Universitatea
Transilvania
din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ
ȘI ȘTIINȚA CALCULATOARELOR

Documentație Instrumente pentru dezvoltarea programelor

Student: Bertescu Andrei

Coordonatori: Iulian-Gabriel Gavrilă, Alexandru Pușcașu

Brașov, 2025

Cuprins

Introducere 3
Obiectivele proiectului 4
Tehnologii folosite 5
Arhitectura aplicatiei 6
Implementarea funcționalităților 7
1. Autentificare și autorizare 7
2. Gestionarea utilizatorilor 8
3. Gestionarea notițelor si evenimentelor 10
4. Interfața utilizator (Frontend) 10
5. Controlere și servicii 11
Testare și profilare 13
6. Testarea performanței - JMeter 13
7. Profiling și monitorizare – VisualVM 14
8. Testare manuala 15
CI-CD și containerizare 16
9. Docker și Docker Compose 16
10. Integrare continua cu GitHub Actions 17
Concluzii 19
Bibliografie 20

Introducere

În era digitalizării accelerate, gestionarea eficientă a informațiilor personale devine o necesitate. Aplicațiile pentru organizarea notițelor, a evenimentelor și a sarcinilor sunt tot mai căutate atât în mediul profesional, cât și în viața personală. Proiectul NotePlan se înscrie în această tendință, propunând o soluție software completă, modernă și sigură pentru centralizarea și gestionarea evenimentelor și a notițelor.

NotePlan este o aplicație web dezvoltată cu ajutorul unor tehnologii mature și populare în ecosistemul Java, cum ar fi Spring Boot, Spring Security, Hibernate și MySQL, dar și cu suport modern pentru testare, containerizare și Continuous Integration/Continuous Delivery. Aceasta oferă utilizatorilor o interfață intuitivă prin care pot crea, edita și șterge evenimente și notițe.

În contextul utilizării tot mai largi a aplicațiilor de tip to-do list și planner digital (ex. Google Keep, Notion, Evernote), mulți utilizatori simt nevoia unei aplicații mai personalizate, care să le permită o autonomie mai mare asupra datelor și să le ofere posibilitatea de a rula aplicația local sau pe un server propriu. Motivația din spatele proiectului este de a oferi o alternativă open-source și extensibilă la aplicațiile existente, punând accent pe autonomie, securitate și control.

Obiectivele proiectului

Obiectivele majore ale aplicației NotePlan sunt:

- Dezvoltarea unei aplicații web full-stack pentru managementul notițelor și evenimentelor
- Implementarea unei autentificări sigure, cu sistem de autorizare pe roluri
- Persistența datelor într-o bază de date relațională
- Testarea automată a funcționalităților cheie
- Integrarea unui proces DevOps complet (CI/CD, containerizare)
- Oferirea unei interfețe prietenoase și moderne pentru utilizatori
- Asigurarea portabilității aplicației prin intermediul Docker

Obiectivele secundare sunt:

- Învățarea unui nou Framework (Spring Boot)
- Deprinderea conceptelor fundamentale de DevOps si full-stack development

Tehnologii folosite

1. **Backend**
   - **Java** – limbajul de programare principal, ales pentru robustețea și ecosistemul vast.
   - **Spring Boot** – framework-ul folosit pentru dezvoltarea aplicației backend. Permite o configurare rapidă, oferind o bază solidă pentru aplicații REST.
   - **Spring Security** – framework-ul pentru gestionarea autentificării și autorizării utilizatorilor.
   - **Hibernate** – ORM folosit pentru a facilita interacțiunea cu baza de date.
2. **Frontend**
   - **HTML 5.0** și **CSS**
3. **Bază de date**
   - **MySQL** – sistemul de gestionare a bazei de date relațională;
   - **MySQL Workbench** – utilizat pentru modelarea și monitorizarea bazei de date în timpul dezvoltării.
4. **Testare și monitorizare**
   - **JMeter** – instrument de testare a performanței aplicației;
   - **VisualVM** – utilizat pentru profiling și identificarea bottleneck-urilor aplicației.
5. **DevOps**
   - **GitHub** – sistem de versionare a codului sursă;
   - **GitHub Actions** – pipeline CI/CD pentru rularea automată a testelor și build-ului;
   - **Docker** – containerizare completă a aplicației;
   - **Docker Compose** – orchestrare simplificată a tuturor serviciilor aplicației.

Arhitectura aplicatiei

Aplicația **NotePlan** respectă modelul de design arhitectural **Model-View-Controller (MVC)**, oferind o separare clară între:

- **Model** – componentele care gestionează datele (entitățile User, Note, Event, etc.);
- **View** – interfața utilizatorului, adică paginile frontend prin care utilizatorul interacționează cu aplicația;
- **Controller** – clasele din backend care primesc cererile HTTP, interacționează cu serviciile și returnează datele înapoi către View.

Frontend (Client) conține:

- Login/Register
- Dashboard personal
- Pagina de profil
- Formulare pentru notițe/evenimente

Backend (Server)

- Realizat în **Spring Boot**.
- Integrează:
    - Servicii (Business Logic)
    - Repozitorii (Acces la date)
    - Controlere (Conexiune cu frontend)
    - Configurații de securitate (Spring Security)

Bază de date

- MySQL, accesată prin Hibernate (JPA).
- Entități modelate: User, Note, Event, Authority.

Autentificare și autorizare

- Login + confirmare email.
- Gestionarea rolurilor (ex. USER, GUEST).

Containerizare și orchestrare

- Aplicația este containerizată cu Docker.
- Serviciile sunt orchestrate cu **Docker Compose** pentru a rula simultan:
    - Aplicația Spring Boot
    - Baza de date

CI/CD

- GitHub Actions folosit pentru:
    - Build automat la push pe ramura principală.
    - Rulare teste.
    - Deployment pe container in GitHub Registry.

Implementarea funcționalităților

Arhitectura modulară a aplicației NotePlan permite o organizare clară a codului în componente: **Controllers**, **Services**, **Repositories**, **Entities**, și **Security**. Această organizare contribuie la o dezvoltare și întreținere eficientă a aplicației.

1. Autentificare și autorizare

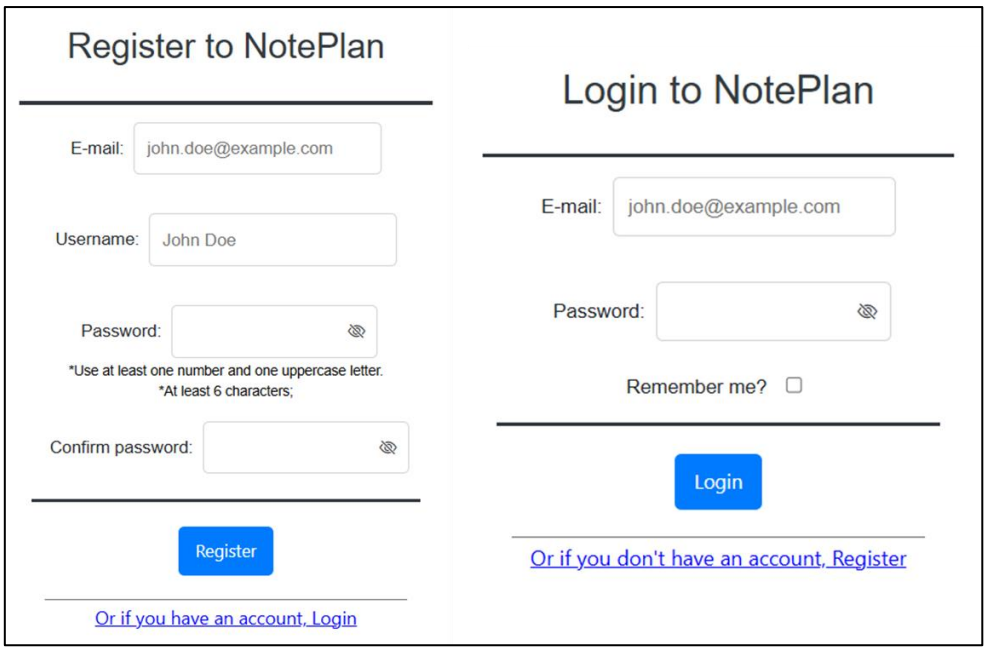
Pentru autentificare și autorizare s-a folosit **Spring Security**, care oferă un mecanism robust și extensibil pentru protejarea resurselor.

Funcționalități implementate:

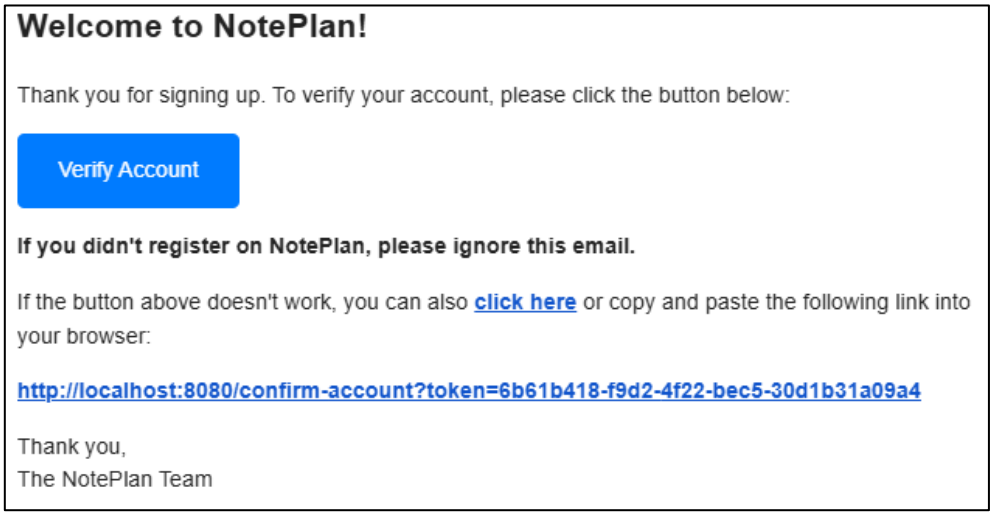
- Înregistrare utilizator (cu rol implicit de USER);
- Autentificare cu nume de utilizator și parolă;
- Confirmare cont prin e-mail.
- Protejarea endpoint-urilor pe baza rolurilor (USER, ADMIN).

Puncte cheie din implementare:

- Clasa **SecurityConfig** definește rutele publice (/login, /register) și cele protejate (/dashboard, /profile, etc.).
- **UserDetailsService** personalizat pentru încărcarea utilizatorilor din baza de date.
- **BCrypt** pentru criptarea parolelor.



Figură 1 Paginile de register si login



Figură 2 Email de confirmare

2. Gestionarea utilizatorilor

Entitatea User este elementul central, reprezentând fiecare cont individual. Se asociază cu tabelele Note, Event, și Authority.

Funcționalități:

- Afișare și editare profil;
- Ștergere cont;
- Listare notițe/evenimente personale;
- Asociere automată la înregistrare cu rol USER.

Entitatea User are următoarele componente:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(length = STRING_LENGTH)
private String password;

@Column(length = STRING_LENGTH)
private String username;

@Column(length = STRING_LENGTH)
private String name;

private boolean timeFormat = false;
private boolean theme = false;

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "user")
private Set<Authority> authorities = new HashSet<>();

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy = "user")
private Set<Event> events = new HashSet<>();

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
mappedBy "user")
 private Set<Note> notes = new HashSet<>();
```

3. Gestionarea notițelor si evenimentelor

Aplicația oferă funcționalități CRUD complete pentru aceste doua elemente: creare, editare, ștergere si vizualizare

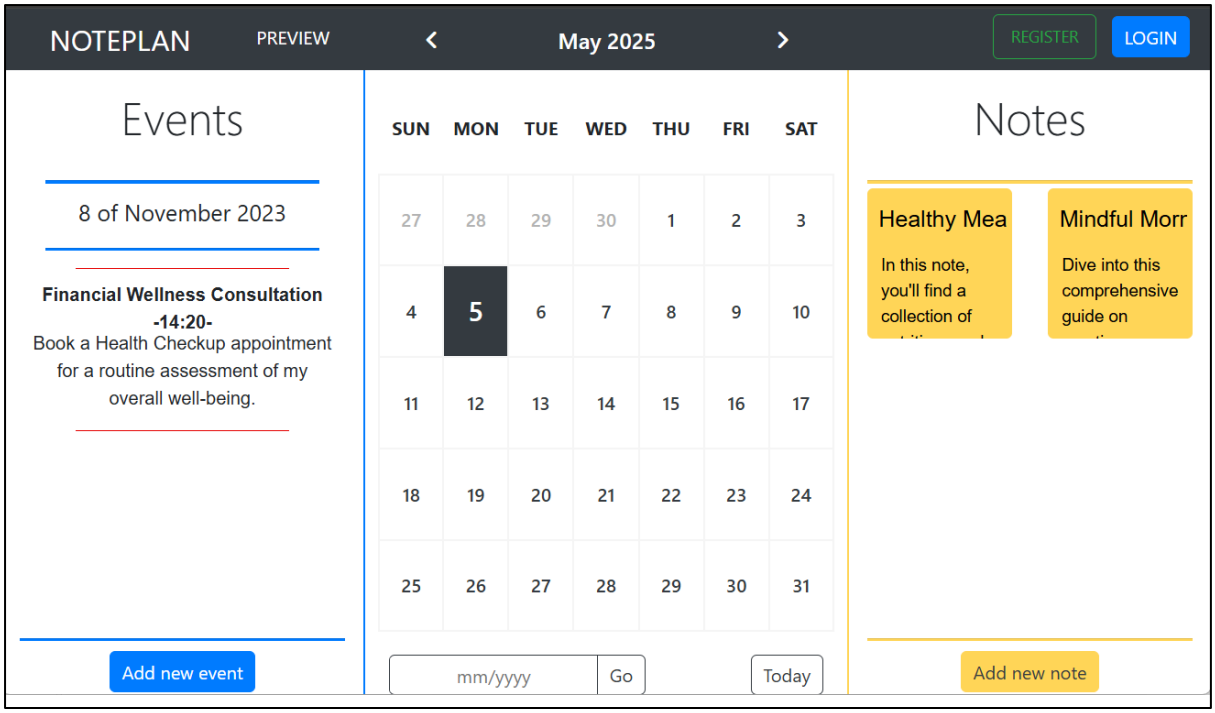
Ca exemplu, iată componentele entității Event:

```
private static final int COLOR_LENGTH = 7;
private static final int STRING_LENGTH = 100;
private static final int NOTE_LENGTH = 1000;

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@ManyToOne
private User user;

@Column(length = STRING_LENGTH)
private String date;

@Column(length = STRING_LENGTH)
private String title;

@Column(length = NOTE_LENGTH)
private String description;

@Column(length = COLOR_LENGTH)
private String color;
```

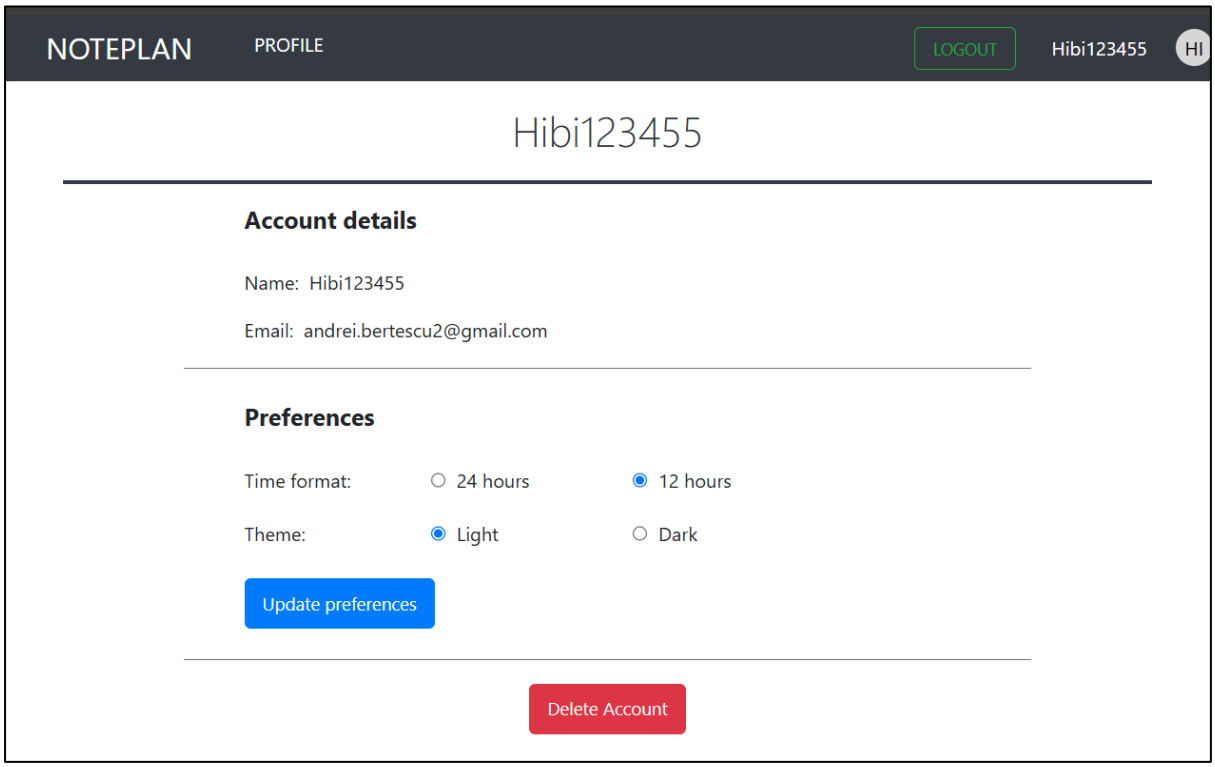
4. Interfața utilizator (Frontend)

Interfața este simplă, intuitivă, și orientată pe eficiență:

- **Login/Register:** formulare de autentificare, validare minimă pe client.
- **Dashboard:** afișează toate evenimentele și notițele.
- **Landing page:** un exemplu de dashboard pentru noii utilizatori
- **Profile Page:** permite modificarea datelor utilizatorului.
- **Formulare:** pentru adăugare/editare notițe și evenimente, integrate în dashboard.



Figură 3 Pagina dashboard



Figură 4 Pagina de profil

5. Controlere și servicii

Aplicația este organizată în mod tipic Spring:

- **Controllers:** rutele REST, expun logica aplicației către frontend.
- **Services:** conțin logica aplicativă – validări, conversii, etc.
- **Repositories:** interfață cu baza de date prin JPA/Hibernate.

Controlerele sunt organizate în trei fișiere, fiecare având o funcționalitate specifică:

- **DashboardController** se ocupă de pagina principală și de request-urile privind operațiile CRUD ale notițelor și evenimentelor.
- **LoginController** coordonează request-urile de autentificare, login si trimiterea de email-uri de confirmare.
- **ProfileController** menține pagina de profil, unde se pot modifica preferințele sau se poate șterge contul.

Mai jos este structura clasei **DashboardController:**

```
// Funcții publice
Event initializeSelectedEvent(void);
Note initializeSelectedNote(void);

const char* rootView(Authentication auth);
const char* rootView2(Authentication auth);
const char* health(void);
const char* dashBoardView(User user, ModelMap model);

const char* newEventForm(User user, Event event);
const char* getEventDetails(User user, long eventId, Model smodel);
const char* updateEventForm(User user, Event event, Model smodel);
const char* deleteEventForm(User user, Model smodel);

const char* newNoteForm(User user, Map<String, String> params, const
char* isChecklist, Note note);
const char* getNoteDetails(User user, long noteId, Model smodel);
const char* updateNoteForm(User user, Note note, Map<String, String>
params, Model smodel);
const char* deleteNoteForm(Model smodel);

void deleteSmodel(Model smodel, HttpServletResponse response);

// Funcții private
const char* formatTime(const char* militaryTime);
const char* getInitials(const char* name);
```

Testare și profilare

În cadrul procesului de dezvoltare software, testarea și profilarea reprezintă etape fundamentale pentru validarea funcționalității, stabilității și performanței aplicației. Aceste activități contribuie esențial la asigurarea calității produsului final, la prevenirea erorilor în mediul de producție și la menținerea unui cod robust și scalabil.

Pentru proiectul **NotePlan**, testarea a fost abordată prin mai multe niveluri: testare unitară pentru verificarea componentelor individuale, testare de performanță pentru evaluarea comportamentului sub sarcină și profilare pentru analiza eficienței aplicației în execuție. Acestea au permis o abordare completă și robustă a procesului de asigurare a calității, ceea ce a contribuit semnificativ la stabilitatea aplicației și a redus timpul necesar depanării.

6. Testarea performanței - JMeter

Pentru a simula trafic intens și a evalua timpii de răspuns ai aplicației, s-a folosit **Apache JMeter**.

Scopuri:

- Evaluarea scalabilității aplicației în medii de producție
- Identificarea bottleneck-urilor de performanță
- Testarea endpoint-urilor REST în condiții de trafic intens

Exemple de teste:

- 50 de utilizatori concurenți care se autentifica și accesează dashboard-ul
- 50 de cereri POST simultane pentru creare de notițe

Rezultate urmărite:

- Timp mediu de răspuns
- Rata de eșec

Mai jos se pot vedea statisticile generate automat după testarea performanței.

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	50	3	6.00%	69.00	2	404	33.50	242.60	360.85	404.00	47.53	439.16	19.50
Create notes	5	0	0.00%	60.60	54	71	56.00	71.00	71.00	71.00	9.06	171.63	7.31
Create notes-0	5	0	0.00%	17.40	12	19	19.00	19.00	19.00	19.00	10.02	2.92	5.79
Create notes-1	5	0	0.00%	43.00	35	53	37.00	53.00	53.00	53.00	9.38	175.02	2.14
Get login page	5	0	0.00%	21.00	6	77	8.00	77.00	77.00	77.00	6.94	18.47	0.87
Submit login	5	3	60.00%	265.40	141	404	247.00	404.00	404.00	404.00	5.65	106.09	3.40
Submit login-0	5	0	0.00%	220.80	109	355	203.00	355.00	355.00	355.00	5.95	3.02	2.50
Submit login-1	5	0	0.00%	44.00	32	49	48.00	49.00	49.00	49.00	9.43	172.35	1.70
Submit logout	5	0	0.00%	9.00	7	12	9.00	12.00	12.00	12.00	9.84	70.31	5.72
Submit logout-0	5	0	0.00%	2.60	2	3	3.00	3.00	3.00	3.00	9.96	5.94	4.08
Submit logout-1	5	0	0.00%	6.20	5	8	6.00	8.00	8.00	8.00	9.90	64.82	1.70

Figură 5 Statisticile testelor de performanță

7. Profiling și monitorizare – VisualVM

Pentru analizarea comportamentului aplicației în timpul rulării s-a folosit **VisualVM** – un instrument de profiling vizual pentru aplicații.

Scopuri:

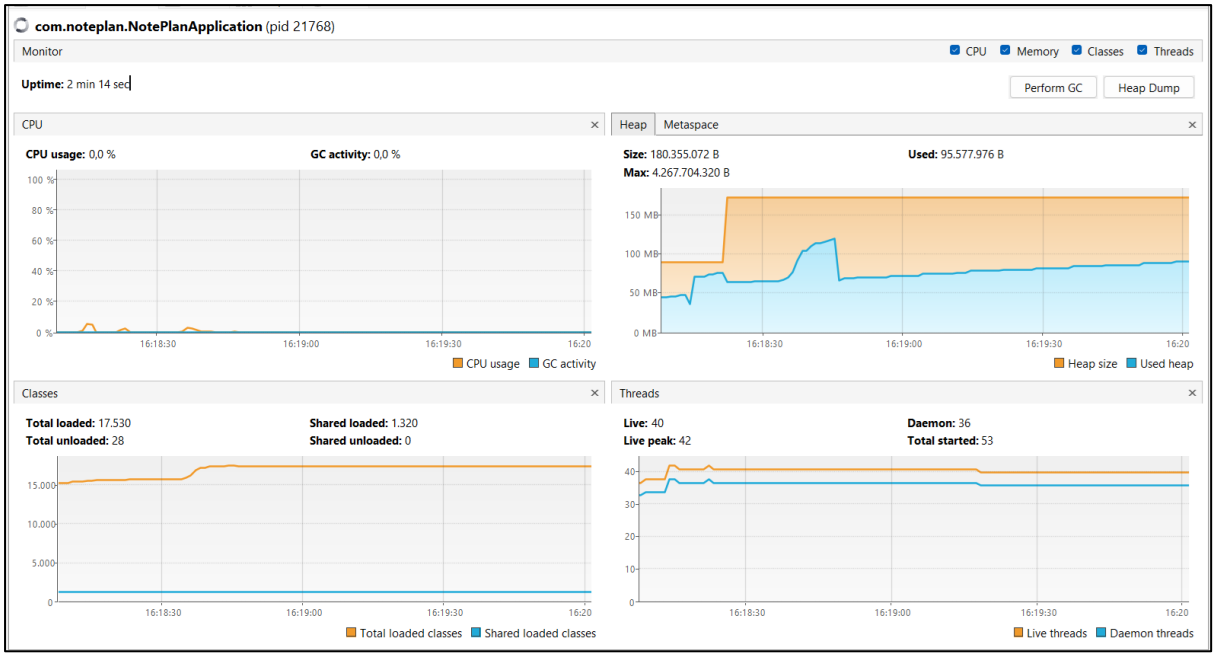
- Monitorizarea consumului de memorie
- Detectarea scurgerilor de memorie (memory leaks)
- Identificarea metodelor lente și a zonelor de cod intens procesate (hotspots)

Funcționalități utilizate:

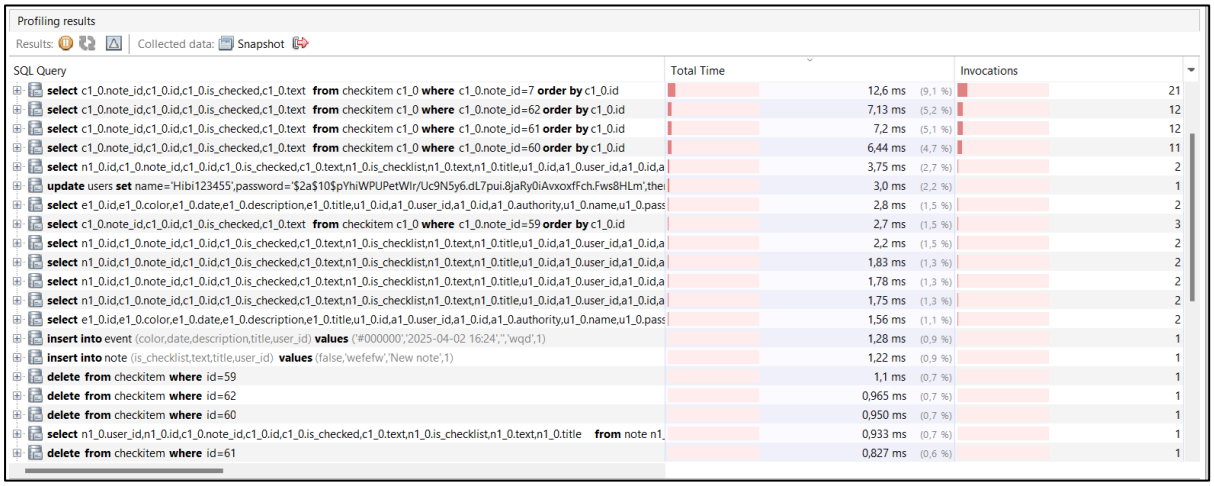
- Heap dump analysis
- Thread monitoring – pentru detectarea blocajelor

Beneficii:

- Optimizare proactivă a codului
- Evitarea degradării performanței în producție



Figură 6 Resource monitor



Figură 7 Profiling pentru cererile SQL

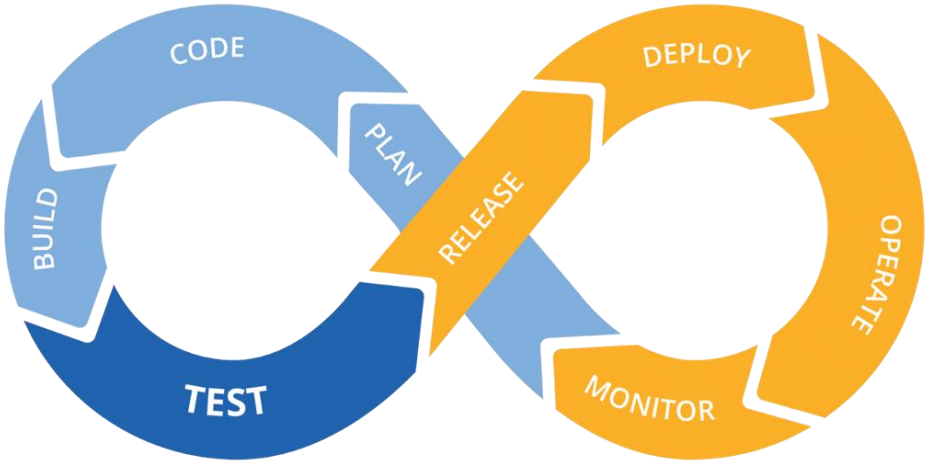
8. Testare manuala

Pe lângă testele automate de profiling si performanță, s-au efectuat și testări manuale pentru a verifica:

- Comportamentul interfeței grafice
- Fluxurile de utilizare complete (login −> dashboard −> editare −> logout)
- Validarea mesajelor de eroare și a redirectărilor

CI-CD și containerizare

Într-un ciclu modern de dezvoltare software, livrarea continuă (CI/CD) și containerizarea sunt fundamentale. Aceste practici asigură nu doar livrarea rapidă a codului în producție, ci și consistența, scalabilitatea și portabilitatea aplicației.



Figură 8 Procesul CI-CD

9. Docker și Docker Compose

Pentru a rula aplicația în medii izolate și portabile, s-a folosit **Docker**, un standard în industrie pentru livrarea aplicațiilor în containere. Acesta se folosește de un fișier de configurație numit Dockerfile. Procesul construirii imaginii se face in 2 pași, unul pentru construirea propriu-zisa a aplicației, iar unul pentru a păstra doar fișierele esențiale rulării aplicației, astfel făcându-se o economie de spațiu.

Fisierul de configuratie Dockerfile este urmatorul:

```
# Stage 1: Build the application
FROM openjdk:17-jdk-slim AS build

WORKDIR /app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src ./src

RUN chmod +x mvnw
RUN ./mvnw clean package -DskipTests

# Stage 2: Run the application
```

```
FROM openjdk:17-jdk-slim

WORKDIR /app

# Copy the built .jar from the build stage
COPY --from=build /app/target/*.jar app.jar

EXPOSE 8080
CMD ["java", "-Xmx256m", "-Xms128m", "-jar", "app.jar"]
```

Pentru a rula întreaga aplicație însă (backend + baza de date), s-a utilizat **Docker Compose**. Acesta definește serviciile în fișierul docker-compose.yml. În acest fișier se creează două servicii, aplicația Spring propriu-zisa si baza de date MySQL, dar și un volum persistent care stochează toate datele strânse, astfel încât să nu se piardă la închiderea aplicației. Fișierul de configurație poate fi găsit mai jos:

```yaml
services:
  # Spring Boot app service
  springboot-app:
    restart: unless-stopped
    build:
      context: .
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATASOURCE_URL=${SPRING_DATASOURCE_URL}
      - SPRING_DATASOURCE_USERNAME=${SPRING_DATASOURCE_USERNAME}
      - SPRING_DATASOURCE_PASSWORD=${SPRING_DATASOURCE_PASSWORD}
      - SPRING_MAIL_USERNAME=${SPRING_MAIL_USERNAME}
      - SPRING_MAIL_PASSWORD=${SPRING_MAIL_PASSWORD}
    depends_on:
      - mysql-db

  # MySQL service
  mysql-db:
    restart: unless-stopped
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: ${SPRING_DATASOURCE_PASSWORD}
      MYSQL_DATABASE: ${SPRING_DATASOURCE_DB_NAME}
    ports:
      - "3307:3306"
    volumes:
      - mysql-data:/var/lib/mysql

volumes:
  mysql-data:  # Define a volume to persist MySQL data
```

10. Integrare continua cu GitHub Actions

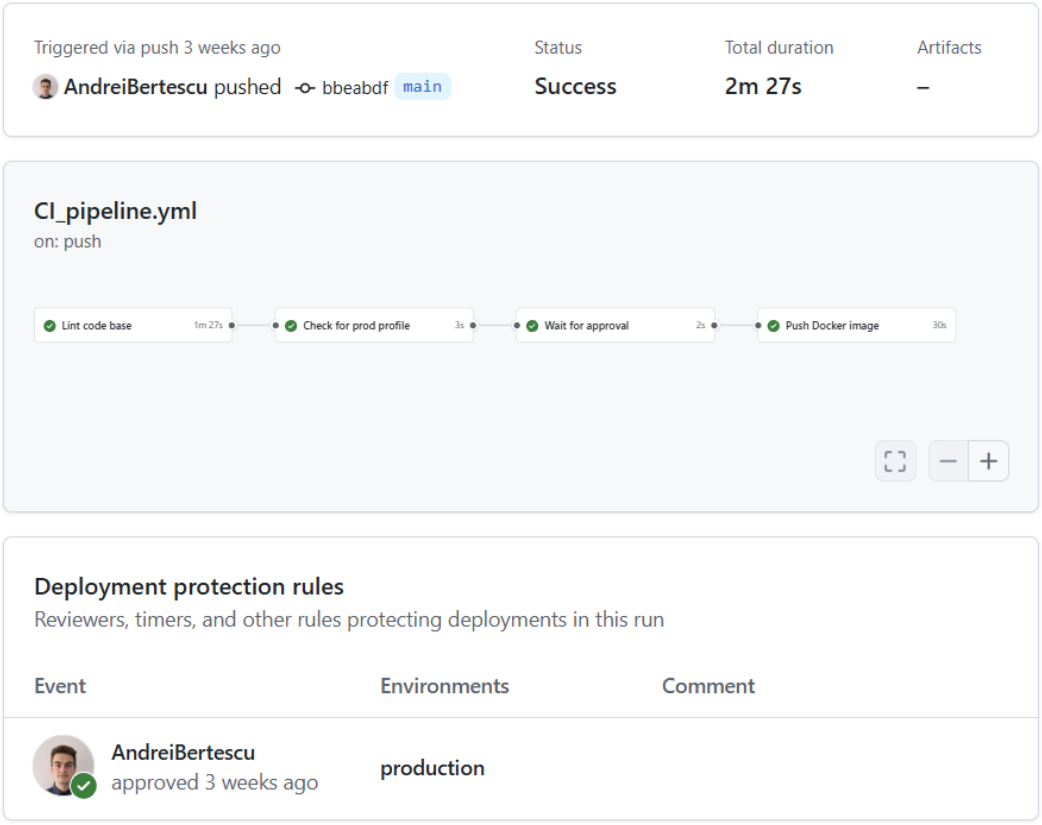
GitHub Actions este resursa folosită pentru a implementa pipeline-ul de integrare continuă. Workflow-ul CI rulează la fiecare push în ramura principală (main), asigurând lintarea și build-ul aplicației.

Etapele pipeline-ului:

1. Lintarea codului sursă folosind Super-Linter
2. Verifica daca proiectul este menit pentru enviroment-ul de producție
3. Așteaptă confirmarea fizica a unui developer
4. Construiește imagine Docker
5. Încarcă imaginea Docker în GitHub Registry

Beneficii:

- Feedback rapid după fiecare commit
- Prevenirea introducerii codului cu erori
- Automatizare completă a procesului de verificare a aplicației



Figură 9 Workflow ce rulează la fiecare push

Pașii 4 si 5 automatizează procesul de deployment. Imaginea construita este trimisa în GitHub Container Registry și apoi rulate automat pe un server. Codul responsabil pentru construirea automată a imaginii și încărcarea acesteia în Registry este dat mai jos.

```
push-image:
  name: Push Docker image
  needs: wait-for-approval
  runs-on: ubuntu-latest

  permissions:
    contents: read
    packages: write
```

```
  steps:
    - name: Checkout source code
      uses: actions/checkout@v3

    - name: Log in to GitHub Container Registry
      uses: docker/login-action@v2
      with:
        registry: ghcr.io
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Build and tag Docker image
      run: |IMAGE_NAME=ghcr.io/andreibertescu/noteplan:latest
        docker build -t $IMAGE_NAME .
        echo "Built $IMAGE_NAME"

    - name: Push Docker image to GHCR
      run: |IMAGE_NAME=ghcr.io/andreibertescu/noteplan:latest
        docker push $IMAGE_NAME
```

Concluzii

Realizarea proiectului NotePlan a reprezentat un demers complex, dar foarte valoros din punct de vedere educațional și tehnic. Aplicația a fost concepută și implementată ca o soluție modernă pentru gestionarea eficientă a notițelor și evenimentelor, oferind o interfață intuitivă, funcționalități complete și o infrastructură scalabilă, potrivită atât pentru utilizare individuală, cât și pentru viitoare extinderi.

Pe parcursul dezvoltării, proiectul a integrat concepte din domeniul ingineriei software, precum arhitectura MVC, interacțiunea cu baze de date relaționale și automatizarea procesului de livrare. Din punct de vedere al securității, aplicația oferă o protecție solidă a datelor utilizatorilor prin autentificare și autorizare bine definite, iar persistarea informațiilor este realizată cu ajutorul unei baze de date relaționale eficiente.

Bibliografie

1. Docker, Inc., "Docker CLI reference," Docker Documentation. [Online]. Available: https://docs.docker.com/reference/cli/docker/. [Accessed: May 22, 2025].
2. Docker, Inc., "Getting started with Docker Compose," *Docker Documentation*. [Online]. Available: https://docs.docker.com/compose/gettingstarted/.[Accessed: May 22, 2025].
3. GitHub, Inc., "About GitHub Actions," *GitHub Docs*. [Online]. Available: https://docs.github.com/en/actions. [Accessed: May 22, 2025].
4. The Apache Software Foundation, "Spring Boot," *Spring.io*. [Online]. Available: https://spring.io/projects/spring-boot. [Accessed: May 22, 2025].
5. The Apache Software Foundation, "Spring Security," *Spring.io*. [Online]. Available: https://spring.io/projects/spring-security. [Accessed: May 22, 2025].
6. Hibernate.org, "Hibernate ORM," *Hibernate.org*. [Online]. Available: https://hibernate.org/orm/. [Accessed: May 22, 2025].
7. Oracle Corporation, "MySQL Documentation," *MySQL.com*. [Online]. Available: https://dev.mysql.com/doc/. [Accessed: May 22, 2025].
8. GitHub, Inc., "Understanding the GitHub flow," *GitHub Docs*. [Online]. Available: https://docs.github.com/en/get-started/quickstart/github-flow. [Accessed: May 22, 2025].
9. Oracle Corporation, "JUnit 5 User Guide," JUnit.org. [Online]. Available: https://junit.org/junit5/docs/current/user-guide/. [Accessed: May 22, 2025].
10. The Apache Software Foundation, "Apache JMeter User Manual," Apache JMeter. [Online]. Available: https://jmeter.apache.org/usermanual/index.html. [Accessed: May 22, 2025].
11. Oracle Corporation, "VisualVM Documentation," VisualVM. [Online]. Available: https://visualvm.github.io/. [Accessed: May 22, 2025].
12. Oracle Corporation, "Java Platform, Standard Edition Documentation," Oracle. [Online]. Available: https://docs.oracle.com/javase/. [Accessed: May 22, 2025].
13. Baeldung, "Introduction to Spring Data JPA," Baeldung.com. [Online]. Available: https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa. [Accessed: May 22, 2025].