Facultatea de Matematică și Informatică

Universitatea București

**Document de Analiză și Proiectare pentru Aplicația Software**

**Sistem de Asistență pentru Decizii Medicale: Un AI care sugerează diagnostice bazate pe simptome și dosarele medicale ale pacienților**

**Cojoacă Alexandru**

**alexandru.cojoaca@s.unibuc.ro**

**Data**

**04.02.2025**

Contents

[**1. Introducere** 3](#_Toc192852575)

[**1.1 Scopul documentului** 3](#_Toc192852576)

[**1.2 Domeniul de aplicare** 4](#_Toc192852577)

[**1.3 Definiții, acronime și abrevieri** 4](#_Toc192852578)

[**1.4 Referințe** 5](#_Toc192852579)

[**1.5 Prezentare generală** 5](#_Toc192852580)

[**2. Analiza cerințelor** 7](#_Toc192852581)

[**2.1 Cerințe funcționale** 7](#_Toc192852582)

[**2.2 Cerințe non-funcționale** 8](#_Toc192852583)

[**2.3 Cazuri de utilizare (Use Cases)** 9](#_Toc192852584)

[**2.4 Constrângeri** 10](#_Toc192852585)

[**2.5 Riscuri** 11](#_Toc192852586)

[**3. Arhitectura și proiectarea sistemului** 13](#_Toc192852587)

[**3.1 Arhitectura generală** 13](#_Toc192852588)

[**3.2 Modelul de date** 14](#_Toc192852589)

[**3.3 Diagrame UML (Unified Modelling Language)** 14](#_Toc192852590)

[**3.4 Tehnologii și stack de dezvoltare** 16](#_Toc192852591)

[**3.5 Interfața utilizatorului (UI/UX)** 16](#_Toc192852592)

[**4. Implementare** 18](#_Toc192852593)

[**4.1 Structura codului** 18](#_Toc192852594)

[**4.2 Module și componente** 19](#_Toc192852595)

[**4.3 Fluxul de date** 20](#_Toc192852596)

[**5. Testare** 22](#_Toc192852597)

[**5.1 Strategia de testare** 22](#_Toc192852598)

[**5.2 Tipuri de teste** 23](#_Toc192852599)

[**5.3 Plan de testare** 24](#_Toc192852600)

[**6. Deployment și mentenanță** 26](#_Toc192852601)

[**6.1 Strategie de deployment** 26](#_Toc192852602)

[**6.2 Administrarea și monitorizarea** 27](#_Toc192852603)

[**6.3 Plan de mentenanță** 28](#_Toc192852604)

[**7. Concluzii** 30](#_Toc192852605)

# **1. Introducere**

## **1.1 Scopul documentului**

*Explicarea obiectivelor documentului de analiză și proiectare.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Acest document are scopul de a defini cerințele, arhitectura și proiectarea unei aplicații software, oferind o bază solidă pentru implementare și testare. El servește drept ghid pentru echipa de dezvoltare și părțile interesate, asigurând o înțelegere clară a funcționalităților, cerințelor non-funcționale și constrângerilor tehnice. De asemenea, documentul detaliază procesul de testare și mentenanță pentru a garanta calitatea și securitatea aplicației.

## **1.2 Domeniul de aplicare**

*Descrierea domeniului aplicației și utilizatorilor vizați.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această aplicație software este destinată gestionării eficiente a proceselor interne ale unei companii, incluzând gestionarea utilizatorilor, monitorizarea activităților și generarea de rapoarte. Utilizatorii vizați includ administratorii de sistem, managerii și angajații, fiecare având acces la funcționalități specifice în funcție de rolurile lor. Aplicația va fi utilizată într-un mediu corporativ și va necesita integrarea cu alte sisteme existente pentru o funcționare optimă.

## **1.3 Definiții, acronime și abrevieri**

*Lista termenilor specifici utilizați în document.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune definește termenii, acronimele și abrevierile utilizate în document pentru a asigura o înțelegere clară și consecventă a conceptelor discutate.

* API (Application Programming Interface) – Un set de reguli și mecanisme care permit aplicațiilor să comunice între ele.
* CI/CD (Continuous Integration/Continuous Deployment) – Practici DevOps care automatizează integrarea codului și livrarea aplicației.
* ERD (Entity-Relationship Diagram) – O diagramă care modelează relațiile dintre entitățile unei baze de date.
* UML (Unified Modeling Language) – Un standard pentru vizualizarea, specificarea și documentarea componentelor unui sistem software.
* UI/UX (User Interface/User Experience) – Designul interfeței utilizatorului și experiența utilizatorului în interacțiunea cu aplicația.

## **1.4 Referințe**

*Listează documente, standarde și surse utilizate pentru elaborarea documentului.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune conține lista documentelor, standardelor și surselor utilizate în procesul de analiză și proiectare a aplicației. Referințele asigură validitatea și conformitatea soluției propuse cu cele mai bune practici și cerințe industriale.

**Documente și Standarde**

* ISO/IEC 25010:2011 – Sisteme și software de inginerie – Modele de calitate pentru produs și utilizare.
* ISO/IEC 27001:2013 – Standarde de securitate a informațiilor. (opțional)
* OWASP Top 10 – Cele mai comune vulnerabilități de securitate software. (opțional)
* UML 2.5 Specification – Standardul utilizat pentru modelarea sistemelor software.

**Surse Web și Bibliografie**

* [exemplu bun de referință] A. Kudriavtseva, "A Software Security Evaluation Framework," *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Lisbon, Portugal, 2024, pp. 150-152, doi: 10.1145/3639478.3639796.
* [exemplu rău de referință] Martin Fowler, "Patterns of Enterprise Application Architecture" – Ghid pentru arhitectura aplicațiilor software.
* "Clean Code" de Robert C. Martin – Principii pentru scrierea unui cod clar și ușor de întreținut.
* Documentația oficială a tehnologiilor utilizate (de ex., PostgreSQL, React, Spring Boot)

## **1.5 Prezentare generală**

*Descriere sumara a conținutului documentului.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Acest document detaliază procesul de analiză și proiectare a aplicației software, acoperind toate aspectele esențiale ale dezvoltării. În prima parte sunt definite scopul, domeniul de aplicare și termenii relevanți. Apoi, documentul prezintă cerințele funcționale și non-funcționale, scenariile de utilizare și constrângerile tehnice. Urmează descrierea arhitecturii software, inclusiv modelul de date și diagrame UML relevante. De asemenea, sunt abordate aspectele legate de implementare, testare, deploy și mentenanță. În final, documentul oferă concluzii și recomandări pentru viitoarele îmbunătățiri ale aplicației.

# **2. Analiza cerințelor**

## **2.1 Cerințe funcționale**

*Enumerarea și descrierea cerințelor funcționale ale aplicației.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Aplicația trebuie să îndeplinească următoarele cerințe funcționale pentru a asigura o utilizare eficientă și o experiență optimă pentru utilizatori:

1. Autentificare și autorizare
   * Utilizatorii trebuie să se autentifice folosind un nume de utilizator și o parolă.
   * Sistemul trebuie să suporte autentificare multi-factor (MFA) (optional)
   * Administratorii pot gestiona drepturile de acces ale utilizatorilor.
2. Gestionarea utilizatorilor
   * Crearea, editarea și ștergerea conturilor utilizatorilor.
   * Atribuirea rolurilor și permisiunilor fiecărui utilizator.
3. Monitorizarea activităților
   * Înregistrarea acțiunilor efectuate de utilizatori în jurnalul de audit.
   * Generarea de rapoarte de activitate pe baza datelor colectate.
4. Generarea de rapoarte
   * Utilizatorii trebuie să poată genera și descărca rapoarte personalizate.
   * Exportul rapoartelor în formate standard (PDF, CSV, Excel).
5. Notificări și alerte
   * Sistemul trebuie să trimită notificări prin e-mail și SMS pentru evenimente importante (ex. resetare parolă, alerte de securitate). (optional)
6. Integrare cu alte sisteme (opțional)
   * Aplicația trebuie să poată comunica cu alte sisteme prin API-uri RESTful.
   * Importul și exportul de date între platforme compatibile.

Aceste cerințe vor fi detaliate în secțiunile ulterioare prin diagrame UML și scenarii de utilizare.

## **2.2 Cerințe non-funcționale**

*Performanță, scalabilitate, securitate, interoperabilitate etc.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Aplicația trebuie să îndeplinească următoarele cerințe non-funcționale pentru a asigura performanța, securitatea și scalabilitatea necesare utilizatorilor:

1. **Performanță (optional)**
   * Sistemul trebuie să răspundă la solicitările utilizatorilor în mai puțin de 2 secunde pentru operațiunile comune.
   * Timpul de încărcare al interfeței nu trebuie să depășească 3 secunde în condiții normale de utilizare.
2. **Scalabilitate (optional)**
   * Aplicația trebuie să poată gestiona până la 10.000 de utilizatori activi simultan.
   * Arhitectura trebuie să permită extinderea resurselor (scalare orizontală și verticală) fără întreruperi majore.
3. **Securitate** 
   * Toate datele sensibile trebuie criptate folosind algoritmi de criptare standard (ex: AES-256 pentru stocare, TLS 1.3 pentru comunicații) (opțional)
   * Sistemul trebuie să implementeze politici stricte de autentificare și autorizare (inclusiv MFA și OAuth 2.0) (opțional)
   * Jurnalizarea și auditul trebuie să fie active pentru toate acțiunile utilizatorilor.
4. **Interoperabilitate**
   * Aplicația trebuie să suporte integrarea cu alte sisteme utilizând API-uri RESTful și standarde de comunicație precum JSON și XML.
   * Compatibilitate cu baze de date SQL și NoSQL pentru gestionarea eficientă a datelor.
5. **Fiabilitate și disponibilitate**
   * Aplicația trebuie să aibă o disponibilitate de cel puțin 99.9% (SLA de tip "high availability") (optional).
   * Sistemul trebuie să suporte backup-uri automate și recuperare rapidă în caz de eșec (optional).
6. **Ușurință în utilizare**
   * Interfața utilizatorului trebuie să fie intuitivă și accesibilă, conform standardelor UI/UX actuale.
   * Sistemul trebuie să includă documentație și suport pentru utilizatori (opțional).

## **2.3 Cazuri de utilizare (Use Cases)**

*Descrierea scenariilor de utilizare cu diagrame UML corespunzătoare.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune prezintă principalele cazuri de utilizare ale aplicației, ilustrate prin diagrame UML relevante.

**UC1: Autentificare utilizator**

**Descriere:** Utilizatorul se autentifică în aplicație folosind un nume de utilizator și o parolă. **Actori:** Utilizator, Sistem de Autentificare

**Pași:**

1. Utilizatorul accesează pagina de login.
2. Introduce numele de utilizator și parola.
3. Sistemul verifică credențiale.
4. Dacă autentificarea este reușită, utilizatorul este direcționat către pagina principală.
5. Dacă autentificarea eșuează, sistemul afișează un mesaj de eroare.

**UC2: Gestionare utilizatori**

**Descriere:** Un administrator poate adăuga, edita sau șterge utilizatori în sistem.

**Actori:** Administrator

**Pași:**

1. Administratorul accesează modulul de gestionare a utilizatorilor.
2. Selectează opțiunea de creare, editare sau ștergere utilizator.
3. Completează informațiile necesare.
4. Salvează modificările.
5. Sistemul validează și aplică schimbările.

**UC3: Generare rapoarte**

**Descriere:** Un utilizator poate genera un raport personalizat bazat pe datele sistemului.

**Actori:** Utilizator, Sistem de Raportare

**Pași:**

1. Utilizatorul selectează tipul de raport dorit.
2. Specifică filtrele și criteriile necesare.
3. Sistemul procesează cererea și generează raportul.
4. Utilizatorul descarcă raportul în formatul dorit (PDF, CSV, Excel).

## **2.4 Constrângeri**

*Limitări tehnice, legale sau operaționale.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune identifică principalele constrângeri tehnice, legale și operaționale care pot afecta dezvoltarea și implementarea aplicației.

1. **Constrângeri tehnice**
   * Aplicația trebuie să fie compatibilă cu sistemele de operare Windows și Linux.
   * Performanța trebuie optimizată pentru a funcționa pe dispozitive cu resurse limitate (ex: 4GB RAM, procesor dual-core).
   * Baza de date trebuie să suporte minim 100.000 de înregistrări fără degradarea performanței.
2. **Constrângeri legale**
   * Aplicația trebuie să fie conformă cu regulamentul GDPR pentru protecția datelor personale.
   * Respectarea standardelor ISO/IEC 27001 privind securitatea informațiilor (opțional)
   * Restricții privind colectarea și stocarea anumitor tipuri de date sensibile (optional)
3. **Constrângeri operaționale**
   * Disponibilitatea aplicației trebuie să fie de cel puțin 99,9% conform SLA (Service Level Agreement) (optional)
   * Sistemul trebuie să permită operarea de la distanță, cu suport pentru conexiuni securizate VPN (optional).
   * Timpul de răspuns pentru acțiuni critice nu trebuie să depășească 3 secunde (optional)

## **2.5 Riscuri**

*Identificarea riscurilor și modalitățile de gestionare a acestora.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune identifică principalele riscuri asociate cu dezvoltarea și implementarea aplicației, precum și modalitățile de gestionare a acestora.

1. **Riscuri tehnice**
   * **Incompatibilitate hardware/software** – Aplicația ar putea să nu fie compatibilă cu anumite dispozitive sau sisteme de operare.
     + *Măsură de atenuare:* Testare extinsă pe multiple platforme și configurații.
   * **Defecțiuni critice în producție** – Erori neprevăzute care pot afecta funcționarea aplicației.
     + *Măsură de atenuare:* Implementarea unui sistem de backup și rollback pentru recuperare rapidă.
2. **Riscuri de securitate**
   * **Acces neautorizat la date** – Posibile atacuri cibernetice asupra bazei de date și componentelor aplicației.
     + *Măsură de atenuare:* Implementarea autentificării MFA, criptarea datelor și monitorizare constantă (optional).
   * **Vulnerabilități în codul sursă** – Exploatarea unor puncte slabe ale aplicației de către atacatori.
     + *Măsură de atenuare:* Realizarea de audituri periodice și scanări de Securitate (optional).
3. **Riscuri de proiect**
   * **Depășirea termenelor limită** – Întârzieri cauzate de complexitatea proiectului sau resurse insuficiente.
     + *Măsură de atenuare:* Planificare riguroasă și utilizarea metodologiilor agile (ex. Scrum, Kanban) (optional).
   * **Depășirea bugetului** – Costuri neprevăzute pentru infrastructură sau licențe software.
     + *Măsură de atenuare:* Monitorizare financiară constantă și optimizare a resurselor utilizate (optional).
4. **Riscuri operaționale**
   * **Lipsa de training pentru utilizatori** – Utilizatorii finali pot avea dificultăți în utilizarea aplicației.
     + *Măsură de atenuare:* Crearea unei documentații detaliate și organizarea de sesiuni de training (optional).
   * **Timp de nefuncționare (downtime)** – Indisponibilitatea aplicației în anumite momente critice.
     + *Măsură de atenuare:* Implementarea unei arhitecturi redundante și mecanisme de failover (optional).

# **3. Arhitectura și proiectarea sistemului**

## **3.1 Arhitectura generală**

*Prezentarea arhitecturii software (de ex. client-server, microservicii, MVC).*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această aplicație este construită pe o arhitectură de tip **microservicii**, utilizând un model **client-server** pentru comunicare între componente. Arhitectura este împărțită în următoarele niveluri:

1. **Nivelul de prezentare (Frontend):**
   * Implementat folosind **React.js** pentru interfața utilizatorului.
   * Comunică cu backend-ul prin API-uri RESTful.
   * Utilizează un sistem de autentificare bazat pe OAuth 2.0 (optional)
2. **Nivelul de logică de business (Backend):**
   * Dezvoltat în **Spring Boot** pentru gestionarea logicii aplicației.
   * Expune servicii REST pentru interacțiunea cu frontend-ul și alte module.
   * Implementare a unor microservicii dedicate pentru autentificare, gestionarea utilizatorilor și generarea de rapoarte.
3. **Nivelul de date:**
   * Utilizează **PostgreSQL** pentru stocarea datelor structurate.
   * Redis este folosit pentru cache și reducerea timpului de răspuns.
   * Implementarea unui mecanism de backup periodic pentru protecția datelor.
4. **Securitate:**
   * Toate datele sunt transmise prin **TLS 1.3** pentru criptare. (optional)
   * Implementare de politici stricte de acces prin RBAC (Role-Based Access Control). (opțional)
   * Logare și monitorizare activă a evenimentelor de securitate prin **ELK Stack** (Elasticsearch, Logstash, Kibana) (opțional).
5. **Deployment și scalabilitate:**
   * Aplicația este containerizată folosind **Docker** și orchestrată prin **Kubernetes** pentru scalabilitate.
   * CI (Continuos Integration)/CD (Continuous Development) implementat cu **GitHub Actions** pentru automatizarea livrării software. (doar dacă folosiți cloud)
   * Sistem distribuit pentru disponibilitate ridicată (99,9% SLA). (opțional)

## **3.2 Modelul de date**

*Diagrama entitate-relație (ERD), descrierea tabelelor și relațiilor.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune prezintă modelul de date utilizat în aplicație, incluzând structura bazei de date și relațiile dintre entități. Modelul este reprezentat printr-o diagramă entitate-relație (ERD) și descrierea tabelului principalelor entități.

**Diagrama Entitate-Relatie (ERD)**

Diagrama ERD prezintă principalele entități și relațiile dintre ele:

1. **Utilizator** *(User)* – Reprezintă un utilizator al sistemului.
2. **Rol** *(Role)* – Definește tipurile de acces și permisiuni.
3. **Permisiune** *(Permission)* – Specifică acțiunile permise pentru un rol.
4. **Autentificare** *(Authentication)* – Stochează datele de autentificare ale utilizatorilor.
5. **Raport** *(Report)* – Conține informații generate pentru utilizatori.
6. **Activitate** *(Activity)* – Înregistrează acțiunile efectuate de utilizatori.
7. **Sesiune** *(Session)* – Reprezintă sesiunile active ale utilizatorilor.

**Descrierea entităților și relațiilor**

* **Utilizator** are un **Rol** (relație de tip 1-N).
* **Rol** poate avea mai multe **Permisiuni** (relație de tip M-N).
* **Utilizator** poate avea mai multe **Sesiuni** active (relație 1-N).
* **Utilizator** poate genera **Rapoarte** și înregistra **Activități**.

## **3.3 Diagrame UML (Unified Modelling Language)**

Această secțiune prezintă principalele diagrame UML utilizate pentru modelarea aplicației.

**1. Diagrama de clase**

Diagrama de clase descrie structura aplicației, evidențiind entitățile principale și relațiile dintre ele.

* **Clase principale:** Utilizator, Rol, Permisiune, Autentificare, Raport Activitate, Sesiune.
* **Relații:**
  + Utilizator ↔ Rol (1-N)
  + Rol ↔ Permisiune (M-N)
  + Utilizator ↔ Sesiune (1-N)
  + Utilizator ↔ Raport\_Activitate (1-N)

**2. Diagrama de secvențe**

Această diagramă ilustrează fluxul principal al unei acțiuni din sistem, cum ar fi autentificarea utilizatorului:

1. Utilizatorul introduce acreditările.
2. Sistemul validează datele.
3. Dacă autentificarea este reușită, sistemul creează o sesiune și direcționează utilizatorul către dashboard.
4. Dacă autentificarea eșuează, se afișează un mesaj de eroare.

**3. Diagrama de stare (optional)**

Prezintă stările prin care trece un utilizator în timpul utilizării aplicației:

* **Stări:** Deconectat → Autentificat → Activ → Inactiv → Deconectat
* **Tranziții:**
  + De la „Deconectat” la „Autentificat” prin procesul de login.
  + De la „Activ” la „Inactiv” după un timp de inactivitate.
  + De la „Inactiv” la „Deconectat” dacă sesiunea expiră.

**4. Diagrama de activitate (optional)**

Descrie fluxul proceselor în aplicație. De exemplu, procesul de generare a unui raport:

1. Utilizatorul accesează secțiunea de rapoarte.
2. Selectează tipul de raport dorit.
3. Specifică filtre și criterii.
4. Aplicația procesează cererea.
5. Raportul este generat și oferit pentru descărcare

## **3.4 Tehnologii și stack de dezvoltare**

*Enumerarea tehnologiilor utilizate (limbaje de programare, baze de date, framework-uri etc.).*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

1. **Date:**
   * **SGBD:** PostgreSQL
   * **Caching:** Redis
   * **Migrarea datelor:** Flyway
2. **DevOps & deployment:**
   * **Containerizare:** Docker
   * **Orchestrare:** Kubernetes
   * **CI/CD:** GitHub Actions
   * **Monitorizare:** Prometheus, Grafana
3. **Alte tehnologii:**
   * **Testare:** JUnit, Selenium
   * **Logare și analiză:** ELK Stack (Elasticsearch, Logstash, Kibana)
   * **Cloud & Hosting:** AWS, Azure

## **3.5 Interfața utilizatorului (UI/UX)**

*Descrierea designului interfeței utilizatorului, mockups, wireframes.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune descrie designul interfeței utilizatorului, principiile UI/UX aplicate și prototipurile utilizate pentru dezvoltare.

**1. Principii UI/UX aplicate**

* **Ușurință în utilizare:** Design intuitiv, navigare simplă și clară.
* **Responsivitate:** Interfața trebuie să fie adaptabilă pentru desktop, tabletă și mobil.
* **Accesibilitate:** Respectarea standardelor WCAG pentru utilizatori cu dizabilități.
* **Design modern:** Folosirea Material Design și Tailwind CSS pentru o experiență vizuală optimizată.

**2. Structura interfeței**

* **Pagina de Autentificare:** Formulare simple cu suport pentru autentificare bazată pe credențiale (suficient pentru proiect) sau autentificare tip multi-factor (MFA) (optional).
* **Dashboard principal:** Prezentare generală a informațiilor relevante pentru utilizator.
* **Meniu de navigare:** Acces rapid la modulele aplicației (ex: utilizatori, rapoarte, setări).
* **Pagini de gestionare:** Interfețe specifice pentru gestionarea utilizatorilor, rapoartelor și activităților.
* **Notificări și alerte:** Feedback vizual și auditiv pentru acțiuni importante.

**3. Protocoale și tehnologii UI**

* **Framework Frontend:** React.js cu Material-UI.
* **Biblioteci de stilizare:** Tailwind CSS și Bootstrap pentru layout flexibil.
* **Protocoale de accesibilitate:** Suport pentru navigare cu tastatura și cititoare de ecran (optional).

**4. Wireframes și mockups (optional)**

* Crearea unor prototipuri interactive folosind **Figma** și **Adobe XD**.
* Testarea designului cu utilizatori înainte de implementare.
* Iterații bazate pe feedback pentru îmbunătățirea experienței utilizatorilor.

# **4. Implementare**

## **4.1 Structura codului**

*Descrierea modului în care este organizat codul aplicației.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Codul aplicației este organizat pe baza unei arhitecturi modulare, precum NUME ARHITECTURĂ, urmând principiile separării responsabilităților și dezvoltării scalabile. Structura principală este următoarea:

1. **Frontend** (React.js):
   * /src/components/ – Componentele reutilizabile ale interfeței utilizatorului.
   * /src/pages/ – Pagini principale ale aplicației (ex: Dashboard, Login, Profile).
   * /src/store/ – Gestionarea stării aplicației cu Redux.
   * /src/services/ – Apelurile API și gestionarea interacțiunii cu backend-ul.
   * /src/styles/ – Fișiere CSS și teme pentru stilizare.
2. **Backend** (Spring Boot):
   * /src/main/java/com/proiect/ – Structura codului backend.
     + /controller/ – Controlerele API-urilor expuse.
     + /service/ – Implementarea logicii de business.
     + /repository/ – Interacțiunea cu baza de date folosind JPA.
     + /model/ – Entitățile și DTO-urile utilizate în sistem.
     + /security/ – Configurarea autentificării și autorizării.
3. **Baza de date**:
   * /db/migrations/ – Scripturi SQL pentru crearea și modificarea structurilor bazei de date.
   * /db/seeds/ – Date predefinite pentru popularea bazei de date inițial.
4. **DevOps & configurație**:
   * /docker/ – Fișierele Docker pentru rularea aplicației containerizate.
   * .github/workflows/ – Configurația CI/CD cu GitHub Actions.
   * /logs/ – Fișierele de log generate de aplicație pentru debugging și audit.

## **4.2 Module și componente**

*Prezentarea modulelor software și a funcționalității fiecăruia.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune descrie modulele software ale aplicației și funcționalitatea fiecăruia, organizate pe arhitectura microservicii.

**1. Modul de autentificare și autorizare**

* Responsabil pentru gestionarea autentificării utilizatorilor (OAuth 2.0, JWT) (opțional)
* Permite utilizarea autentificării multi-factor (MFA) (optional).
* Verifică și atribuie permisiunile pe baza rolurilor definite în sistem.

**2. Modul de gestionare a utilizatorilor**

* Permite crearea, editarea, activarea/dezactivarea utilizatorilor.
* Atribuie și gestionează rolurile utilizatorilor.
* Stochează istoricul modificărilor asupra conturilor.

**3. Modul de generare a rapoartelor**

* Permite utilizatorilor să genereze rapoarte personalizate.
* Suportă exportul în formate multiple (PDF, CSV, Excel).
* Utilizează un motor de procesare pentru generarea eficientă a rapoartelor.

**4. Modul de monitorizare și audit**

* Înregistrează toate acțiunile utilizatorilor într-un jurnal de audit (recomandat să existe o structură de tipul data și oră / utilizator / descriere acțiune).
* Permite vizualizarea activității utilizatorilor pentru conformitate și securitate.
* Integrează notificări și alerte pentru evenimente critice (opțional).

**5. Modul de integrare API**

* Expune API-uri RESTful pentru interacțiunea cu alte sisteme (la cerere dar opțional).
* Suportă autentificare API prin OAuth 2.0 (optional).
* Asigură interoperabilitate cu servicii externe prin webhook-uri și mesagerie (doar la nevoieîn funcție de proiectul ales).
* Această modularizare permite scalabilitate, securitate și o mentenanță ușoară a aplicației (optional).

## **4.3 Fluxul de date**

*Descrierea modului în care datele sunt procesate în aplicație.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune descrie modul în care datele circulă prin diferitele componente ale aplicației, de la introducerea lor de către utilizatori până la procesare și stocare.

**1. Fluxul general al datelor**

* Utilizatorul introduce datele prin interfața aplicației (frontend - React.js).
* Datele sunt validate la nivelul clientului și apoi transmise către backend (Spring Boot) prin API REST.
* Backend-ul procesează cererea și interacționează cu baza de date PostgreSQL.
* După procesare, datele sunt returnate către frontend și afișate utilizatorului.

**2. Procesarea datelor**

* **Validare**: Toate datele primite de la utilizator sunt validate pentru a preveni erori (necesar) sau atacuri de tip SQL Injection/XSS (optional).
* **Autentificare și autorizare**: Utilizatorul trebuie să fie autentificat pentru a accesa datele relevante (necesar), utilizând OAuth 2.0 și JWT (opțional).
* **Stocare**: Datele sunt salvate în PostgreSQL, iar informațiile volatile sunt gestionate prin Redis pentru optimizare.
* **Backup**: Datele critice sunt arhivate periodic pentru a asigura recuperarea în caz de defecțiuni (optional).

**3. Flux de date pentru rapoarte**

* Utilizatorul solicită generarea unui raport din interfață.
* Backend-ul preia datele relevante din baza de date și le procesează.
* Raportul este generat în formatul specificat (PDF, CSV, Excel).
* Utilizatorul primește un link pentru descărcarea raportului.

**4. Notificări și evenimente**

* Evenimente critice (ex: schimbare de parolă, erori) sunt înregistrate și gestionate printr-un sistem de logging (ELK Stack) (necesar dar tool-ul poate să desfire).
* Utilizatorii primesc notificări prin email/SMS pentru acțiuni importante (opțional).

# **5. Testare**

## **5.1 Strategia de testare**

*Definirea abordării pentru testarea aplicației.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Strategia de testare pentru această aplicație se bazează pe o abordare combinată de testare manuală și automată pentru a asigura calitatea, securitatea și performanța software-ului. Procesul de testare se va desfășura în mai multe etape și va include următoarele aspecte:

1. **Testare unitară**
   * Verificarea funcționării corecte a fiecărui modul individual (ex: autentificare, gestionare utilizatori, generare rapoarte).
   * Utilizarea unor framework-uri precum **JUnit** (Java) și **Jest** (React.js) pentru testare automată.
2. **Testare de integrare**
   * Asigurarea faptului că modulele individuale funcționează corect împreună.
   * Testarea API-urilor RESTful utilizând **Postman** și **JUnit cu MockMVC**.
3. **Testare de performanță**
   * Evaluarea timpului de răspuns și a comportamentului aplicației sub sarcină ridicată.
   * Utilizarea **Apache JMeter** pentru simularea a 10.000 de utilizatori simultani.
4. **Testare de securitate (optional)**
   * Identificarea și remedierea vulnerabilităților de securitate.
   * Scanări automate cu **OWASP ZAP** și **Burp Suite**.
   * Testarea protecției împotriva atacurilor XSS, SQL Injection, CSRF.
5. **Testare de acceptanță**
   * Evaluarea aplicației din perspectiva utilizatorilor finali.
   * Crearea scenariilor de testare bazate pe cerințele funcționale.
   * Feedback din partea utilizatorilor pentru optimizarea experienței UI/UX.
6. **Automatizarea testelor**
   * Implementarea testelor automate pentru regresie și validare continuă.
   * Integrarea testelor în pipeline-ul CI/CD folosind **GitHub Actions**.

## **5.2 Tipuri de teste**

De urmărit laboratorul sau alte tutoriale specifice tehnologiei de implementare alese

* Teste unitare
* Teste de integrare
* Teste de performanță
* Teste de securitate
* Teste de acceptanță

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

1. **Teste unitare**
   * Verifică funcționarea corectă a fiecărei componente individuale.
   * Se utilizează **JUnit** pentru backend (Spring Boot) și **Jest** pentru frontend (React.js).
2. **Teste de integrare**
   * Asigură buna funcționare a modulelor interconectate.
   * Se utilizează **Postman** și **Spring Boot Test** pentru validarea API-urilor.
3. **Teste de performanță**
   * Evaluarea vitezei de răspuns și scalabilității aplicației.
   * Testarea este realizată cu **Apache JMeter** pentru simularea utilizatorilor concurenți.
4. **Teste de securitate**
   * Identificarea vulnerabilităților precum SQL Injection, XSS și CSRF.
   * Scanări automate utilizând **OWASP ZAP** și **Burp Suite**.
5. **Teste de acceptanță**
   * Validarea funcționalităților conform cerințelor utilizatorilor finali.
   * Crearea scenariilor de testare bazate pe cerințele documentate.

## **5.3 Plan de testare**

*Planificarea testelor cu scenarii specifice.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune definește strategia de testare a aplicației, incluzând scenariile de testare specifice pentru a verifica conformitatea cu cerințele funcționale și non-funcționale.

**1. Obiectivele testării**

* Asigurarea corectitudinii funcționalităților esențiale.
* Verificarea interoperabilității între module.
* Testarea performanței și securității aplicației.

**2. Scenarii de testare**

**Testarea autentificării**

**Scenariu:** Un utilizator încearcă să se autentifice în aplicație cu credențiale valide.

* *Pași:*
  1. Accesarea paginii de autentificare.
  2. Introducerea numelui de utilizator și a parolei corecte.
  3. Apăsarea butonului „Autentificare”.
  4. Sistemul redirecționează utilizatorul către dashboard.
* *Rezultat așteptat:* Utilizatorul este autentificat cu succes.

**Testarea gestionării utilizatorilor**

**Scenariu:** Un administrator creează un cont nou de utilizator.

* *Pași:*
  1. Accesarea secțiunii „Gestionare utilizatori”.
  2. Selectarea opțiunii „Adaugă utilizator”.
  3. Introducerea datelor noului utilizator (nume, e-mail, rol).
  4. Salvarea modificărilor.
* *Rezultat așteptat:* Utilizatorul este adăugat în baza de date și apare în listă.

**Testarea generării de rapoarte**

**Scenariu:** Un utilizator generează un raport de activitate.

* *Pași:*
  1. Accesarea secțiunii „Rapoarte”.
  2. Selectarea tipului de raport dorit.
  3. Aplicarea filtrelor relevante.
  4. Generarea și descărcarea raportului.
* *Rezultat așteptat:* Raportul este generat și descărcat cu succes în formatul selectat.

**Testarea performanței**

**Scenariu:** Se verifică timpul de răspuns pentru cererile API.

* *Pași:*
  1. Simularea a 1000 de cereri simultane către server.
  2. Monitorizarea timpului de răspuns.
* *Rezultat așteptat:* Timpul mediu de răspuns nu depășește 2 secunde.

**Testarea securității**

**Scenariu:** Se testează vulnerabilitățile la atacuri SQL Injection.

* *Pași:*
  1. Introducerea unui input malițios în câmpurile de login.
  2. Observarea reacției sistemului.
* *Rezultat așteptat:* Sistemul respinge atacul și înregistrează încercarea în jurnalul de securitate.

**3. Instrumente utilizate pentru testare**

* **JUnit** și **Jest** pentru testarea unitară.
* **Postman** pentru testarea API.
* **Apache JMeter** pentru testarea performanței.
* **OWASP ZAP** pentru testarea securității.

# **6. Deployment și mentenanță**

## **6.1 Strategie de deployment**

*Descrierea procesului de implementare a aplicației (CI/CD, servere, containere).*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Procesul de implementare al aplicației utilizează o abordare modernă bazată pe DevOps, asigurând livrarea rapidă și sigură a noilor versiuni.

**1. Medii de deploy**

* **Mediu de dezvoltare** (Development) – Teste inițiale efectuate de echipa de dezvoltare.
* **Mediu de testare** (Staging) – Validare și testare înainte de implementarea în producție.
* **Mediu de producție** (Production) – Versiunea finală accesibilă utilizatorilor finali.

**2. Procesul CI/CD (Continuous Integration / Continuous Deployment)**

1. **Commit & push** – Dezvoltatorii trimit codul în repository-ul GitHub/GitLab.
2. **Testare automată** – Se rulează teste unitare și de integrare utilizând **JUnit**, **Jest**, și **Postman**.
3. **Build automatizat** – Aplicația este construită folosind **Maven/Gradle** pentru backend și **webpack** pentru frontend.
4. **Containerizare** – Codul este împachetat într-un container Docker pentru portabilitate.
5. **Orchestrare** – Aplicația este implementată în Kubernetes pentru scalabilitate.
6. **Deploy automatizat** – Folosirea **GitHub Actions** sau **Jenkins** pentru livrarea noilor versiuni pe medii specifice.
7. **Monitorizare și logging** – Se utilizează **Prometheus**, **Grafana** și **ELK Stack** pentru analiza performanței și detectarea erorilor.

**3. Tehnologii Utilizate**

* **Sisteme de versionare:** GitHub/GitLab
* **Pipeline CI/CD:** GitHub Actions, Jenkins
* **Containerizare:** Docker
* **Orchestrare:** Kubernetes
* **Load Balancing:** Nginx, HAProxy
* **Hosting:** AWS, Azure, Google Cloud

## **6.2 Administrarea și monitorizarea**

*Instrumente utilizate pentru monitorizarea aplicației și a performanței (opțional)*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune descrie strategiile și instrumentele utilizate pentru administrarea și monitorizarea aplicației pentru a asigura performanța, securitatea și disponibilitatea continuă.

**1. Monitorizarea performanței**

* **Prometheus & Grafana** – Colectează și analizează metrice privind utilizarea resurselor (CPU, RAM, latime de bandă).
* **New Relic** – Oferă vizibilitate în timp real asupra performanței aplicației și a tranzacțiilor utilizatorilor.
* **ELK Stack (Elasticsearch, Logstash, Kibana)** – Centralizează și analizează jurnalele de evenimente pentru identificarea rapidă a problemelor.

**2. Monitorizarea securității**

* **Wazuh SIEM** – Detectează activități suspecte și posibile atacuri asupra aplicației.
* **OWASP ZAP** – Scanări periodice pentru vulnerabilități de securitate web.
* **Audit Trail & Logging** – Toate acțiunile critice sunt înregistrate pentru a asigura conformitatea și posibilitatea de audit.

**3. Administrarea aplicației**

* **Kubernetes Dashboard** – Monitorizarea clusterului și gestionarea containerelor.
* **Docker Swarm** – Administrarea serviciilor containerizate.
* **CI/CD Pipeline Logs** – Analiza erorilor din pipeline-ul de deploy (GitHub Actions, Jenkins).

**4. Notificări și alerte**

* **Slack & Email Alerts** – Alerte automate pentru evenimente critice (ex: downtime, erori de securitate, depășirea resurselor alocate).
* **PagerDuty** – Escaladarea alertelor către echipele de intervenție.

## **6.3 Plan de mentenanță**

*Actualizări, corecturi de erori, suport tehnic.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Planul de mentenanță asigură funcționarea optimă și actualizarea continuă a aplicației, incluzând intervenții proactive și reactive pentru remedierea problemelor și îmbunătățirea performanței.

**1. Tipuri de mentenanță**

* **Mentenanță corectivă** – Identificarea și remedierea erorilor raportate de utilizatori sau detectate prin sistemele de monitorizare.
* **Mentenanță preventivă** – Actualizări periodice pentru prevenirea potențialelor probleme și optimizarea performanței.
* **Mentenanță evolutivă** – Adăugarea de noi funcționalități și îmbunătățiri în funcție de cerințele utilizatorilor.
* **Mentenanță adaptivă** – Ajustări pentru compatibilitatea cu noi tehnologii și platforme.

**2. Program de mentenanță**

* **Zilnic:** Monitorizarea performanței aplicației și jurnalizarea alertelor de securitate.
* **Săptămânal:** Revizuirea jurnalelor de erori și remedierea bug-urilor critice.
* **Lunar:** Aplicarea actualizărilor de securitate și testarea compatibilității cu infrastructura IT.
* **Trimestrial:** Revizuirea performanței generale și optimizarea componentelor cheie.
* **Anual:** Planificarea îmbunătățirilor arhitecturale și migrarea către tehnologii mai eficiente.

**3. Instrumente și procese de mentenanță**

* **Gestionarea Ticketelor:** Utilizarea **JIRA** sau **Trello** pentru urmărirea problemelor și solicitărilor utilizatorilor.
* **Automatizarea Patch-urilor:** Aplicarea corecțiilor automate prin **Ansible** sau **Terraform**.
* **Testare după Actualizări:** Implementarea testelor automate pentru validarea stabilității aplicației după fiecare actualizare.
* **Backup și Recuperare:** Realizarea backup-urilor automate și testarea periodică a procesului de restaurare.

**4. Suport tehnic**

* **Asistență 24/7:** Sistem de ticketing și suport prin email/chat.
* **Documentație:** Un wiki intern și ghiduri de utilizare pentru echipa tehnică și utilizatori.
* **Canale de comunicare:** Slack, Microsoft Teams pentru escaladarea problemelor și colaborarea rapidă.

# **7. Concluzii**

*Sumarizarea concluziilor principale și recomandări pentru dezvoltări viitoare.*

[EXEMPLU – Se ajustează pentru proiectul fiecăruia]

Această secțiune oferă o sinteză a principalelor concluzii rezultate din analiza și proiectarea aplicației, precum și recomandări pentru dezvoltări viitoare.

**1. Rezumat al principalelelor constatări**

* Aplicația este proiectată utilizând o arhitectură scalabilă bazată pe microservicii, ceea ce permite extinderea și integrarea ușoară cu alte sisteme.
* Cerințele funcționale și non-funcționale au fost clar definite pentru a asigura o experiență de utilizare optimă.
* Implementarea tehnologiilor moderne precum **React.js**, **Spring Boot**, **PostgreSQL**, **Docker** și **Kubernetes** contribuie la performanța și securitatea aplicației.
* Strategia de testare cuprinde teste unitare, de integrare, de performanță și de securitate pentru a minimiza riscurile asociate implementării.
* Sistemul de administrare și monitorizare asigură o disponibilitate ridicată și o gestionare eficientă a resurselor.

**2. Recomandări pentru dezvoltări viitoare**

* **Optimizarea performanței:** Monitorizarea constantă a performanței și identificarea eventualelor îmbunătățiri în procesarea datelor și timpii de răspuns.
* **Extinderea funcționalităților:** Adăugarea de noi module, cum ar fi raportare avansată bazată pe AI sau integrarea cu sisteme externe pentru automatizare.
* **Îmbunătățirea experienței utilizatorilor:** Efectuarea de teste de utilizabilitate și ajustarea interfeței pe baza feedbackului utilizatorilor.
* **Consolidarea securității:** Implementarea unor mecanisme avansate de protecție, precum monitorizarea comportamentală și detecția automată a amenințărilor.
* **Automatizarea proceselor de deploy și mentenanță:** Extinderea pipeline-ului CI/CD și îmbunătățirea proceselor de rollback și disaster recovery.