Titlul lucrării

PROIECT DE DIPLOMĂ

Autor: **Prenume NUME**

Conducător științific: **Titlu. ing. Prenume NUME**

|  |  |
| --- | --- |
| DECAN  **Prof. dr. ing. Liviu MICLEA** | Vizat,  DIRECTOR DEPARTAMENT AUTOMATICĂ  **Prof. dr. ing. Honoriu VĂLEAN** |

Autor: **Prenume NUME**

Titlul lucrării

1. **Enunțul temei:** *O scurtă descriere a temei proiectului de diplomă*
2. **Conținutul proiectului:** *(enumerarea părților componente) Pagina de prezentare, Declarație privind autenticitatea proiectului, Sinteza proiectului, Cuprins, Titlul capitolului 1, Titlul capitolului 2,… Titlul capitolului n, Bibliografie, Anexe.*
3. **Locul documentării:** *Universitatea Tehnică din Cluj-Napoca, alte locuri dacă este cazul*
4. **Consultanți:** *ing. Prenume Nume (dacă este cazul)*
5. **Data emiterii temei:**
6. **Data predării:**

Semnătura autorului

Semnătura conducătorului științific

**Declaraţie pe proprie răspundere privind**

**autenticitatea proiectului de diplomă**

Subsemnatul(a) **Prenume NUME**  , legitimat(ă) cu CI/BI seria nr. , CNP ,

autorul lucrării:

elaborată în vederea susținerii examenului de finalizare a studiilor de licență la **Facultatea de Automatică și Calculatoare**, specializareaChoose an item.**,** din cadrul Universității Tehnice din Cluj-Napoca, sesiunea Choose an item. a anului universitar 2022-2023, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data Prenume NUME

(semnătura)

**SINTEZA**

proiectului de diplomă cu titlul:

Titlul lucrării

Autor: **Prenume NUME**

Conducător științific: **Titlu. ing. Prenume NUME**

1. Cerințele temei:

2. Soluții alese:

3. Rezultate obținute:

4. Testări și verificări:

5. Contribuții personale:

6. Surse de documentare:

Semnătura autorului

Semnătura conducătorului științific

Cuprins

[1 Introducere 2](#_Toc136375228)

[1.1 Context general 2](#_Toc136375229)

[1.2 Obiective 2](#_Toc136375230)

[1.3 Specificații 2](#_Toc136375231)

[2 Studiu bibliografic 3](#_Toc136375232)

[2.1 Aplicație WEB 3](#_Toc136375233)

[3 Analiză, proiectare, implementare 13](#_Toc136375234)

[4 Concluzii 14](#_Toc136375235)

[4.1 Rezultate obținute 14](#_Toc136375236)

[4.2 Direcții de dezvoltare 14](#_Toc136375237)

[5 Reguli de formatare 15](#_Toc136375238)

[5.1 Formatarea paginii 15](#_Toc136375239)

[5.2 Titluri și stiluri 15](#_Toc136375240)

[5.3 Figuri, tabele și ecuații 16](#_Toc136375241)

[5.3.1 Figuri 16](#_Toc136375242)

[5.4 Tabele 16](#_Toc136375243)

[5.5 Ecuații 16](#_Toc136375244)

[5.6 Referințe bibliografice 17](#_Toc136375245)

[6 Bibliografie 18](#_Toc136375246)

# Introducere

## Context general

În introducere familiarizați cititorul cu motivația lucrării, plasați lucrarea într-un context care să permită cititorului să înțeleagă obiectivele.

Descrieți importanța lucrării, de ce merita să o faceți, plasați ideile într-un context larg.

Susțineți studiul: de ce exact aceasta aplicație/implementare. Comentați asupra aspectelor teoretice sau practice care v-au făcut să o alegeți.

În funcție de natura lucrării, ar putea fi necesar să prezentați informații de fond asupra domeniului în care se încadrează aplicația, mai ales dacă aveți o lucrare într-un domeniu multidisciplinar. În acest caz, puteți introduce terminologia pe care o utilizați în continuare.

Descrieți pe scurt lucrarea: ce conține fiecare capitol.

## Obiective

Enumerați și explicați obiectivele lucrării: ce v-ați propus să realizați în contextul prezentat anterior.

Obiectivele pot fi prezentate sub formă de listă care să evidențieze precis orientarea lucrării, să identifice conceptele fundamentale pe care le studiați, să stabilească scopul aplicației pe care o realizați, sau enunțați întrebările la care intenționați să răspundeți în lucrare.

## Specificații

În specificațiile lucrării detaliați cerințele. Descrieți ce intenționați să obțineți. Vă puteți referi la funcțiile aplicației, interfață, nivele de performanță, structuri de date, elemente, securitate, fiabilitate, calitate, limitări, etc.

# Studiu bibliografic

## Aplicație WEB

O aplicație web este un software care rulează într-un browser web, cum ar fi Google Chrome sau Mozilla FireFox, aceasta este stocată intr-un server care se află la distanță și se conectează la interfața aflată în browser prin intermediul internetului.

Majoritatea caracteristicilor din aplicațile folosite în ziua de azi sunt la bază aplicații web, cum ar fi: cărucioarele de cumpărături de la magazinele online sau fluxurile de informații din rețelele de socializare.

Acest tip de aplicații oferă posibilitatea de a accesa anumite funcționalități complexe fără a fi nevoie de un software care să fie instalat sau configurat în prealabil, este nevoie doar o conexiune la internet pentru a se putea accesa toate funcționalitățile disponibile pe serverul aplicației, acest avantaj simplifică aplicația pentru utilizator fiind mult mai ușor de folosit. În același timp se elimină și nevoia de mentenanță pentru spațiul de stoacare. Cu alte cuvinte aplicația este mereu actualizată la ultima versiune crescând securitatea utilizatorilor.

Unul dinre avantajele importante este că aplicația poate sa fie accesată concomitent de mai mulți utilizatori fără a fi nevoie de o infrastructură îmbunătățită a aplicației. Un alt avantaj este posibilitatea de a accesa o aplicație prin intermediul mai multor web browser-e și cu o gamă largă de dispozitive, din orice loc, singura cerință fiind nevoia de conexiune la internet.

Aplicaștile Web folosesc arhitectura client-server, tot codul este împărțit în două componente, componenta clientului care conține interfața și componenta serverului care conține medodele, clasele și logica de funcționalitate a aplicației.

Componenta clientului se ocupă cu funcționalitatea interfeței cum ar fi: butoanele, check box-urile și dropdown-urile. Fiecare acțiune a userului cu interfața, de exemplu un clcik pe un buton, trimite o cerere (request) de la utilizator către sever.

Componenta serverului se ocupă cu procesarea de date. Serverul procesează cererea clientului si trimite înapo un răspuns în funcție de modul în care este implementată logica de gestionare a datelor. Cererile primite de server în general se referă la posibilitatea de a primii mai multe informații din baza de date, de a edita informații din baza de date sau pentru a salva informații noi. Răspunsul trimis de server este informația cerută sau un mesaj care arată daca cererea s-a realizat cu succes sau nu, aceste cereri sunt de tip HTTP.

## Conceptele HTTP

Acronimul HTTP vine de la Hyper Text Transfer Protocol, este modul prin care browserul web adică componenta client comunică cu componenta sever. O astfel de cerere este facută de client către un port care aparține de un server, cererea are ca obiective accesarea unei resurse aflată în baza de date a sistemului. Această bază de date nu poate sa fie accesată în mod direct.

Acest protocol definește mai multe concepte cheie care exemplifică modul de funcționare. HTTP conține mai multe metode pentru interacțiunea cu resursele serverului, cum ar fi: metoda GET, care face o cerere pentru o resursă prin intermediul serverului, metoda POST trimite informații către server pentru a fi procesate, de cele mai multe ori această metodă este folosită pentru a creea resurse noi în baza de date, metoda PUT care este folosită pentru a actualiza o resursă deja existentă, metoda DELETE se ocupă de ștergerea uneia sau a mai multor resurse din baza de date. Aceste metode constituie CRUD care este un acronim pentru create, read, update, delete, acestea fiind acțiunile metodelor de bază.

Un alt concept important poartă denumirea de HTTP headers care oferă informații adiționale despre cererile către server si raspunsurile de la server. Acest headers este o componentă dintr-o cerere trimisă de client pentru a primii informații despre autorizări sau cookie-uri intr-o aplicație.

Componenta headers care apare într-un raspuns trimis de către server pentru a oferii informații suplimentare despre rasounsul din care face parte cuim ar fi: un mesaj de succes sau o eroare, în funcție de cum a decurs cererea trimisă în prima parte. Ultima componentă a unei cereri HTTP este reprezentată de un Response Body (Corpul răspunsului) acesta reprezentând conținutul mesajului propriu-zis.

## API

API (Application Programming Interface) este un set de reguli și de protocoale careoferă aplicaților software posibilitatea de a comunica și interacționa între ele.

Un API se bazează pe modelul cerere-răspuns (request-response) în care o aplicație face o cerere pentru o anumită informație sau acțiune către o altă aplicație care răspunde cu informația cerută sau realizează acțiunile cerute.

API-urile oferă un nivel de abstractizare care ascunde complexitatea programelor cu care lucrează, astfel programatorul poate să lucreze cu o interfață simplificată fără a fi nevoie ca acesta să înțeleagă toată logica de programare din spatele metodelor pe care le folosește.

API-urile se construiesc după anumite standarde și protocoale cum ar fi HTTP sau REST pentru a asigura consecvență și interoperabilitate în toate sistemele în care sunt folosite.

Există mai multe tipuri de API-uri:

* API-urile Web care sunt folosite pentru comunicarea prin intermediul internetului, acestea sunt bazate pe protocolul HTTP și de obieci folosesc principiile RESTful pentru a creea,citii, actualiza și șterge
* Există API-uri care sunt librării sau framework-uri deja construite pe care programatorii le pot folosii în aplicațile lor. Acestea oferă metode care simplifică dezvoltarea unei aplicații prin refolosirea codului
* Sistemele de operare dispun de API-uri care permit ca aplicațiile să interacționeze cu resursele sistemului cum ar fi: fișiere, procese sau conexiuni la internet. Aceste API-uri permit programatorilor să creeze aplicații care sunt specifice unei singure platforme.
* Există și API-uri pentru bazele de date cum ar fi: JDBC pentru JAVA ODBC pentru alte limbaje de programare, prin aceste API-uri se poate comunica cu baza de date pentru a se executa comezni specifice acestora ca preluarea sau modificarea informaților

Folosirea API-urilor vine cu niște beneficii importanate:

* Cu ajutorul acestor un sistem foarte complex se poate împărții în componente mai mici și mai usor de refolosit între mai multe simste, făcand codul mai ușor de urmărit
* Interoperabilitatea oferă posibilitatea pentru diferite aplicații, servicii sau sisteme de a lucra împreuna, promovând astfel integrarea între diferite tehnologii
* Cum exista API-uri care oferă componente deja construite care se pot integra într-o aplicație se reduce timpul de dezvoltare și se reduce efortul dezvoltării unei noi aplicașii nemaifiind nevoie de a scrie cod care deja este folosit pentru a altă aplicație
* API-urile au un ecosistem unde programatorii pot să își expună servicile care pot mai departe să fie folosite de alți programatori pentru diferite proiecte astfel se promovează inovațile

## MVC Framework

MVC este acronimul de la Model View Controller, acesta fiind un șablon de arhitectură pentru o aplicație software. Acesta oferă un mod de a structura modul prin care o aplicație este construită, separând aceasta în trei componente separate: Model, View și Controller. Fiecare din aceste componente este destinată pentru un aspect specific al aplicației.

Aici pune o diagrama cu MVC

### Model

Model este componenta care se ocupă cu logica legată de datele/informațile? cu care utilizatorul poate să lucreze. Această componentă ține toată informația incapsulată pentru a putea fii acesată doar prin intermediul unor metode speciale care se ocupă cu manipularea acestor informații, cum ar fii obținerea informaților din baza de date sau validarea acestora.

### View

Componenta View este reprezentată de interfața utilizatorului. Aceasta definește modul in care informația de la componenta model este reprezentată pentru utilizatorul aplicației. O astfel de componentă de obicei este alcătuita din elemente de HTML, CSS si JavaScript conținând butoane, dropdown-uri și check box-uri cu care utilizatorul poate interacționa.

### Controller

Rolul componentei Controller este de a asigura un canal intre componenta Model și View.Aceasta componentă există pentru a procesa inputul primit de la utlilizator după care sa fie procesat si transmis componentei Model in funcție de request-ul primit. Cu ajutorul Controller-ului se asigură separarea si independența fiecaărei componente. Controllerul este componenta care decide modul în care informatia este folosită pentru întreaga aplicație folosinduse de View pentru a afișa aceste informații si de Model pentru a le obține din baza de date.

2.2.4 Avantajele MVC/ sa scriu asta la 2.2 MVC

Avantajul major penntru acest framework este că oferă o spearare clară pentru fiecare componentă a aplicației astfel programatorii pot sa lucreze independent pentru diferite aspecte, schimbari ale unei componente nu afeactează in mod direct restul componentelor doar dacă aceste schimbari urmaresc șablonul și regulile arhitecturii MVC. Acest framework oferă posibilitatea de a reutiliza codul, modularitatea acestuia.

Compromisul pentru aceste avantaje este creșterea complexitații programului fiind împărțit in diferite componente.

2.3 Spring Boot

Spring Boot este un framework bazat pe limbajul de programare Java, deschis pentru publicaul larg/”open source”, care simplifică dezvoltarea aplicaților Java în special pentru aplicatile web și miroservicii. Este construit cu ajutorul framework-ului Spring, oferind un mod de a configura și a lansa aplicații Java care pot să ajungă direct în stadiul de producție.

Principalul scop al acestui framework este de a minimiza procesul nesesar de configurare și înființare pentru o aplicație Spring prin adoptarea unui set de configurări logice implicite permițând astfel programatorilor să înceapă dezvoltarea unei aplicații noi fară fara a aloca prea mult timp pe o schemă de amploare.

Unele din caracteristicile cheie ale acestui framework este:

* Auto-Configurarea: Spring Boot poate să configurează mai multe componente în funcție de dependințele din proprietațile aplicației, astfel se elimină nevoia configurării manuale și se reduce probabilitatea apariției unor erori
* Server Încorporat: Spring Boot are în componență un server încorporat cum ar fii Tomcat care oferă posibilitatea aplicaților sa fie pornite fară a fi nevoie de instalarea pe un server extern
* Managementul Dependențelor: Spring Boot simplifică managementul versiunilor dependințelor astfel se asigura o buna compatibilitate între acestea
* Integrarea ecosistemului Spring: Spring Boot integrează alte componente din ecosistemul Sping cum ar fi: Spring Security, Spring Cloud și Spring Data. Toate acestea fiind integrate intr-un ecosistem se pot folosii necesitând un nivel minim de configurare.

2.3.1 Dependințele în Spring Boot

În Spring Boot dependințele sunt genstionate de instrumente de automatizare cum ar fii: Apache Maven sau Gradle. Aceste dependințe se definec în aplicație in fișierul “pom.xml” in Maven sau în “build.gradle” în Gradle. Cele mai folosite dependințe pentru o aplicatie web construită cu Spring Boot sunt:

* Spring-Boot-Starter-Web: această dependtință include librariile necesare pentru o aplicație web, cum ar fi un server încorporat (Tomcat) și librarii pentru gestionarea cererilor și raspunsurilor HTTP
* Spring-Boot-Starter-Data-Jpa: această dependință oferă suport pentru folosirea Java Persistence API cu Spring Data, incluzând librariile necesare pentru legătura cu baze de date relaționale.
* MySql-Connector-Java: este o dependință specifică bazei de date relaționale MySql. Include driver-ul JDBC care oferă posibilitatea aplicației construită cu Spring Boot să se conecteze la baza de date MySql

2.4 Spring Security

Spring Security este un framework folosit pentru aplicațile Java create cu ecosistemul Spring, oferind un mod de a securiza aplicația. Acest framework pune la dispoziția programatorului mai multe caracteristici si opțiuni pentru a manipula autentificarea, autorizarea și alte nevoi legate de securitate pentru aplicație, intr-un mod fexibil și modular. Unele din obiectivele pe care Spring Security le îndeplinește sunt urmatoarele:

* Autentificarea: acest framework pune la dispoziție diferite mecanisme pentru autentificarea utilizatorilor cum ar fi autentificarea bazată pe un formular, autentificarea simpla HTTP sau cu ajutorul token-urilor JWT(Json Web Token)
* Autorizarea: Spring Security oferă mecanisme pentru a controla accesul utilizatorilor pentru diferite părți ale aplicației și poate securiza URL-uri sau anumite metode pentru a fi accesate din exteriorul aplicației. Aceste mecanisme pot să fie bazate pe rolurile utilizatorilor, permisiuni sau reguli predefinite.

## Spring Data

Spring Data este un sub framework din framework-ul Spring care are ca scop simplificarea accesului la diferite nivele de acces de informații pentru aplicațile care au la bază limbajul Java. Spring Data oferă un model de programare unificat și consistent pentru interacțiunile cu mai multe tehnologii pentru stocarea informaților incluzând și bazele de date relaționale.

Caracteristicile cheie pentru Spring Data sunt:

* Abstractizarea, Spring Data prevede abstractizări si interfețe pentru a ascunde complexitatea anumitor tehnologii de acces a informaților. În acest mod se permite ca programatorii să aibă opțiunea de a scrie cod care nu interacționează direct cu codul bazi de date
* Conceptul de “Repository Abstraction”. Acesta prevede interacțiunea cu un API care se ocupă de operațile de baza, creare, citire, actualizare, ștergere pentru interacțiunea cu informațile din baza de date reducând nevoia de a scrie o parte semnificativă de cod
* Posibilitatea de a pagina și de a sorta informația din baza de date fiind simplificată prin anumite abstractizări
* Metode pentru creeare de query-uri. Spring Data oferă posibilitatea de a definii metode cu query-uri prin anumite convenții. Codul SQL sau NoSQL se generează automat bazânduse pe numele metodei, reducânduse nevoia de a scrie query-uri manual

2.4 HTML CSS JavaScript

Separarea acestor limaje

2.5 HTML

Acronimul HTML vine de la “Hypertext Markup Language”, este un limbaj standard folosit pentru creearea de pagini și aplicații web. Structura și conținutul acestuia se reprezintă îm WWW(World Wide Web). Acest limbaj se foloseste de o serie de adnotații pentru a creea și pentru a structura o pagina web.

Codul pentru un document HTML se poate scrie în orice editor de text, singura condiție fiind ca extensia fișierului sa fie “.html”. Orice document HTML contine anotația: “<!DOCTYPE html>” care reprezintă locul de unde incepe codul HTML. Structura unei astfel de pagini se reprezintă prin anotațile „<html> </html>” care incadrează codul prorpiu-zis, „<head> </head>” care conține titlul paginii sau informații despre formatare, „<body> </body>” unde se reprezintă informația din pagină cum ar fi paragrafe cu text sau imagini, uneori din structura unei pagini web face parte și „<footer> </footer>”, reprezentând partea de jos a paginii unde se afla în general informații de contact de exemplu numarul de telefon sau adresa de e-mail.

2.6 CSS

CSS (Cascading Style Sheets) este un limaj folosit împreuna cu limbajul HTML pentru a îmbunatății aspectul vizual pentru o pagina web. Cu ajutorul acestui limbaj programatorul poate să adauge sau să schimbe aspectele vizuale ale paginii după proprile preferințe, cum ar fi: font-urile, mărimea literelor, culorile textului, culorile de fundal sau redimensionarea imaginilor.

Pentru a stiliza un element specific din HTML, CSS pune la dispoziție o serie de selectori pentru componentele HTML. Acești selectori se pot baza pe numle elementelor (h1{}), id-urile acestora (#id{}) sau pe relațile dintre elemente(body > p {}).

Acest limbaj pune la dispozitie si proprietăți prin care elementele se pot poziționa sau alinia intr-o pagină cum ar fi: “display” , “flexbox” sau “grid” . Cu aceste proprietăți se pot obține pagini care își schimbă aspectul în funcție de mărimea ecranului, aceeași aplicație poate să aibă un aspect diferit pe un Desktop față de aspectul de pe un telefon mobil.

2.6.1 Bootstrap

Bootstrap este un framework destinat pentru partea clientului adică pentru interfață. Acesta dispune de o serie de colecții de componente HTML, CSS si JavaSpric predefinite astfel procesul de dezoltare a unei pagini sau aplicații web este mult mai rapid și mai simplu de realizat.

Unul din marile avantaje pentru folosirea acestui framework este că majoritatea componentelor predefinite oferă un design? care este adaptabil pentru diferite dispozitive. Bootstrap oferă si un sistem în care este împărțit ecranul sub forma de table cu 12/doisprezece coloane care se poate personaliza creeând diferite aspecte pentru simplificarea poziționării componentelor într-o pagină.

2.7 JavaScript

JavaScript este un limaj de programare de nivel înalt, orientat pe obiect care funcționează in browserele web. În cele mai multe cazuri este folosit pentru dezvoltarea aplicaților sau paginilor web. Cu acest limbaj se adaugă posibilitatea de a interacționa dinamic cu elementele create cu limajul HTML în pagină. Acest limaj se ocupă în principal cu gestionarea evenimentelor care apar într-o interfață web cum ar fi: click-uri, mișcări ale mouse-ului sau interacțiuni cu tastatura. Gestionarea evenimentelor oferă posibilitatea actualizării paginilor web fară a fi nevoie de reîncărcarea paginii.

Chiar daca la început a fost dezvoltat pentru creearea de cod pentru interfața aplicaților prin intermediul framework-ului Node.js acest limaj se poate utiliza și pentru construirea serverului, astfel acest limaj oferă suport pentru întreaga aplicație web.

Toate browserele din ziua de azi oferă suport și motoare pentru executarea codului JavaScript în mod eficient. În general aplicația trebuie să se muleze pentru nevoile utilizaorilor de aceea find important acest aspect în care se poate folosii orice browser după preferințe, JavaScript fiind cea mai bună alegere pentru dezvoltarea interfețelor aplicaților web.

JavaScript pune la dispoziție mecanisme pentru gestionarea informației primite de la un server pentru o interfață dintr-o pagina web. Fiind un limaj orientat pe obiecte suporta tipul de date obiect dar are si tipuri de date simple cum ar fi: “string” în care se pot stoca litere, cuvinte și numere, “numbers” unde se pot stoca doar numere și “arrays” unde se pot stoca diferite tipuri de date sub formă de șiruri unidimensionale sau bidimensionale. JavaScript poate sa aducă informație în pagina web din surse externe cum ar fi API-urile (Application Programming Interfaces) și astfel se poate afișa pe pagina conținut dinamic prin intermediul actualizărilor în timp real.

S-au dezvoltat mai multe framework-uri pentru o dezvoltare mai eficientă si mai rapidă a interfțelor web, toate acestea având la bază limbajul JavaScript, unele dintre cele mai populare framework-uri care se ocupă cu asa ceva sunt: React, Angular și jQuery.

2.7 React

React este una dintre cele mai populare framework-uri pentru dezvoltarea interfețelor destinate utilizatorilor și a aplicaților web, arhitectura pe care este bazată se folosește de reutilizarea codului și modularitatea acestuia.

Proprietatea de modularitate este atinsă prin divizarea interfețelor în componente independent construite care conțin propria logică de gestionarea a datelor, propriul mod de aranjare și stilizare a elementelor în pagina, chiar și propriul set de teste. Prin această modularizare se pot obține mult mai ușor aplicați complexe în care este nevoie de o mulțime de elemente la care se poate lucra independent și în paralel aducând un plus de eficientizare a procesului si de îmbunatățire a vitezei de lucru.

React introduce o reprezentare virtuală pentru DOM(Document Object Model) care se numește “Virtual DOM”. În loc să se folosească DOM-ul propriu-zis, React construiește o reprezentare virtuală a acestuia în memorie care îmbunătățește performanța programului, tot procesul realizânduse mult mai rapid deoarece în momentul în care sunt schimbări în starea sau în proprietățile unei componente, React actualizează efficient doar părțile necesare din DOM și utlizează la un minim necesar acest proces.

Framework-ul React folosește o sintaxă declarative față de una imperativă.Prin sintaxa declarative programatorul scrie codul astfel încât să descriei aplicației modul în care vrea sa fie afișată informația primită iar React se ocupă de actualizarea interfeței pentru a arata schimbarile în funcție de ce informații primește, astfel procesul de programare a interfeței se simplifică nemaifiind nevoie ca programatorul sa se ocupe manual de fiecare component DOM pentru diferite informații primite ca intrare pentru componentă.

Față de o aplicație web clasică care nu folosește un framework pentru creearea interfeței unde in mod normal pentru a creea componente care se pot afișa pe ecran se folosește HTML, în React se utilizează JSX ( JavaScriptXML ) care este o extensie de sintaxă unde se încorporează cod asemănător cu cel HTML în JavaScript. Extensia JSX este formată dintr-o combinație de structurare a codului HTML cu logica din spatele limajului JavaScript. În acest mod se pot definii toate structurile de componente și proprietățile deja existente în HTML dar intr-un mod intuitiv și mai ușor de recitit pentru alți programatori.

În acest framework se introduce conceptul de “React Hooks”. Această caracteristică pune la dispoziția programatorilor opțiunea de a folosii starea de utilizare a unei componente și alte proprietăți ale framework-ului fără a mai fi nevoie de componente Clasă. Există mai multe tipuri de “Hooks”, dar cele folosite în majoritatea aplicaților sunt “useState”, “useEffect” și “useContext” care simplifică managementul stării componentelor, maipularea evenimentelor secundare și împărțirea contextului între componente. De exemplu pentru a stoca și a seta starea unei componente se folosește useState, aceast “Hook” se declară astfel: const [componentă, setComponentă] = useState(‘’); astfel starea componentei este stocată in constanta numită “componentă” și se poate seta cu funcția “setComponentă”, valoarea de bază a acesteia fiind un string gol.

La fel ca și Spring, React are un ecosistem vast. La acest ecosistem contribuie foarte mult comunitatea care aduce constant componente refolosibile care se pot utiliza foarte ușor. Cele mai utilizate librării din acest ecosistem care ajută la îmbunătățirea productivității sunt: “React Router” care se ocupă cu rutarea pentru URL-urile paginilor din React, prin această librările se poate securiza accesul către pagini cu ajutorul rolurilor de utilizator, “Redux” librărie folosită pentru administrarea stării aplicației și “Axios” pentru cererile HTTP.

2.7.1 Axios

Axios este o librările React folosită pentru a face cereri HTTP dintr-un browser web la un server. Oferă un mod simplu si intuitiv pentru programatori de a apela un API și a de a se ocupa cu operații asincrone în comunicarea cu un server.

Axios se bazează pe un model bazat pe Promises (promisiuni), ceea ce însemnă ca pentru fiecare cerere HTTP facută astfel se returnează un Promise. Un promise este un obiect care reprezintă modul în care o operație asincronă a decurs, în general aducerea informației de la un server, adică poate să rezulte intr-un răspuns de la server sau o eroare în momentul în care ceva nu a decurs bine. Mai departe un Promise pune la dispoziția programatorului un mod în care poate să gestioneze raspunsul primit și să fie salvat intr-o componentă sau un mod în care poate vedea eroarea primită împreună cu alte detalii legate de aceasta pentru a putea fi rezolvată cat mai ușor. Pentru cazul în care se primește răspunsul de la server se utilizează funcția “.then()” iar pentru cazul în care se primește o eroare există funcția “.catch()” . Aceste funcții se folosesc concomitent pentru a se acoperii ambele cazuri la fiecare cerere HTTP.

Pentru a putea folosii Axios este necesară instalarea, care se poate realiza printr-un manager de pachete cum ar fi: npm. După instalare se poate importa în paginile în care este nevoie de cererile HTTP. Acestă librărie are suport pentru toate browserele importante și funcționează după aceleași reguli în fiecare penru o consistență în răspunsuri.

Modul în care se fac cererile HTTP este intuitiv astfel că pentru metodele de bază CRUD (POST, GET, PUT, DELETE) sintaxa axios arată astfel: “axios.post()” , “axios.get()” și asa mai departe. Configurările cererilor se pot face prin opțiunile adiționale: headers, query parameters, request data și timeouts.

2.7.2 Redux

Redux este o librărie JavaScrit pentru managementul stării unei aplicații care se folosește în general cu framework-uri ca și React sau Angular. Cu această librărie se creează un container pentru starea aplicației în care se rețin diferite detalii cum ar fi datele utilizatorului curent pentru fiecare sesiune de pe fiecare dispozitiv unde funcționează aplicația.

Store este singurul obiect în care se găsește starea aplicației care folosește Redux în orice moment, atât timp cat aceasta funcționează.

Actions sunt obiecte JavaScript simple care descriu schimbări pentru starea unei aplicații.

Reducers sunt funcții care se ocupă de actualizarea stării aplicației în funcție de informația primită din Actions.

Dispatch se ocupă de procesul de trimitere a unei Action pentru a actualiza starea aplicației prin apelarea unui Reducer.

Selectors sunt funcții care au rolul de a obține informații specifice din store-ul Redux. Aceste funcții pun la dispoziția programatorului un mod structurat de a accesa informația oferind posibilitatea de a accesa și informațile derivate din cele de bază având opțiunea de a accesa doar anumite părți cheie.

Librăria Redux se bazează pe 3/trei principii fundamentale:

* Redux stochează starea întregii aplicații într-un singur obiect JavaScript numit “store”. Acest obiect store asigură faptul că toate componentele aplicației au acces la aceleași informații în orice moment, astfel se asigură protejare împotriva duplicării informaților și accseul rapid la acestea.
* Starea aplicație în Redux este invariabilă, asta însemnând ca nu poate fi modificată în mod direct. Pentru a se putea interacționa cu stare aplicaței există așa numitele “Actions” care descriu schimbările de stare necesare, după “Reducers” gestionează aceste acțiuni creeând o stare nouă care se bazează pe acțiunile trimis și starea precedentă.
* Reducers sunt doar niște funcții care primesc starea trecută a aplicației și un Action ca și parametrii de funcție, după se returnează o nouă stare a aplicației. Astfel stare aplicației este ușor de testat și este predictibilă deoarece aceste funcții trebuie sa returneze aceeași ieșire pentru aceeași intrare

2.8 MySQL

MySQL este o bază de date relațională deschisă pentru publicul larg?(open-source) fiind foarte folosită pentru a stoca informație structurată. Oferă un sistem robust, scalabil și flexibil pentru o gamă largă de tipuri de aplicații. Limbajul SQL (Structured Query Language) stă la baza acestui sistem de baze de date.

O bază de date relațională înseamnă că informația este organizată sub formă de tabele cu linii si coloane. Stabiliarea relaților între tabele se realizează prin intermediul cheilor oferind integritate informaților stocate.

MySQL are suport pentru stocarea a diferite tipuri de date: numeric, string, dată. Această bază de date dispune de mai multe motoare de stocare cum ar fiȘ MyISAM sau InnoDB fiecare având avantaje și dezavantaje.

În MySQL se pot realiza toate operațile de baza CRUD primite de la un server cum ar fi: inserare, actualizare, citire, ștergere și multe altele, dispunând și de operațile SQL de subinterogări, alăturări, agregări și sortări pentru manipularea datelor într-un mod eficient. Pentru a putea folosii aceste metode este nevoie de unul sau mai multe query-uri. Acestea sunt comenzi dedicate în genral bazelor de date pentru a manipula informația implementând metodele de mai sus. Un query se construiește din una sau mai multe clauze care specifică ce operație trebuie executată și care sunt criteriile pentru obținerea informației respective. Cele mai folosite instrucțiuni sunt: “SELECT” este folosită pentru a obține informașie din una sau mai multe baze de date și se poate specifica numele coloanelor de care este nevoie, “FROM” indentifică tabelul sau tabelele din care informația este extrasă, “WHERE” această clauză este folosită pentru a filtra informația extrasă în funcție de criterile dorite, “INSERT” este folosită pentru a adăuga noi informații într-un tabel specificând numele tabelului și valorile care urmează să fie inserate, “DELETE” este folosită pentru a șterge informație din baza de date specificând condițile pentru a determina ce se dorește pentru ștergere.

Securitatea în această bază de date este oferită de posibilitatea de a creea conturi pentru utilizatori multiplii, fiind nevoie de a te autentifica înainte de a accesa baza de date. Fiecare utilizator având diferite privilegii și nivele de acces, fiind sigur că doar cine are autorizarea potrivită poate sa execute anumite acțiuni. În plus pentru a securiza informația aflată în baza de date MySQL are suport pentru mecanisme de encriptare a informației cand este stocată dar și in momentul în care se află în tranzit. Există posibilitatea de a encripta comunicarea folosind protocoale SSL/TLS. Astfel chiar dacă apare acces unautorizat, informația ramane criptată fara a avea cheia necesară pentru decriptare.

Scalarea bazei de date MySQL funcționează pe verticală, însemnând că prind adaugarea mai multor resurse hardware se poate mării capacitatea sau pe orizontală prin distribuirea informației între servere multiple.

Indexarea îmbunătățește performanța query-urilor deoarece aceștia activează posibilitatea de a primii informațile din tabele mult mai rapid. MySQL oferă suport pentru mai multe tipuri de indexare cum ar fi: hash, graf sau text în funcție de cerințele necesare pentru query-uri.

Conține o analiză a ceea ce s-a realizat/studiat anterior. Arătați că ați studiat materiale bibliografice și că ați înțeles ceea ce ați citit.

Puteți include diferite puncte de vedere asupra problemei pe care o rezolvați în lucrare.

Nu uitați să citați corespunzător autorii oricărei idei extrase dintr-o sursă bibliografică.

# Analiză, proiectare, implementare

## Analiza

Această aplicație este dedicată în cea mai mare parte pacienților, în special pacienții

care locuiesc într-o localitate diferită de cea în care se află medicul de familie. Prin intermediul aplicației se simplifică procesul prin care un pacient poate să intre în posesia unei trimiteri medicale. Cu această aplicație se dorește indeplinirea următoarelor obiective:

* Posibilitatea tuturor utilizatorilor de a creea un cont securizat cu diferite nivele de acces
* Pentru medici și farmaciști aprobarea înregistrării în contul creat doar după ce datele personale sunt confirmate de instituția de care acesta aparține
* Chiar dupa înregistare pacientul poate sa își actualizeze contul cu datele medicului de familie de care aparține, având posibilitatea de schimba aceste date în orice moment dorit
* Medicul de familie are posibilitatea să vizualizeze pacienții de care aparține și poate creea trimiteri medicale pentru aceștia, trimiterile fiind deja completate cu datele personale a pacientului selectat
* Pacientul își poate vizualiza trimiterile creeate cu numele său și le poate atașa într-o programare creeată în prealabil către medicul specialist
* Medicul specialist poate sa vizualizeze trimiterile primite de la pacienți acestea fiind ordonate crescător în ordinea după data și ora programării
* În funcție de trimitere și de investigația pe care o face medicul specialist poate să creeze o prescriere în care specifică medicamentele de care pacientul are nevoie
* Pacientul poate să își vizualizeze prescrierile medicale în cont la secțiunea “Prescrieri”
* Pacientul poate să aleagă una dintre farmaciile prezente, adresa acesteia fiind afișată, în aplicație și să îsi trimită prescrierea
* Farmacistul vizualizează prescrierile primite și poate să pregăteasca medicamentele, după poate să notifice pacientul că meidcamentele sunt pregătite, specifică prețul, ora recomandată de ridicare pentru a nu creea aglomerație și poate să ofere detalii suplimentare pentru modul de administrare
* Pacientul poate sa revină oricând să vizualizeze indicațile primite de la farmacist

Scopul general al acestei aplicații este acela de a facilita tot procesul care pornește de la nevoia de o investigație facută de un medic specialist pentru o anuită problemă de sănătate. Astfel un medic de familie poate sa realizeze o trimitere mult mai ușor fiind salvată în format electronic astfel se ușurează modul în care pacientul poate să își primească trimiterea de la distanță.

Această parte a procesului desfășurându-se în mediul online și trimiterea fiind în format electronic este necesar ca și următoare parte a procesului să se desfăsoare la fel, astfel și prescrierile care vin de la medicul specialist se vor creea în format electronic.

Prin aceată aplicație se digitalizează tot procesul care pornește de la medicul de famile care creează o trimitere și se termină în momentul în care farmacisul notifică pacientul ca medicamentele de care are nevoie sunt pregătite.

Formularele de trimiterii și de prescrieri sunt construite astfel încat să conțină toate câmpurile necesare din formularele de trimitere și prescriere din format fizic. Întregul proces realizat în această aplicație se mulează după procesul de trimiteri și prescrieri deja existent din sistemul medical din România.

## Proiectare

Această aplicație web are la bază arhitectura de proiectare RESTful. La bază arhitectura este orientată pe modul de gestionare a resurselor, fiecare resursă are un identificator unic URI și poate să fie accesat și manipulat prin metodele standard din HTTP. Resursele sunt reprezentate de utilizatori, trimiteri, prescrieri sau de notificările farmaciștilor.

Această athitectură se folosește pentru construirea de servicii web. Se bazează pe principile arhitecturale “Representational State Transfer” (REST) care se folosesc de protocoalele și standardele web existente, resursele sunt identificate de adrese URL unice și clienții interacționează cu aceste resurse folosing standardul HTTP prin metodele de bază CURD (GET, POST, PUT, DELETE).

Astfel interfața, partea pe care o vede orice utilizator care folosește aplicația, comunică cu serverul prin aceste meotode din standardul HTTP. În acest mod se asigură separare dintre client și sever, acestea nu pot să comunice în mod direct ci doar prin intermediul metodelor definite cu acest scop.

Aici pun o schema cu RESTful

Arhitectura folosită penru realizarea părții de server este MVC (Model-View-Controller). După cum îi spune și numele această arhitectură împarte aplicație în trei componente: Model (modele), View (interfețe) și Controller (controlori). Modelele reprezintă datele și logica aplicației, tot aici se implementează și accesul la date.

În această componentă se definește modelul utilizatorilor, pacienți, doctori, farmaciști dar și a trimiterilor, prescrierilor sau a programărilor și administratori împreună cu proprietățile acestora cum ar fi rolurile, numele sau instituțile din care fac parte. Tot în această componentă se creează și servicile pentru fiecare din aceste componente, servicii unde sunt definite metodele prin care se poate interacționa cu proprietățile entităților.

Componeta View în această aplicație reprezintă antetul metodelor create pentru interacționarea cu resursele. Acestea nu conțin deloc implementarea metodelor și pot să fie doar accesate din componenta controller care folosește metodele.

Componenta Controller se ocupă de interacțiunea dintre Model și View. Aceasta primește datele utilizatorului prin intermediul componentei View și actualizează componenta Model în funcție de această intrare.

Merge o poza cu MVC daca nu puneam si la cap 2

Arhitectura parții clientului se bazează în mare parte pe tiparul Flux care este implementat prind Redux. Această arhitectură se bazează pe acțiuni care declanșează schimbări într-o componentă centrală a aplicației numită “store”. Schimbările care apar în interfața utilizatorului se actualizează în funcție de schimbările care apar în această componentă centrală. Aceste schimbări în store constau in cereri HTTP din client către server deoarce întrega aplicație este construită pe baza arhitecturii RESTful.

Merge si aici o schema cu Redux.

## Proiectare

Pentru a se putea pune baza aplicației am realizat următoarele diagrame care au ajutat la conceperea funcționalității: diagrama claselor, diagrama cazurilor de utilizare, diagrama secvențială și diagrama bazei de date unde sunt reprezentate tabele și relațile dintre acestea.

### Diagrama Claselor

### Diagrama Cazurilor de utilizare

### Diagrama Secvențială

### Diagrama bazei de date

## Implementare

Aplicația web care se ocupă de digitalizarea sistemului care se ocupă de trimiterile și prescrierile medicale în format electronic și de folosirea acestora prin intermediul internetului s-a folosit mediul de dezvoltare Intellij IDEA pentru partea serverului și Visual Studio Code pentru partea de client.

Întrega aplicație se bazează pe arhitectura RESTful care separă codul în două mari componente Clientul care este destinat utilizatorului și se ocupă de interfața acestuia și Serverul care implementează logica de lucru cu informațile.

Componenta serverului este implementată cu limbajul de programare Java și prin ajutorul framework-ului Spring Boot care are la bază arhitectura Spring MVC. Această arhitectura împarte codul din server în 3 componente model, view și controller, ficare cu un rol bine definit pe care îl are de îndeplinit.

Baza de date este implementată folosind serverul MySQL iar schimbările făcute în baza de date s-au observat prin programul MySQL Workbench, limbajul folosit pentru baza de date este SQL. Pentru comunicarea cu baza de date în server există o componentă numită Repository.

Componenta care se ocupă cu interfața utilizatorului este implementată cu ajutorul framework-ului React care se bazeză pe o arhitectură Redux. Limbajul de programare folosit este JavaScript, limbajele folosite pentru construirea interfeței sunt HTML și JSX și limbajul pentru stilizarea paginilor este CSS. Această componentă comunică cu serverul prin intermediul protocolului HTTP.

Aici o poza cu toata arhitectura aplicatiei.

### Serverul

Într-o aplicație web construită cu arhitectura RESTful serverul se referă la componenta back-end care se ocupă de cererile clienților, procesarea de date și generarea de răspunsuri pebtru client.

Pentru a putea face ca codul scris în limbajul Java din componenta de back-end s-a folosit Apache Tomcat. Acesta este un server web și un container servlet deschis publicului larg/ open source care este foarte folosit pentru a ține o aplicație bazată pe limbajul Java pentru a putea fi folosită dintr-un server. Servlets sunt clase Java care se ocupă de cererile care vin de la client și generează un raspuns dinamic. Tomcat gestionează ciclul de viață a acestor clase, incluzând inițializarea, tratarea cererilor și alocarea resurselor. Tomcat funcționând ca un server web poate să trateze cereri HTTP în mod direct.

Tomcat oferă un mod direct pentru implementarea și configurarea aplicaților web de care trebuie să se ocupe. Aplicațile pot să fie implementate prin copierea fișierelor WAR (Web Application Archive) în directorul potrivit. Tomcat detectează automat și implementează aceste aplicații. Pentru configurare instrucțiunile se scriu in fișierele server.xml și web.xml, aici se pot oferii detalii despre conectori și securitate. Prin securitatea oferită de Tomcat s-au implementat roluri care permit accesul la diferite metode.

Pentru construirea proiectului în partea de back-end s-a folosit tool-ul Apache Maven. Acest tool este folosit pentru a construii automat codul proiectului și pentru a administra dependințele proiectului. Oferind un model standardizat pentru construirea proiectelor se realizează un mod organizat de a compila codul sursă, executa testele necesare și pentru folosiarea dependințelor. Conceptul pe care se bazează acest tool este Project Object Model (POM), este un fișier XML care este folosit pentru configurarea proiectului. În el se află instrucțiuni pentru Maven care arată modul în care proiectul să fie construit și modul în care să fie administrat mai departe. În pom.xml există informații cu privire la id-ul proiectului, id-ul artefactului, versiunea și tipul pachetului, în acest caz fiin JAR, astfel acești identificatori identifică în mod unic proiectul. În acest fișier se specifică și versiunea framework-urilor folosite, cum ar fi Spring Boot sau Spring Security și conectorul pentru baza de date.

Aici merge o poza cu cv dependency

Componenta serverului se imparte în trei componente principale, Model, View și Controller și în doua componente secundare, Repository care comunică cu baza de date și Security care se ocupă de securitatea metodelor și asigură diferite nivele de securitate în functie de anumite roluri.

Prima componentă cu care s-a început este Model. Aici avem definite toate entitățile care fac parte din aplicația web. În Spring Boot o entitate se referă la o clasă Java care reprezintă obiect cu informații. Pentru a semnaliza pentru Spring Boot care clase sunt entități trebuie să folosim adnotarea @Entity deasupra numelui clasei. Cand Spring Boot gasește această adnotare construiește progarmul astfel încat clasa să fie reprezentată ca un obiect de date persistent, acestea vor fi reprezentate în baza de date sub formă de tabele. Adnotarea “@Entity” face parte din Java Persistance API (JPA), care oferă o mapare obiect relațională (ORM) pentru a reprezenta obiecte Java în tabelele din baza de date relațională. Pentru fiecare din aceste entități urmează adnotarea “@Table”. Prin această adnotare cu atributul “name=” se specifică numele tabelului, în ghilimele, prin care urmează să fie reărezentată entitatea în baza de date astfel “@Table(name = “numele\_entității)”“. Primul atribut din orice entitate este identificatorul unic prin care proprietățile unei entități se inregistrează in baza de date. Acesta are numele de id și este de tipul “Long” pentru a putea stoca cat mai multe înregistrări unice în baza de date, acest tip de date nu suportă numerele cu virgula și este reprezentat pe 8 octeți. Pentru a semnaliza în Spring Boot că acesta este un identifictor unic se folosește adnotația “@Id”, dupa aceasta urmează adnotația care descrie modul în care se generează aceste valori id.

Modul de generare este specificat cu adnotația “@GeneratedValue” care primește un parametru numit “strategy” care specifică tipul de generare. În acest caz s-a ales “@GenerateValue(strategy = GenerationType.IDENTITY)” arată bazei de date să genereze id-ul dupa o coloană identitate care se auto incrementează, cand o noua line cu informații se inserează în baza de date valoarea id-ului nou crește cu o unitate față de inserarea precedentă, prima inserare având valoarea unu, astfel se asigură ca nu se vor repeta valorile id-urilor într-un tabel.

Pentru restul proprietăților dintr-o entitate se specifică cu adnotația “@Column” numele sub care proprietatea respectivă se va înregistra în baza de date, de exemplu pentru “@Column(name = “nume”)” vom aveam o coloana în baza de date care se numește “nume”.

La fiecare proprietate dintr-o entitate se specifică numele, tipul de date (Long, Boolean sau String) și modificatorul de acces “private”. Acest modficator de acces restricționează vizibilitatea și accesibilitatea proprietăților. Modificatorul “private” face ca proprietatea să poată să fie accesată dor din aceeași clasă în care a fost declarată. În acest mod proprietățile unei clase nu se pot accesa în mod direct menținând integritatea datelor. Aceste proprietăți se pot accesa sau modifica doar prin intermediul unor metode speciale care se numesc “Getters” pentru metodele de accesare și “Setters” pentru metodele prin care se pot seta aceste proprietăți.

Pentru a creea automat aceste metode de acces s-a folosit adnotația “@Data” deasupra numelui clasei. Această adnotație face parte din librăria Lombok și generează metode de acces pentru toate prprietățile unei clase pentru a putea fi accesate și in exteriorul claselor în care au fost declarate.

zic de entitatile care eu transient sau enum

Aceasta parte a lucrării este flexibilă și depinde foarte mult de natura lucrării, poate fi organizată în mai multe capitole și conține contribuțiile personale ale autorului.

Includeți:

* + Detalii referitoare la analiză și proiectare:
    - descrierea metodelor pe care le-ați aplicat pentru rezolvarea problemei,
    - descrierea materialelor, procedurilor
    - calcule, tehnici, descrierea echipamentelor
    - metodologia de proiectare
    - informațiile necesare pentru ca cineva să poată reface lucrarea
  + Implementare :
    - Descrieți detaliile tehnice ale implementării aplicației: mediul de implementare, modul de prezentare, modul de utilizare al aplicației, etc.
  + Testare si validare :
    - Descrieți metodologia de testare a aplicației și rezultatele
    - Includeți experimentele pe care le-ați realizat, analiza rezultatelor pe care le-ați obținut.

# Concluzii

## Rezultate obținute

Evidențiați toate rezultatele pe care le-ați obținut și trageți concluzii din ele. Puteți prezenta o analiză critică a ceea ce ați realizat comparativ cu alte lucrări/studii anterioare.

Includeți o listă a contribuțiilor pe care le-ați avut în domeniul temei abordate.

## Direcții de dezvoltare

Descrieți direcțiile posibile de dezvoltare.

# Reguli de formatare

## Formatarea paginii

* + Dimensiunea paginii: A4
  + Margini: 2.5 cm (sus, jos, stânga, dreapta)
  + Antet și subsol: 1.27 cm de la marginea paginii
  + În antetul paginii (header): titlul capitolului, centrat, stil: Header\_style
  + În subsolul paginii: numărul paginii, centrat

## Titluri și stiluri

Titlurile capitolelor și subcapitolelor se marchează cu stilurile Heading 1 – 4, conform documentului model anexat în format Word. Descrierea stilurilor utilizate în document este prezentată în Tabelul 5.1.

Tabelul 5.1. Stiluri utilizate în acest document

| Nr. | Stil | Utilizat pentru | Format |
| --- | --- | --- | --- |
| 1 | Normal | Text normal | Font: (Default) Cambria, 12 pt, Justified, Line spacing: Multiple 1.1 li, Space After: 6 pt |
| 2 | Titlu | Titlul proiectului, prima pagină | Font: 24 pt, Small caps, Centered Line spacing: single, Space Before: 126pt, After: 0 pt, |
| 3 | Titlu2 | Titlul proiectului, pagina de prezentare | Font:14pt, Bold, Centered |
| 4 | Heading 1 | Titlurile capitolelor (nivel 1) | Font: 24 pt, Indent: Left: 0 cm Hanging: 0.76 cm, Space Before: 24pt, After: 12pt |
| 5 | Heading 2 | Titlurile subcapitolelor (nivel 2) | Font: 14 pt, Bold, Indent: Left: 0 cm  Hanging: 1.02 cm, Space Before: 18pt, After: 12pt |
| 6 | Heading 3 | Titlurile secțiunilor (nivel 3) | Font: Bold, Indent: Left: 0 cm Hanging: 1.27 cm, Space Before: 6 pt, After: 6pt |
| 7 | Heading 4 | Titlurile secțiunilor (nivel 4) | Font: Italic, Indent: Left: 0 cm Hanging: 1.52 cm, Space Before: 2 pt, After: 0 pt |
| 8 | Caption | Legenda figurilor și tabelelor | Font: Italic, Font color: Text 1, Line spacing: single, Space After: 10 pt, |
| 9 | Header\_style | Antetul paginii | Font: 10 pt, Italic, Centered, Border: Bottom: (Single solid line, Background 1, 0.5 pt Line width) |

## Figuri, tabele și ecuații

### Figuri

Figurile se inserează în text centrate, cu etichetă de numerotare și legendă (Caption) în partea de jos a figurii. Numărul figurii include și numărul capitolului, după exemplul prezentat în Figura 5.1.



Figura 5.1. Figură exemplu, stil: Caption

## Tabele

Tabelele se inserează în text centrate, cu etichetă și legendă (Caption) în partea de sus a tabelului, aliniată la stânga. Numărul tabelului include și numărul capitolului, după cum este prezentat, de exemplu, în Tabelul 5.1.

## Ecuații

Ecuațiile se inserează în text centrate, cu numerotare în partea dreaptă. Numărul ecuației include și numărul capitolului, conform exemplului din relația (5.1).

|  |  |
| --- | --- |
|  | (5.1) |

## Referințe bibliografice

Se recomandă ca citarea referințelor bibliografice să fie făcută în formatul IEEE.

În secțiunea Bibliografie sunt prezentate exemple pentru: o citare a unui capitol dintr-o carte [1], un articol publicat într-o revistă [2] și un articol publicat la o conferință [3].

Detalii cu privire la formatul citării diverselor tipuri de referințe pot fi găsite în [4] sau [5].

Referințele bibliografice se pot insera în text utilizând facilitățile Word de a adăuga surse și bibliografie unui document (References -> Citations & Bibliography). Dacă formatul IEEE pentru bibliografie nu este instalat implicit în Word, se poate descărca gratuit de la:

<https://bibword.codeplex.com/wikipage?title=Styles&referringTitle=Home>

Instrucțiunile de instalare pentru diferite versiuni de Word se pot obține de la aceeași adresă.

# Bibliografie

|  |  |
| --- | --- |
| [1] | P. Nume, "Titlul capitolului," în *Titlul cartii*, Oras, Editura, 2016, pp. 1-24. |
| [2] | P. Nume, "Titlul articolului," *Titlul revistei,* vol. 1, no. 2, pp. 22-30, 2016. |
| [3] | P. Nume, "Titlul articolului," în *Numele conferintei*, Oras, 2015. |
| [4] | "IEEE Citation Reference," 2009. [Online]. Available: https://www.ieee.org/documents/ieeecitationref.pdf. |
| [5] | "IEEE Editorial Style Manual," 2016. [Online]. Available: https://www.ieee.org/documents/style\_manual.pdf. |