Ανάπτυξη Συστήματος Διαχείρισης Έργων (Project Management System) με Microservices σε Περιβάλλον Docker

Μαβιτζής Σταμάτιος
2018030040
University Department: TUC - ECE
Supervising Professor: ΕΥΡΙΠΙΔΗΣ ΠΕΤΡΑΚΗΣ

Υπηρεσίες στο Υπολογιστικό Νέφος και την Ομίχλη (ΠΛΗ513)
October 26, 2025

Contents

1. Εισαγωγή

Η παρούσα εργασία εκπονήθηκε στο πλαίσιο του μαθήματος **Υπηρεσίες στο Υπολογιστικό Νέφος και την Ομίχλη (ΠΛΗ513)** και έχει ως στόχο την ανάπτυξη ενός **Συστήματος Διαχείρισης Έργων (Project Management System - PMS)** βασισμένου σε αρχιτεκτονική **Microservices** και υλοποιημένου σε περιβάλλον **Docker**.

Η εφαρμογή σχεδιάστηκε ως ένα απλοποιημένο σύστημα τύπου *Jira* ή *Trello*, επιτρέποντας τη διαχείριση ομάδων, χρηστών και εργασιών με τρόπο οργανωμένο και εύχρηστο. Στόχος ήταν η κατανόηση και πρακτική εφαρμογή σύγχρονων τεχνολογιών web development και cloud deployment, συνδυάζοντας έννοιες όπως **REST APIs**, **authentication**, **containerization** και **multi-role access control**.

Το σύστημα υλοποιήθηκε με τη χρήση **Python (Flask framework)** για το backend, **PostgreSQL** για τη βάση δεδομένων και **HTML/CSS/JavaScript** για το frontend interface. Η ανάπτυξη βασίστηκε στη λογική των microservices, όπου κάθε βασική λειτουργία (π.χ. διαχείριση χρηστών, ομάδων, εργασιών) υλοποιείται ως ανεξάρτητο service με δικό του API.

Κάθε χρήστης της εφαρμογής έχει διαφορετικό ρόλο και δικαιώματα:

- Ο **Διαχειριστής (Admin)** μπορεί να δημιουργεί ομάδες, να ενεργοποιεί χρήστες και να αλλάζει ρόλους.
- Ο **Αρχηγός Ομάδας (Team Leader)** δημιουργεί εργασίες, τις αναθέτει στα μέλη και παρακολουθεί την πρόοδο.
- Το **Μέλος (Member)** βλέπει τις εργασίες του, ενημερώνει την κατάστασή τους και σχολιάζει την εξέλιξη.

Τέλος, η όλη υποδομή αναπτύχθηκε με σκοπό να μπορεί να «σηκωθεί» και να εκτελεστεί μέσω **Docker containers**, προσφέροντας φορητότητα, επεκτασιμότητα και ευκολία ανάπτυξης σε διαφορετικά περιβάλλοντα.

Η εργασία αυτή συνδυάζει έννοιες προγραμματισμού, βάσεων δεδομένων, cloud υπηρεσιών και σύγχρονων τεχνικών ανάπτυξης λογισμικού, παρέχοντας μια ολοκληρωμένη πρακτική εμπειρία στην ανάπτυξη web εφαρμογών με **microservices** αρχιτεκτονική.

2. Αρχιτεκτονική του Συστήματος

2.1 Συνολική Αρχιτεκτονική

Το σύστημα έχει υλοποιηθεί ως **ενιαία Flask εφαρμογή** που εκτελείται στη θύρα **5000**, με τη λογική αρχιτεκτονικής **modular monolith**. Αντί να διαχωρίζεται σε ανεξάρτητα microservices, η εφαρμογή χωρίζεται σε **Blueprints** — δηλαδή αυτόνομα modules, καθένα υπεύθυνο για ένα συγκεκριμένο λειτουργικό κομμάτι: διαχείριση χρηστών, ομάδων, εργασιών και εμφάνιση διεπαφής.

Όλα τα modules χρησιμοποιούν κοινή **PostgreSQL** βάση δεδομένων μέσω του αρχείου `db.py`, το οποίο χρησιμοποιεί τη βιβλιοθήκη `psycopg2`. Η παραμετροποίηση της σύνδεσης βρίσκεται στο αρχείο `config.py`. Η αυθεντικοποίηση και διαχείριση συνδέσεων γίνεται με `Flask sessions`, και τα μηνύματα προς τον χρήστη προβάλλονται με `flash()`.

Module / Blueprint	**Ρόλος - Ευθύνη**	**Αρχεία Κώδικα**	**Port**
User Management	Διαχείριση χρηστών, αυθεντικοποίηση, εξουσιοδότηση ρόλων (Admin, Team Leader, Member)	`admin_authenticate.py`, `teamLeader_authenticate.py`, `member_authenticate.py`	5000
Admin Management	Διαχείριση χρηστών και ομάδων, αλλαγή ρόλων, ενεργοποιήσεις/απενεργοποιήσεις	`admin_mainpage.py`, `admin_teamLeader_member_options.py`	5000
Team Management	Δημιουργία, προβολή και διαχείριση ομάδων, προσθήκη/αφαίρεση μελών	`teamLeader_mainpage.py`	5000
Task Management	Διαχείριση εργασιών (tasks), κατάστασης (TODO / IN_PROGRESS / DONE), σχολίων και προθεσμιών	`teamLeader_mainpage.py`, `member_mainpage.py`	5000
Frontend (UI)	Παρουσίαση διεπαφής, routing, templates για όλους τους ρόλους	`homepage.py`, φάκελος `templates/`	5000
Database Layer	Σύνδεση με βάση PostgreSQL	`db.py`, `config.py`	-

Table 2.1: Blueprints και Αντίστοιχες Ευθύνες στο Σύστημα

2.2 Περιγραφή των Modules

2.2.1 User Management

Αρχεία: `admin_authenticate.py`, `teamLeader_authenticate.py`, `member_authenticate.py`
Υλοποιεί τη διαχείριση σύνδεσης και εγγραφής χρηστών. Ελέγχει ρόλους και δημιουργεί sessions για κάθε login. Κάθε χρήστης καταχωρείται στη βάση στον πίνακα `users` με ρόλο `ADMIN`, `TEAM_LEADER` ή `MEMBER`.

Ενδεικτικά endpoints:

```
POST /admin-login
POST /teamLeader-login
POST /member-login
POST /admin-signup
POST /teamLeader-signup
```

Χρησιμοποιούμενα templates: `admin_login.html`, `teamLeader_login.html`, `member_login.html`

2.2.2 Admin Management

Αρχεία: `admin_mainpage.py`, `admin_teamLeader_member_options.py`
Ο διαχειριστής μπορεί να ενεργοποιεί/απενεργοποιεί χρήστες, να αλλάζει ρόλους, να δημιουργεί ομάδες και να διαγράφει υπάρχουσες. Η επικοινωνία με τη βάση γίνεται μέσω `RealDictCursor` για άμεση χρήση των αποτελεσμάτων στα templates.

Ενδεικτικά endpoints:

```
GET /admin-mainpage
POST /activate_user/<username>
POST /deactivate_user/<username>
POST /change_role/<username>
GET /admin-manageTeams
POST /admin-deleteTeam/<team_id>
```

Χρησιμοποιούμενα templates: `admin_mainpage.html`, `admin_manageTeams.html`, `admin_viewTeam.html`

2.2.3 Team Management (Team Leader)

Αρχείο: `teamLeader_mainpage.py`
Ο αρχηγός ομάδας δημιουργεί, προβάλλει και επεξεργάζεται τις ομάδες του. Μπορεί να προσθέτει ή να αφαιρεί μέλη και να αναθέτει εργασίες.

Ενδεικτικά endpoints:

```
GET /teamLeader-mainpage
GET /teamLeader-manageTeams
GET /teamLeader-manageTeam/<team_id>
POST /teamLeader-addMember/<team_id>
POST /teamLeader-removeMember/<team_id>
```

Χρησιμοποιούμενα templates: `teamLeader_mainpage.html`, `teamLeader_manageTeams.html` `teamLeader_teamDetails.html`

2.2.4 Task Management

Αρχεία: `teamLeader_mainpage.py`, `member_mainpage.py`
Υλοποιεί την πλήρη διαχείριση εργασιών: δημιουργία, επεξεργασία, αλλαγή κατάστασης, σχόλια, deadlines. Οι εργασίες αποθηκεύονται στον πίνακα `tasks`, ενώ τα σχόλια στον πίνακα `comments`.

Ενδεικτικά endpoints:

```
POST /teamLeader-createTask/<team_id>
GET  /teamLeader-viewTask/<task_id>
POST /teamLeader-editTask/<task_id>
POST /teamLeader-addComment/<task_id>
POST /member-addCommentPage/<task_id>
```

Χρησιμοποιούμενα templates: `teamLeader_createTask.html`, `teamLeader_viewTask.html`, `teamLeader_editTask.html`, `teamLeader_addComment.html`, `member_viewTasks.html`, `member_addComment.html`, `member_notifications_and_deadlines.html`

2.2.5 Frontend (UI)

Αρχεία: `homepage.py`, `admin_teamLeader_member_options.py`
Το UI παρέχει τις αρχικές σελίδες και δρομολογεί τον χρήστη στο κατάλληλο Blueprint ανάλογα με τον ρόλο του. Η εμφάνιση βασίζεται σε templates και χρήση `flash` για ειδοποιήσεις.

Χρησιμοποιούμενα templates: `index.html`, `admin_teamLeader_member_options.html`, `layout.html`, `base.html`, `style.css` (στον φάκελο `static/css`)

2.2.6 Βάση Δεδομένων

Αρχεία: `db.py`, `config.py`, `create_tables.sql`
Η επικοινωνία με τη βάση PostgreSQL γίνεται μέσω της συνάρτησης:

```
get_db_connection()
```

Το αρχείο `create_tables.sql` δημιουργεί τους πίνακες: `users`, `teams`, `team_members`, `tasks`, `comments`.

2.3 Επικοινωνία και Ροές

Μοντέλο Επικοινωνίας:

- Όλες οι κλήσεις γίνονται μέσω HTTP (routes του ίδιου Flask app).
- Τα Blueprints επικοινωνούν εσωτερικά (in-process), όχι μέσω REST μεταξύ τους.
- Η Flask εφαρμογή επικοινωνεί με τη βάση μέσω `psycopg2`.
- Τα δεδομένα επιστρέφονται ως templates ή JSON responses.

Ενδεικτική Ροή:

1. Ο Team Leader συνδέεται μέσω `/teamLeader-login`.
2. Δημιουργεί νέα ομάδα στο `/teamLeader-manageTeams`.
3. Δημιουργεί εργασία (`/teamLeader-createTask/<team_id>`).
4. Το μέλος βλέπει το task στο `/member-viewTasks`.
5. Το μέλος προσθέτει σχόλιο στο `/member-addCommentPage/<task_id>`.

2.4 Διάγραμμα Αρχιτεκτονικής

Το διάγραμμα παρουσιάζει την οργάνωση των Blueprints, τη σύνδεση με τη βάση PostgreSQL και το ενιαίο Flask app.

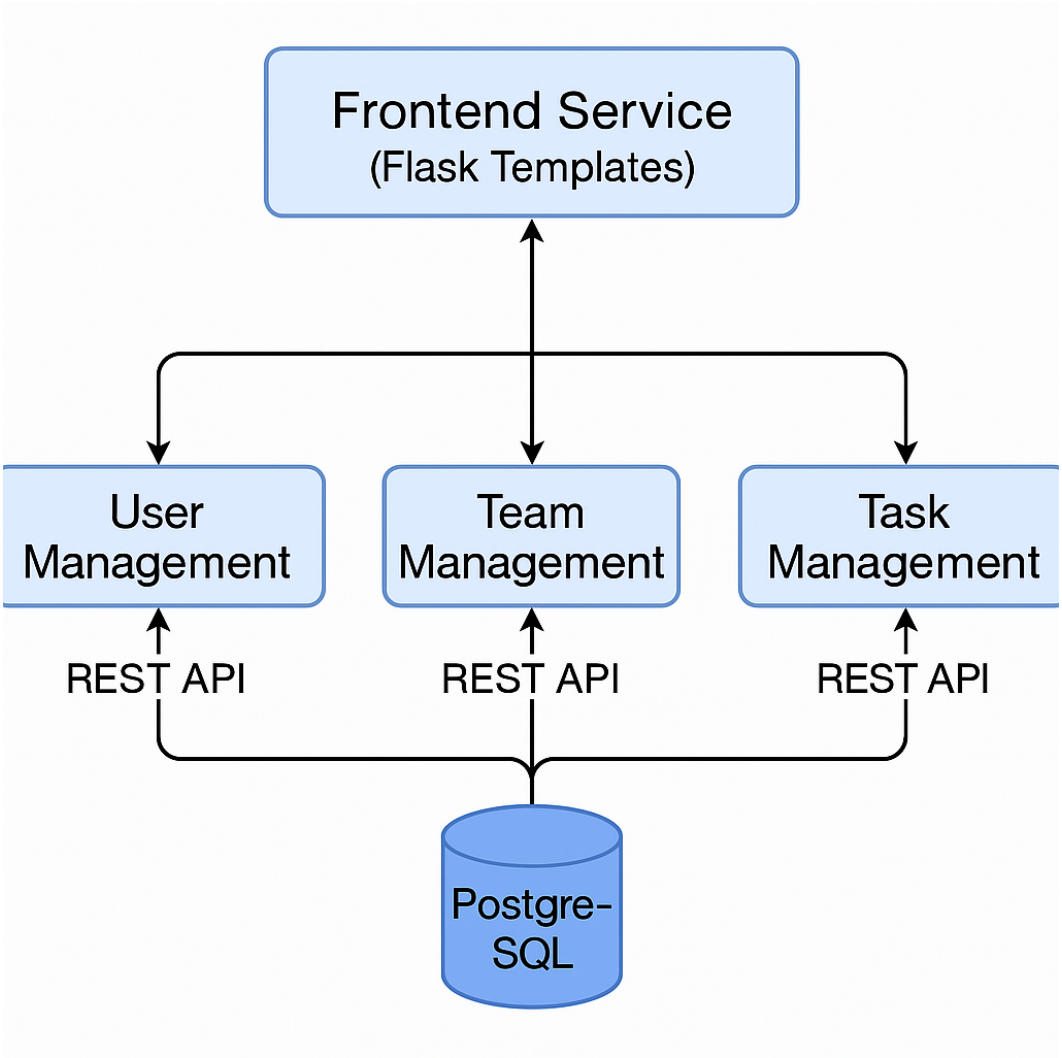

Figure 2.1: Αρχιτεκτονική Εφαρμογής Flask με Blueprints (Port 5000) και PostgreSQL

2.5 Πλεονεκτήματα

- **Καθαρός διαχωρισμός ευθυνών:** κάθε Blueprint διαχειρίζεται ένα λειτουργικό κομμάτι.
- **Ευκολία επεκτασιμότητας:** νέα modules μπορούν να προστεθούν χωρίς να επηρεάσουν τα υπάρχοντα.
- **Απλοποιημένη συντήρηση:** κάθε module έχει δικά του routes και templates.

- **Ενοποιημένη ανάπτυξη:** όλο το σύστημα εκτελείται σε ένα Flask app, στην πόρτα 5000.
- **Σαφής ροή δεδομένων:** όλες οι λειτουργίες βασίζονται σε SQL queries και templates.

3. Περιγραφή Βάσης Δεδομένων

Η βάση δεδομένων του συστήματος σχεδιάστηκε με γνώμονα τη σαφή οργάνωση των δεδομένων, τη διατήρηση της ακεραιότητας και την ευκολία επέκτασης στο πλαίσιο μιας microservices αρχιτεκτονικής. Χρησιμοποιήθηκε η PostgreSQL, μια σχεσιακή βάση δεδομένων ανοιχτού κώδικα, η οποία εξασφαλίζει υψηλή απόδοση, ασφάλεια και σταθερότητα, καθώς και πλήρη υποστήριξη για σχέσεις και συναλλαγές.

3.1 Σύνδεση με τη Βάση

Η επικοινωνία των υπηρεσιών με τη βάση δεδομένων πραγματοποιείται μέσω του αρχείου `db.py`, το οποίο υλοποιεί τη συνάρτηση `get_db_connection()`. Η συνάρτηση αυτή δημιουργεί μια σύνδεση με τη βάση PostgreSQL αξιοποιώντας τις ρυθμίσεις του αρχείου `config.py`, το οποίο περιέχει τα απαραίτητα στοιχεία (όνομα βάσης, χρήστη, κωδικό και host):

```
DB_CONFIG = {
    "host": "localhost",
    "database": "postgres",
    "user": "postgres",
    "password": "xotour"
}
```

Listing 3.1: Ρύθμιση σύνδεσης στη βάση δεδομένων

Κάθε microservice (όπως το `admin_mainpage`, `teamLeader_mainpage`, `member_mainpage`) διατηρεί τη δική του σύνδεση με τη βάση, ακολουθώντας το πρότυπο της απομονωμένης πρόσβασης (database-per-service), το οποίο ενισχύει την ανεξαρτησία και την ασφάλεια των υπηρεσιών.

3.2 Σχεσιακό Μοντέλο

Η βάση δεδομένων αποτελείται από πέντε κύριους πίνακες που αντανακλούν τις βασικές οντότητες του συστήματος: χρήστες, ομάδες, μέλη ομάδων, εργασίες και σχόλια. Η μεταξύ τους σχέση είναι ιεραρχικά και λογικά οργανωμένη ώστε να υποστηρίζει τις λειτουργίες δημιουργίας, ανάθεσης και παρακολούθησης έργων.

Table 3.1: Κύριες οντότητες της βάσης δεδομένων

Οντότητα	**Περιγραφή**
`users`	Περιλαμβάνει πληροφορίες για όλους τους χρήστες (Admin, Team Leaders, Members).
`teams`	Αντιπροσωπεύει τις ομάδες έργου που διαχειρίζονται οι χρήστες.
`team_members`	Συνδέει τα μέλη με τις ομάδες (πολλοί-προς-πολλούς).
`tasks`	Περιλαμβάνει τις εργασίες (tasks) που ανατίθενται στα μέλη κάθε ομάδας.
`comments`	Αποθηκεύει τα σχόλια των χρηστών σχετικά με κάθε εργασία.

3.3 Διάγραμμα Σχέσεων (ER Diagram)

Η λογική σχέση των πινάκων μπορεί να απεικονιστεί ως εξής:

```
users (1) ---> teams (N)
users (N) ---> team_members (N:M) <--- teams (N)
teams (1) ---> tasks (N)
tasks (1) ---> comments (N)
```

3.4 Αναλυτική Περιγραφή Πινάκων

3.4.1 Πίνακας users

Table 3.2: Πεδία του πίνακα users

Πεδίο	**Τύπος**	**Περιγραφή**
user_id	SERIAL PRIMARY KEY	Μοναδικό αναγνωριστικό χρήστη.
username	VARCHAR(50)	Όνομα χρήστη.
email	VARCHAR(100) UNIQUE	Ηλεκτρονική διεύθυνση.
password	VARCHAR(100)	Κωδικός πρόσβασης.
name	VARCHAR(100)	Όνομα.
surname	VARCHAR(100)	Επώνυμο.
role	ENUM('ADMIN','TEAM_LEADER','MEMBER')	Ρόλος χρήστη.
status	ENUM('ACTIVE','INACTIVE')	Κατάσταση ενεργοποίησης.

3.4.2 Πίνακας teams

Table 3.3: Πεδία του πίνακα teams

team_id	SERIAL PRIMARY KEY	Αναγνωριστικό ομάδας.
name	VARCHAR(100)	Όνομα ομάδας.
description	TEXT	Περιγραφή της ομάδας.
leader_id	INTEGER (FK → users.user_id)	Ο αρχηγός της ομάδας.
created_at	TIMESTAMP	Ημερομηνία δημιουργίας.

3.4.3 Πίνακας team_members

Table 3.4: Πεδία του πίνακα team_members

team_id	INTEGER (FK → teams.team_id)	Η ομάδα στην οποία ανήκει το μέλος.
user_id	INTEGER (FK → users.user_id)	Ο χρήστης που συμμετέχει.
PRIMARY KEY	(team_id, user_id)	Σύνθετο κλειδί για μοναδικότητα εγγραφών.

3.4.4 Πίνακας tasks

Table 3.5: Πεδία του πίνακα tasks

task_id	SERIAL PRIMARY KEY	Αναγνωριστικό εργασίας.
title	VARCHAR(100)	Τίτλος της εργασίας.
description	TEXT	Αναλυτική περιγραφή.
created_by	INTEGER (FK → users.user_id)	Δημιουργός της εργασίας.
assigned_to	INTEGER (FK → users.user_id)	Ο χρήστης στον οποίο έχει ανατεθεί.
team_id	INTEGER (FK → teams.team_id)	Η ομάδα στην οποία ανήκει.
status	ENUM('TODO','IN_PROGRESS','DONE')	Κατάσταση εργασίας.
priority	ENUM('LOW','MEDIUM','HIGH')	Προτεραιότητα.
due_date	DATE	Προθεσμία ολοκλήρωσης.
created_at	TIMESTAMP	Ημερομηνία δημιουργίας.

3.4.5 Πίνακας comments

Table 3.6: Πεδία του πίνακα comments

comment_id	SERIAL PRIMARY KEY	Αναγνωριστικό σχολίου.
task_id	INTEGER (FK → tasks.task_id)	Η εργασία στην οποία αναφέρεται.
author_id	INTEGER (FK → users.user_id)	Ο χρήστης που το έγραψε.
content	TEXT	Το περιεχόμενο του σχολίου.
created_at	TIMESTAMP	Ημερομηνία καταχώρησης.

3.5 Κανόνες Ακεραιότητας και Περιορισμοί

- Ξένα κλειδιά (Foreign Keys) για τη σωστή αντιστοίχιση μεταξύ χρηστών, ομάδων, εργασιών και σχολίων.
- Σύνθετα κλειδιά (Composite Keys) στον πίνακα `team_members` ώστε να αποτρέπεται η διπλή εγγραφή.
- Περιορισμοί ENUM στα πεδία `role`, `status`, `priority` και `task status` για διατήρηση συνέπειας.
- ON DELETE CASCADE ώστε η διαγραφή ομάδας να επιφέρει αυτόματα διαγραφή σχετικών μελών, εργασιών και σχολίων.

3.6 Συμπέρασμα

Η βάση δεδομένων εξασφαλίζει τη σωστή λειτουργία όλων των microservices, παρέχοντας ασφάλεια, συνοχή και ευκολία συντήρησης. Με αυτόν τον τρόπο υποστηρίζεται πλήρως η λειτουργία της πλατφόρμας διαχείρισης έργων, όπως η δημιουργία ομάδων, η ανάθεση εργασιών και η επικοινωνία μεταξύ χρηστών.

4. Περιγραφή Λειτουργιών (ανά ρόλο)

4.1 Ρόλος Διαχειριστή (Admin)

Ο **Διαχειριστής Συστήματος (Admin)** διαθέτει τον πλήρη έλεγχο της εφαρμογής. Έχει τη δυνατότητα διαχείρισης χρηστών, δημιουργίας και διαγραφής ομάδων, καθώς και εποπτείας όλων των έργων και εργασιών.

Βασικές Λειτουργίες

- **Εγγραφή και Είσοδος:** Μέσω `/admin-signup` και `/admin-login`.
- **Πίνακας Ελέγχου:** `/admin-mainpage` — προβολή πληροφοριών και επιλογών διαχείρισης.
- **Διαχείριση Χρηστών:**
  - Προβολή όλων των χρηστών (`/admin-manageUsers`).
  - Ενεργοποίηση / απενεργοποίηση λογαριασμών (`/activate_user`, `/deactivate_user`).
  - Αλλαγή ρόλου χρηστών (`/change_role`).
- **Διαχείριση Ομάδων:**
  - Δημιουργία, προβολή και διαγραφή ομάδων (`/admin-manageTeams`, `/admin-deleteTeam`).
  - Προβολή λεπτομερειών ομάδας και μελών (`/admin-viewTeam/<team_id>`).
- **Εποπτεία Εργασιών και Έργων:**
  - Συγκεντρωτική προβολή όλων των εργασιών και έργων (`/admin-show_tasks_and_projects`).

4.2 Ρόλος Επικεφαλής Ομάδας (Team Leader)

Ο **Team Leader** έχει την ευθύνη διαχείρισης των ομάδων του, των μελών τους και των εργασιών. Μπορεί να δημιουργεί, να επεξεργάζεται και να παρακολουθεί την πρόοδο των tasks.

Βασικές Λειτουργίες

- **Εγγραφή και Είσοδος:** Μέσω `/teamLeader-signup` και `/teamLeader-login`.
- **Κεντρική Σελίδα:** `/teamLeader-mainpage` με πληροφορίες του αρχηγού.
- **Διαχείριση Ομάδων:**
  - Προβολή όλων των ομάδων που διαχειρίζεται ή συμμετέχει (`/teamLeader-manageTeams`).
  - Προβολή λεπτομερειών ομάδας (`/teamLeader-manageTeam/<team_id>`).
  - Προσθήκη ή αφαίρεση μελών ομάδας (`/teamLeader-addMember`, `/teamLeader-removeMemb`
- **Διαχείριση Εργασιών:**
  - Δημιουργία νέας εργασίας (`/teamLeader-createTask/<team_id>`).
  - Επεξεργασία (`/teamLeader-editTask/<task_id>`) ή διαγραφή (`/teamLeader-deleteTask` `id>`).
  - Προβολή λεπτομερειών και σχολίων (`/teamLeader-viewTask/<task_id>`).
  - Προσθήκη σχολίων σε εργασίες (`/teamLeader-addComment/<task_id>`).
- **Παρακολούθηση Προόδου:** Μέσω της σελίδας `/teamLeader-manageTasksProjects` εμφανίζονται όλες οι εργασίες ανά έργο, με δυνατότητα εποπτείας προθεσμιών και κατάστασης.

4.3 Ρόλος Μέλους Ομάδας (Team Member)

Ο **Team Member** είναι υπεύθυνος για την εκτέλεση των εργασιών που του ανατίθενται. Μπορεί να βλέπει τις ομάδες του, να ενημερώνει την πρόοδο των εργασιών του και να επικοινωνεί μέσω σχολίων.

Βασικές Λειτουργίες

- **Εγγραφή και Είσοδος:** Μέσω `/member-signup` και `/member-login`. Ο λογαριασμός ενεργοποιείται από τον Admin.
- **Προβολή Ομάδων:** Στη σελίδα `/member-teamsIncluded` προβάλλονται οι ομάδες στις οποίες συμμετέχει.
- **Διαχείριση Εργασιών:**
  - Προβολή όλων των εργασιών του (`/member-tasks`).
  - Προβολή λεπτομερειών εργασίας (`/member-viewTask/<task_id>`).
  - Αλλαγή κατάστασης εργασίας (`/member-changeStatus/<task_id>`).
  - Προσθήκη σχολίων (`/member-addComment/<task_id>`).
- **Ειδοποιήσεις και Προθεσμίες:** Η διαδρομή `/member-notifications_and_deadlines` παρουσιάζει:
  - πρόσφατα σχόλια άλλων χρηστών σε δικές του εργασίες,
  - ενεργές προθεσμίες για εργασίες με `status` $\neq$ `DONE`.

4.4 Πίνακας Σύνοψης Δικαιωμάτων Ανά Ρόλο

Table 4.1: Σύνοψη Δικαιωμάτων και Λειτουργιών Ανά Ρόλο

Λειτουργία	**Admin**	**Team Leader**	**Member**
Εγγραφή / Είσοδος	✓	✓	✓
Δημιουργία / Διαγραφή Ομάδων	✓	–	–
Προσθήκη / Αφαίρεση Μελών Ομάδας	–	✓	–
Προβολή Όλων των Ομάδων	✓	✓	✓(μόνο τις δικές του)
Δημιουργία Εργασίας	–	✓	–
Επεξεργασία / Διαγραφή Εργασίας	–	✓	–
Προβολή Εργασιών Ομάδας	✓	✓	✓(ανατεθειμένες)
Αλλαγή Κατάστασης Εργασίας	–	✓	✓
Προσθήκη Σχολίων	–	✓	✓
Προβολή Ειδοποιήσεων / Deadlines	–	–	✓
Ενεργοποίηση / Απενεργοποίηση Χρηστών	✓	–	–
Αλλαγή Ρόλων Χρηστών	✓	–	–
Εποπτεία Όλων των Έργων	✓	–	–

Ο παραπάνω πίνακας συνοψίζει με σαφήνεια τις δυνατότητες κάθε ρόλου. Η διαχείριση του συστήματος ακολουθεί την ιεραρχία: **Admin** → **Team Leader** → **Member**, εξασφαλίζοντας ασφάλεια, σαφή κατανομή αρμοδιοτήτων και σωστή ροή εργασιών.

5. Τεχνολογίες & Εργαλεία

5.1 Επισκόπηση

Η ανάπτυξη του συστήματος πραγματοποιήθηκε με τη χρήση σύγχρονων εργαλείων λογισμικού για την ανάπτυξη, υλοποίηση και εκτέλεση web εφαρμογών σε περιβάλλον **microservices**. Η αρχιτεκτονική στηρίζεται στη λογική **Service Separation**, όπου κάθε υπηρεσία είναι αυτόνομη, ανεξάρτητη και υλοποιεί συγκεκριμένες λειτουργίες.

5.2 Backend / Microservices

Το backend αναπτύχθηκε σε **Python** με χρήση του **Flask**, ενός ελαφριού microframework για web εφαρμογές. Κάθε service (Admin, Team Leader, Member) έχει ξεχωριστό Blueprint, ανεξάρτητες διαδρομές (routes) και επικοινωνεί με τη βάση δεδομένων μέσω `psycopg2`.

- **Γλώσσα Προγραμματισμού:** Python 3.12
- **Framework:** Flask
- **Βάση Δεδομένων:** PostgreSQL
- **Driver:** psycopg2 με `RealDictCursor`
- **Δομή:** Modular Flask Blueprints (π.χ. `admin_authenticate.py`, `member_mainpage.py`)
- **Αυθεντικοποίηση:** Flask Sessions
- **API Τύπος:** RESTful endpoints σε μορφή JSON

5.3 Database Layer

Η βάση δεδομένων είναι **PostgreSQL 16**, επιλεγμένη για την αξιοπιστία και την υποστήριξη πολύπλοκων σχέσεων. Η επικοινωνία με τα services γίνεται μέσω TCP/IP, και η σύνδεση καθορίζεται στο αρχείο `config.py`.

- **Κεντρικές Οντότητες:** Users, Teams, Tasks, Comments
- **Εργαλεία:** SQL scripts για δημιουργία πινάκων (`create_tables.sql`)
- **Λειτουργίες CRUD:** Δημιουργία, Ανάγνωση, Ενημέρωση, Διαγραφή

5.4 Frontend / User Interface

Το frontend βασίστηκε σε **HTML5**, **CSS3** και **JavaScript**, χωρίς τη χρήση framework, ώστε να διατηρηθεί η απλότητα και η φορητότητα. Η επικοινωνία με το backend επιτυγχάνεται μέσω των Flask templates (Jinja2).

- **Templating Engine:** Jinja2
- **Δομή Σελίδων:**
  - `index.html` — Κεντρική σελίδα εισόδου
  - `admin_mainpage.html` — Διαχείριση χρηστών και ομάδων
  - `teamLeader_mainpage.html` — Προβολή και δημιουργία εργασιών
  - `member_viewTasks.html` — Προβολή και ενημέρωση ανατεθειμένων εργασιών
- **Static Files:** `static/css/style.css`, `static/script/script.js`

5.5 Containerization & Deployment

Η εφαρμογή είναι σχεδιασμένη ώστε να τρέχει εξολοκλήρου σε **Docker containers**. Η διαχείριση πολλαπλών υπηρεσιών γίνεται με χρήση **docker-compose** για ταυτόχρονη εκκίνηση όλων των microservices και της βάσης δεδομένων.

- **Container Tool:** Docker
- **Service Orchestration:** Docker Compose
- **Images:**

  - `python:3.12-slim` για το Flask backend
  - `postgres:latest` για τη βάση δεδομένων
- **Δίκτυο:** Κοινό Docker network για ασφαλή επικοινωνία services

5.6 Εργαλεία Ανάπτυξης

Η ανάπτυξη και η συντήρηση του συστήματος πραγματοποιήθηκε με τη χρήση των εξής εργαλείων:

- **IDE:** Visual Studio Code
- **Έλεγχος Εκδόσεων:** Git & GitHub
- **Testing:** Postman (έλεγχος API endpoints)
- **Documentation:** LaTeX
- **Διαχείριση Containers:** Docker Desktop

5.7 Συνοπτικός Πίνακας Τεχνολογιών

Table 5.1: Σύνοψη Τεχνολογιών και Εργαλείων Ανάπτυξης

Τεχνολογία / Εργαλείο	**Ρόλος στο Σύστημα**	**Περιγραφή / Σκοπός Χρήσης**
Python 3.12	Backend Development	Υλοποίηση λογικής εφαρμογής και REST endpoints
Flask	Web Framework	Δημιουργία RESTful services και routing μέσω Blueprints
PostgreSQL	Database System	Αποθήκευση χρηστών, ομάδων, εργασιών και σχολίων
psycopg2	Database Connector	Επικοινωνία Flask – PostgreSQL με ασφαλή σύνδεση
HTML, CSS, JavaScript	Frontend Layer	Διεπαφή χρήστη και απλή επικοινωνία με backend
Jinja2	Templating Engine	Δυναμική δημιουργία HTML σελίδων
Docker	Containerization Platform	Πακετάρισμα υπηρεσιών σε απομονωμένα containers
Docker Compose	Orchestration Tool	Εκκίνηση πολλαπλών containers με ένα αρχείο ρύθμισης
Git & GitHub	Version Control	Διαχείριση εκδόσεων και συνεργασία στον κώδικα
Postman	API Testing Tool	Έλεγχος λειτουργίας REST endpoints
LaTeX	Documentation Tool	Δημιουργία της τελικής αναφοράς της εργασίας

5.8 Σύνοψη

Ο συνδυασμός των παραπάνω τεχνολογιών επέτρεψε την υλοποίηση ενός επεκτάσιμου, ασφαλούς και φορητού συστήματος διαχείρισης έργων. Η χρήση Docker εξασφαλίζει την αναπαραγωγιμότητα του περιβάλλοντος, ενώ ο διαχωρισμός των υπηρεσιών σε microservices διευκολύνει τη συντήρηση και μελλοντική επέκταση.

6. Οδηγίες Εγκατάστασης & Εκτέλεσης

6.1 Απαιτήσεις Συστήματος

Για την εγκατάσταση και εκτέλεση της εφαρμογής απαιτούνται τα παρακάτω:

- **Λειτουργικό Σύστημα:** Ubuntu 22.04 LTS ή νεότερο (προτείνεται).
- **Python:** Έκδοση 3.12 ή νεότερη.
- **Docker & Docker Compose:** Για την εκτέλεση των microservices.
- **PostgreSQL:** Βάση δεδομένων για αποθήκευση των χρηστών, ομάδων, εργασιών και σχολίων.

6.2 Εγκατάσταση Τοπικά (χωρίς Docker)

1. Κλωνοποιήστε το αποθετήριο του έργου:

```
git clone https://github.com/stamatis-mavitzis/project-
    management-system.git
cd project-management-system
```

2. Δημιουργήστε και ενεργοποιήστε ένα virtual environment:

```
python3 -m venv venv
source venv/bin/activate
```

3. Εγκαταστήστε τις απαραίτητες εξαρτήσεις:

```
pip install -r requirements.txt
```

4. Δημιουργήστε τη βάση δεδομένων **PostgreSQL**:

```
CREATE DATABASE project_management;
```

5. Εκτελέστε το script δημιουργίας πινάκων:

```
psql -U postgres -d project_management -f create_tables.
   sql
```

6. Ενημερώστε το αρχείο `config.py` με τα σωστά στοιχεία σύνδεσης:

```
DB_CONFIG = {
    "host": "localhost",
    "database": "project_management",
    "user": "postgres",
    "password": "your_password"
}
```

Listing 6.1: Παράδειγμα αρχείου config.py

7. Εκτελέστε την εφαρμογή Flask:

```
source source_run_flask.sh
```

8. Ανοίξτε τον browser και επισκεφθείτε τη διεύθυνση:

`http://127.0.0.1:5000`

6.3 Εκτέλεση μέσω Docker

Η εφαρμογή υποστηρίζει πλήρη ανάπτυξη σε περιβάλλον Docker για εύκολη αναπαραγωγή.

6.3.1 Βήματα Εγκατάστασης

1. Βεβαιωθείτε ότι το `docker` και το `docker-compose` είναι εγκατεστημένα:

```
docker --version
docker compose version
```

2. Δημιουργήστε το αρχείο `docker-compose.yml` με τα services της εφαρμογής (Admin, Team Leader, Member, Database).

3. Εκκινήστε όλα τα containers:

```
docker compose up --build
```

4. Ελέγξτε ότι τα services τρέχουν:

```
docker ps
```

5. Ανοίξτε το web interface στη διεύθυνση:

   `http://localhost:5000`

6.4 Εκτέλεση στο Google Cloud Platform (GCP)

Για την ανάπτυξη στο GCP:

1. Δημιουργήστε ένα **Compute Engine VM** με λειτουργικό **Ubuntu**.
2. Εγκαταστήστε Docker και Docker Compose στο VM:

```
sudo apt update
sudo apt install docker.io docker-compose -y
```

3. Μεταφέρετε (π.χ. μέσω `scp`) τα αρχεία του project στο VM.
4. Εκτελέστε:

```
docker compose up --build -d
```

5. Ανοίξτε την αντίστοιχη πόρτα (π.χ. 5000) από τα firewall rules του GCP.
6. Επισκεφθείτε το URL:

   `http://<your-external-ip>:5000`

6.5 Πιστοποίηση Εγκατάστασης

Για να επιβεβαιωθεί η επιτυχής εκτέλεση:

- Εμφανίζεται η αρχική σελίδα (homepage) με επιλογή ρόλου χρήστη.
- Ο Admin μπορεί να δημιουργήσει/ενεργοποιήσει χρήστες και ομάδες.
- Ο Team Leader μπορεί να διαχειριστεί εργασίες και μέλη.
- Ο Member μπορεί να δει και να ενημερώσει τις εργασίες του.

6.6 Συνοπτικές Εντολές Εκτέλεσης

```
# Local run
source source_run_flask.sh

# Docker run
docker compose up --build

# Stop containers
docker compose down
```

7. Συμπεράσματα και Μελλοντικές Επεκτάσεις

7.1 Συμπεράσματα – Εμπειρία Ανάπτυξης

Η ανάπτυξη του συστήματος **Project Management System** αποτέλεσε μια ολοκληρωμένη εμπειρία εφαρμογής των εννοιών των **microservices**, της **containerization** μέσω Docker και της **διαχείρισης ρόλων χρηστών** σε ένα πραγματικό web περιβάλλον.

Κατά την υλοποίηση, εμβάθυνα στην αρχιτεκτονική του Flask, τη δομή των Blueprints και τη σύνδεση με τη βάση δεδομένων PostgreSQL. Η δημιουργία διαφορετικών microservices για κάθε ρόλο (Admin, Team Leader, Member) βοήθησε στην κατανόηση του πώς μια εφαρμογή μπορεί να διαχωρίζεται σε ανεξάρτητα μέρη με σαφή ευθύνη και κοινή επικοινωνία μέσω REST API endpoints.

Μέσα από τη χρήση του Docker, κατανόησα τη σημασία της **αναπαραγωγιμότητας** και της **απομόνωσης περιβάλλοντος ανάπτυξης**. Η δυνατότητα εκτέλεσης της εφαρμογής μέσω containers απλοποίησε σημαντικά τη διαχείριση εξαρτήσεων και την εγκατάσταση των υπηρεσιών.

7.1.1 Δυσκολίες που Αντιμετωπίστηκαν

Κατά την ανάπτυξη προέκυψαν ορισμένες προκλήσεις:

- Η διαχείριση των session variables ανά ρόλο (Admin, Team Leader, Member) ώστε να αποφεύγεται η σύγχυση μεταξύ διαφορετικών χρηστών.
- Η διαχείριση σύνθετων SQL ερωτημάτων για προβολή και φιλτράρισμα δεδομένων (π.χ. tasks ανά ομάδα ή ανά χρήστη).
- Η σωστή ροή δεδομένων μεταξύ Flask templates και Python routes.
- Η επικοινωνία μεταξύ microservices και η ανάγκη για σαφές API σχεδιασμό.

Ωστόσο, μέσα από αυτές τις δυσκολίες ενισχύθηκε η κατανόηση των αρχών του **modular design** και της **δομημένης επικοινωνίας μέσω API**.

7.1.2 Μελλοντικές Επεκτάσεις – Βελτιώσεις

Στο μέλλον, η εφαρμογή μπορεί να εμπλουτιστεί με τις ακόλουθες βελτιώσεις:

- **WebSockets για live notifications:** Εμφάνιση σχολίων, αλλαγών κατάστασης ή προθεσμιών σε πραγματικό χρόνο χωρίς ανανέωση της σελίδας.
- **File attachments:** Δυνατότητα προσθήκης αρχείων (π.χ. screenshots, αναφορές) στις εργασίες και στα σχόλια, αποθηκευμένων στο server ή σε cloud storage.
- **Dashboard με γραφήματα:** Χρήση βιβλιοθηκών όπως Chart.js για απεικόνιση της προόδου ομάδων και των εργασιών ανά κατάσταση (TODO / IN PROGRESS / DONE).
- **Email notifications:** Αυτόματη αποστολή ενημερώσεων στα μέλη για νέες αναθέσεις ή αλλαγές προθεσμιών.
- **Role-based API Gateway:** Δημιουργία ενός API gateway (π.χ. με Nginx ή FastAPI) που θα ενοποιεί την επικοινωνία όλων των microservices.
- **Migration στο Cloud (GCP):** Ανάπτυξη της εφαρμογής στο Google Cloud Platform μέσω Docker Compose, για πλήρη παραγωγική λειτουργία.

Η δομή της εφαρμογής επιτρέπει τέτοιες επεκτάσεις χωρίς να απαιτείται ανασχεδίαση της αρχιτεκτονικής, αποδεικνύοντας την ευελιξία της microservices προσέγγισης.

7.2 Παραρτήματα

7.2.1 Απόσπασμα SQL (create_tables.sql)

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    name VARCHAR(50),
    surname VARCHAR(50),
    role VARCHAR(20),
```

```
    status VARCHAR(20)
);

CREATE TABLE teams (
    team_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    leader_id INTEGER REFERENCES users(user_id),
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE tasks (
    task_id SERIAL PRIMARY KEY,
    title VARCHAR(100),
    description TEXT,
    created_by INTEGER REFERENCES users(user_id),
    assigned_to INTEGER REFERENCES users(user_id),
    team_id INTEGER REFERENCES teams(team_id),
    status VARCHAR(20),
    priority VARCHAR(20),
    due_date DATE,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE comments (
    comment_id SERIAL PRIMARY KEY,
    task_id INTEGER REFERENCES tasks(task_id),
    author_id INTEGER REFERENCES users(user_id),
    content TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Listing 7.1: Δημιουργία βασικών πινάκων στη βάση δεδομένων

7.2.2 Παραπομπές – Πηγές

- Flask Documentation: `https://flask.palletsprojects.com/`
- PostgreSQL Docs: `https://www.postgresql.org/docs/`
- Docker Docs: `https://docs.docker.com/`

- Tutorial: Flask + PostgreSQL CRUD Application
- Chart.js Official Documentation: `https://www.chartjs.org/docs/latest/`