόησέ το και προχώρα στο κανονικό login
  }
}



    try {
        const user = await User.findOne({ username });
        if (!user || !(await bcrypt.compare(password, user.password))) {
            return res.status(401).json({ message: 'Λάθος όνομα χρήστη ή κωδικός.'
});
        }

        const token = jsonwebtoken.sign({ user: username }, secret, { expiresIn: "1h"
});
        res.cookie("authToken", token, { httpOnly: true , path: '/', sameSite:
'Lax'});
        user.jwtToken = token;
        await user.save();



        const userDir = `./users/${username}`;
        const settingsPath = `${userDir}/settings.js`;
        const flowFile = 'flows.json';

        const userData = {
            username: user.username,
            firstname: user.firstname,
            lastname: user.lastname,
            university: user.university,
            email: user.email,
            role: 'user',
            port: user.port
        };

        if (!fs.existsSync(userDir)) {
```

```
            fs.mkdirSync(userDir, { recursive: true });
        }

        // Δημιουργία settings.js
const settingsContent = `
const cookie = require("cookie");
const jwt = require("jsonwebtoken");

// ίδιο με του Express στην τρέχουσα εκκίνηση
const SECRET = "${secret}";

module.exports = {
  // Admin endpoints (editor, /auth/*) + συγχρονισμένο logout
  httpAdminMiddleware: function (req, res, next) {
    try {
      // logout του Node-RED → καθάρισε cookies
      if (req.method === "POST" && req.path === "/auth/revoke") {
        res.clearCookie("authToken", { path: "/", sameSite: "Lax" });
        res.clearCookie("node_red_auth", { path: "/" });
        res.clearCookie("NR-Auth", { path: "/" });
        return next();
      }

      // Έλεγχος JWT για ΟΛΑ τα admin endpoints
      const cookies = cookie.parse(req.headers.cookie || "");
      const token = cookies.authToken;
      if (!token) {
        res.clearCookie("authToken", { path: "/", sameSite: "Lax" });
        return res.status(401).send("JWT απαιτείται. Κάνε login.");
      }

      jwt.verify(token, SECRET);
      return next();
    } catch (e) {
      // άκυρο/παλιό μετά από restart , καθάρισμα
      res.clearCookie("authToken", { path: "/", sameSite: "Lax" });
```

```
      return res.status(401).send("Μη έγκυρο ή ληγμένο JWT.");
    }
  },

  // http nodes & Dashboard (/ui, socket.io)
  httpNodeMiddleware: function (req, res, next) {
    try {
      const cookies = cookie.parse(req.headers.cookie || "");
      const token = cookies.authToken;
      if (!token) {
        res.clearCookie("authToken", { path: "/", sameSite: "Lax" });
        return res.status(401).json({ ok: false, error: "JWT required" });
      }

      req.user = jwt.verify(token, SECRET);
      return next();
    } catch (e) {
      res.clearCookie("authToken", { path: "/", sameSite: "Lax" });
      return res.status(401).json({ ok: false, error: "Invalid/expired JWT" });
    }
  },

  functionGlobalContext: {
  user: ${JSON.stringify(userData, null, 2)},
  sessionData: {
    loginTime: ${Date.now()},
    sessionId: "${crypto.randomUUID()}"
  }
}
};
`;
```

```
        fs.writeFileSync(settingsPath, settingsContent);

        exec(`node-red -p ${user.port} --userDir ${userDir} --settings
${settingsPath} --flowFile ${flowFile}`, (error) => {
            if (error) {
                console.error(` Σφάλμα εκκίνησης Node-RED: ${error}`);
                return res.status(500).json({ message: 'Σφάλμα εκκίνησης Node-RED'
});
            }
            console.log(` Node-RED εκτελείται στη θύρα ${user.port} για
${username}`);
        });

        res.json({ success: true, redirect: `http://localhost:${user.port}/` });

    } catch (error) {
        console.error(' Σφάλμα σύνδεσης:', error);
        res.status(500).json({ message: 'Σφάλμα στον server.' });
    }
});

//  Προστατευμένο API: Προφίλ Χρήστη
app.get('/api/profile', async (req, res) => {
    try {
        const username = req.auth.user;
        const user = await User.findOne({ username }).select('-password -jwtToken');
        if (!user) return res.status(404).json({ success: false, message: 'User not
found' });
        res.json({ success: true, user });
    } catch (err) {
        console.error(' /api/profile error:', err);
        res.status(500).json({ success: false, message: 'Σφάλμα στον server.' });
    }
});



//  Ρύθμιση Node-RED
const settings = {
```

```
    userDir: path.join(__dirname, '.node-red'),
    functionGlobalContext: {}
};
RED.init(server, settings);
app.use(RED.httpAdmin);
app.use(RED.httpNode);



//  Εκκίνηση server
server.listen(1880, async () => {
    console.log(' Server τρέχει στο http://localhost:1880/');
    const { default: open } = await import('open');
    open('http://localhost:1880/login.html');
});

//  Εκκίνηση Node-RED (κεντρικό)
RED.start();

//  Ανακατεύθυνση dashboard
app.get('/redirect', (req, res) => {
    res.redirect('/ui');
});
```

# ΠΑΡΑΡΤΗΜΑ Β: ΚΩΔΙΚΑΣ HTML

***login.html:***

```
<!DOCTYPE html>
<html lang="el">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Σύνδεση</title>
  <style>
    body {
      background-color: #f1f4fa;
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
    .container {
      max-width: 400px;
      margin: 100px auto;
      background-color: #ffffff;
      padding: 40px;
      border-radius: 10px;
      box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
      text-align: center;
    }
    h2 {
      color: #2b4ca0;
      margin-bottom: 20px;
    }
    input[type="text"],
    input[type="password"] {
      width: 100%;
      padding: 12px;
      margin: 10px 0;
```

```
      border: 1px solid #ccc;
      border-radius: 6px;
      font-size: 14px;
    }
    button {
      width: 100%;
      padding: 12px;
      background-color: #3458d3;
      color: #fff;
      border: none;
      border-radius: 6px;
      font-size: 16px;
      cursor: pointer;
    }
    button:hover {
      background-color: #2b4ca0;
    }
    p {
      margin-top: 20px;
      font-size: 14px;
    }
    a {
      color: #3458d3;
      text-decoration: none;
    }
    a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>Καλώς ήρθατε στο Node-RED Dashboard</h2>
    <p>Παρακαλώ συνδεθείτε με τα στοιχεία σας</p>
    <form id="login-form">
      <input type="text" id="username" placeholder="Username" required>
```

```
      <input type="password" id="password" placeholder="Password" required>
      <button type="submit">Σύνδεση</button>
    </form>
    <p>Δεν έχετε λογαριασμό; <a href="register.html">Εγγραφείτε εδώ</a></p>
  </div>

  <script>
    document.getElementById("login-form").addEventListener("submit", async
function(event) {
      event.preventDefault();

      const username = document.getElementById("username").value;
      const password = document.getElementById("password").value;

      try {
        const response = await fetch("/login", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ username, password })
        });

        const data = await response.json();
        if (data.success) {
          // στείλε στο welcome.html με query param το URL του instance
          const target = encodeURIComponent(data.redirect);
          window.location.href = `/welcome.html?to=${target}`;
        } else {
          alert("Σφάλμα: " + data.message);
        }
      } catch (error) {
        console.error("Error:", error);
        alert("Παρουσιάστηκε σφάλμα κατά τη σύνδεση.");
      }
    });
  </script>
</body>
</html>
```

***register.html:***

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f4f4f4;
        }
        form {
            background: white;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        input[type="text"], input[type="password"], input[type="email"],
input[type="file"] {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 3px;
        }
        input[type="submit"] {
            background: #28a745;
            color: white;
```

```
            border: none;
            padding: 10px;
            border-radius: 3px;
            cursor: pointer;
            width: 100%;
        }
        .requirements {
            font-size: 0.9em;
            color: #555;
            margin-top: -5px;
            margin-bottom: 15px;
        }
    </style>
</head>
<body>

    <form id="registrationForm">
        <h2>Register</h2>

        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
        <div class="requirements">Password must be 6-12 characters long, contain at
least one uppercase letter and one number.</div>

        <label for="firstname">First Name:</label>
        <input type="text" id="firstname" name="firstname" required>

        <label for="lastname">Last Name:</label>
        <input type="text" id="lastname" name="lastname" required>

        <label for="university">University:</label>
        <input type="text" id="university" name="university" required>

        <label for="email">Email:</label>
```

```
        <input type="email" id="email" name="email" required>
        <div class="requirements">Please enter a valid email address.</div>

        <label for="photo">Upload Photo:</label>
        <input type="file" id="photo" name="photo" accept="image/*">

        <input type="submit" value="Register">
    </form>

    <script>
        document.getElementById('registrationForm').addEventListener('submit', async
(event) => {
            event.preventDefault(); // Prevent the default form submission

            const formData = new FormData(event.target); // Get the form data
 try {
                const response = await fetch('/register', {
                    method: 'POST',
                    body: formData
                });

                if (response.ok) {
                    // Successful registration
                    alert('Registration successful! Redirecting to login...');
                    window.location.href = '/login.html'; // Redirect to login page
                } else {
                    const errorData = await response.json();
                    alert('Error: ' + errorData.message); // Show error message
                }
            } catch (error) {
                console.error('Error during registration:', error);
                alert('An unexpected error occurred. Please try again.');
            }
        });
    </script>
</body>
</html>
```

***welcome.html:***

```
<!DOCTYPE html>
<html lang="el">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Καλωσήρθες</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f4f4f4;
      display: flex;
      align-items: center;
      justify-content: center;
      height: 100vh;
      margin: 0;
      color: #333;
    }
    .card {
      background: #fff;
      padding: 28px 32px;
      border-radius: 10px;
      box-shadow: 0 8px 24px rgba(0,0,0,.08);
      text-align: center;
      max-width: 420px;
    }
    h1 {
      margin: 0 0 10px;
      color: #4CAF50;
    }
    p {
      margin: 6px 0;
      font-size: 15px;
    }
    .small {
```

```
      color: #777;
      font-size: 14px;
      margin-top: 12px;
    }
  </style>
</head>
<body>
  <div class="card">
    <h1>Welcome, <span id="username">...</span>!</h1>
    <p>Η σύνδεσή σου επιβεβαιώθηκε.</p>
    <p class="small">Σε λίγα δευτερόλεπτα θα μεταφερθείς στο προσωπικό σου
dashboard.</p>
    <p class="small">Αν δεν γίνει αυτόματα, <a id="dashLink" href="/">πάτησε
εδώ</a>.</p>
  </div>

  <script>
    // Διάβασε το ?to=<instance-url> από το query string
    const params = new URLSearchParams(window.location.search);
    const target = params.get('to') || '/';
    document.getElementById('dashLink').href = target;

    // Ζήτα το προφίλ για να εμφανίσεις το username (JWT cookie αποστέλλεται
αυτόματα)
    fetch('/api/profile')
      .then(r => {
        if (!r.ok) throw new Error('unauthorized');
        return r.json();
      })
      .then(data => {
        document.getElementById('username').textContent = data?.user?.username ||
'User';
        // Ανακατεύθυνση μετά από 2 δευτερόλεπτα
        setTimeout(() => { window.location.replace(target); }, 2000);
      })
      .catch(() => {
        // Αν δεν υπάρχει έγκυρο cookie → επιστροφή στο login
        window.location.replace('/login.html');
```

```
      });
  </script>
</body>
</html>
```

***profile.html:***

```
<!DOCTYPE html>
<html lang="el">
<head>
  <meta charset="UTF-8">
  <title>Προφίλ Χρήστη</title>
  <style>
    body { font-family: sans-serif; margin: 2em; }
    .profile { border: 1px solid #ccc; padding: 1em; border-radius: 8px; max-width:
400px; }
    .label { font-weight: bold; }
  </style>
</head>
<body>

  <h1>Προφίλ Χρήστη</h1>
  <div id="profile" class="profile">Φόρτωση...</div>

  <script>
    fetch('/api/profile')
      .then(res => {
        if (!res.ok) throw new Error('Unauthorized');
        return res.json();
      })
      .then(data => {
        if (!data.success) throw new Error('Unauthorized');

        const user = data.user;
        document.getElementById('profile').innerHTML = `
          <p><span class="label">Χρήστης:</span> ${user.username}</p>
          <p><span class="label">Όνομα:</span> ${user.firstname} ${user.lastname}</p>
          <p><span class="label">Email:</span> ${user.email}</p>
          <p><span class="label">Πανεπιστήμιο:</span> ${user.university}</p>
          <p><span class="label">Port Node-RED:</span> ${user.port}</p>
```

```
        `;
      })
      .catch(err => {
        // Αν δεν είναι login, redirect στο login
        alert('Δεν έχετε συνδεθεί.');
        window.location.href = '/login.html';
      });
  </script>

</body>
</html>
```

# ΠΑΡΑΡΤΗΜΑ Γ: ΚΩΔΙΚΑΣ DOCKER ΚΑΙ BATCH

***Dockerfile:***

```
FROM node:20-bullseye

# build tools για bcrypt κ.λπ.
RUN apt-get update && apt-get install -y --no-install-recommends \
    python3 make g++ && rm -rf /var/lib/apt/lists/*

# node-red CLI global ώστε να υπάρχει το binary "node-red"
RUN npm install -g --unsafe-perm node-red

WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev

COPY . .
RUN mkdir -p uploads users public

EXPOSE 1880
CMD ["node", "node-red-server.js"]
```

***docker-compose.yml:***

```
version: "3.9"
services:
  mongo:
    image: mongo:6
    restart: unless-stopped
    volumes:
      - mongo_data:/data/db
    ports:
      - "1880:1880"
      - "1881-1980:1881-1980"


  app:
    build: .
    restart: unless-stopped
    depends_on:
      - mongo
    # Μοιράζεται το network με το mongo ⇒ το "mongodb://localhost:27017" δουλεύει όπως είναι.
    network_mode: "service:mongo"
    volumes:
      - app_users:/app/users
      - app_uploads:/app/uploads
volumes:
  mongo_data:
  app_users:
  app_uploads:
```

***Start.bat***:

```
@echo off
set shortcut="%USERPROFILE%\Desktop\MyApp.lnk"
echo Set oWS = WScript.CreateObject("WScript.Shell") > "%temp%\make.vbs"
echo sLinkFile = %shortcut% >> "%temp%\make.vbs"
echo Set oLink = oWS.CreateShortcut(sLinkFile) >> "%temp%\make.vbs"
echo oLink.TargetPath = "%~dp0Start.bat" >> "%temp%\make.vbs"
echo oLink.IconLocation = "shell32.dll, 43" >> "%temp%\make.vbs"
echo oLink.Save >> "%temp%\make.vbs"
cscript /nologo "%temp%\make.vbs"
del "%temp%\make.vbs"

echo Εκκίνηση εφαρμογής...
docker compose up -d
timeout /t 5 >nul
start http://localhost:1880/login.html
```

***Stop.bat:***

```
@echo off

cd /d "%~dp0"

docker compose down

pause
```