Proiectarea si implementarea unui sistem inteligent de management al locurilor de parcare

# Documentul de proiectare

Cuprins

[1. Introducere 1](#_Toc160527836)

[1.1 Scopul documentului 1](#_Toc160527837)

[2. Prezentare generală și abordări de proiectare 2](#_Toc160527838)

[2.1 Prezentare generală 2](#_Toc160527839)

[2.2 Presupuneri/ Constrângeri/ Riscuri 2](#_Toc160527840)

[2.2.1 Presupuneri 2](#_Toc160527841)

[2.2.2 Constrângeri 2](#_Toc160527842)

[2.2.3 Riscuri 3](#_Toc160527843)

[3. Considerațiii de proiectare 4](#_Toc160527844)

[3.1 Obiective și linii directoare (ghiduri) 4](#_Toc160527845)

[3.2 Metode de dezvoltare 4](#_Toc160527846)

[3.3 Strategii de arhitectură 4](#_Toc160527847)

[4. Arhitectura Sistemului și Proiectarea Arhitecturii 6](#_Toc160527848)

[4.1 Vedere logică 6](#_Toc160527849)

[4.2 Arhitectură hardware 6](#_Toc160527850)

[4.3 Arhitectură software 6](#_Toc160527851)

[4.4 Arhitectura informațiilor 7](#_Toc160527852)

[4.5 Arhitectura de comunicații interne 7](#_Toc160527853)

[4.6 Diagrama de arhitectură a sistemului 8](#_Toc160527854)

[5. Proiectarea sistemului 9](#_Toc160527855)

[5.1 Proiectarea bazei de date 9](#_Toc160527856)

[5.1.1 Obiecte de date și structuri de date rezultante 9](#_Toc160527857)

[5.1.2 Fișiere și baze de date 9](#_Toc160527858)

[5.2 Conversii de date 9](#_Toc160527859)

[5.3 Interfețe utilizator 10](#_Toc160527860)

[5.3.1 Intrări 10](#_Toc160527861)

[5.3.2 Ieșiri 10](#_Toc160527862)

[5.4 Proiectarea interfețelor cu utilizatorul 10](#_Toc160527863)

[6. Scenarii de utilizare 11](#_Toc160527864)

[7. Proiectare de detaliu 12](#_Toc160527865)

[7.1 Proiectare hardware de detaliu 12](#_Toc160527866)

[7.2 Proiectare software de deatliu 12](#_Toc160527867)

[7.3 Proiectare detaliată de securitate 13](#_Toc160527868)

[7.4 Proiectare de detaliu pentru performanța sistemului 14](#_Toc160527869)

[7.5 Proiectare detaliată a comunicațiilor interne (între componente) 14](#_Toc160527870)

[8. Controale pentru verificarea integrității sistemului 15](#_Toc160527871)

[Anexa A: Gestiunea modificărilor documentului 16](#_Toc160527872)

[Anexa B: Acronime 17](#_Toc160527873)

[Anexa C Documente la care se face referire 18](#_Toc160527874)

## Introducere

Informații de identificare pentru sistemul existent/proiectat:

* Numele Proiectului: Proiectarea si implementarea unui sistem inteligent de management al locurilor de parcare
* Acronim: SIMLP
* Sistem Existenta/Propus: Sistemul poate fi implementat fie pentru o parcare deja existentă, fie pentru o parcare nouă. În ambele cazuri, scopul sistemului este de a îmbunătăți gestionarea și monitorizarea parcării.

### Scopul documentului

Prin acest livrabil se documentează și urmăresc informațiile necesare pentru a defini eficient arhitectura și designul sistemului, în scopul de a oferi echipei de dezvoltare îndrumare asupra arhitecturii sistemului ce urmează să fie dezvoltat.

Documentele de proiectare sunt produse incremental și iterativ pe parcursul ciclului de viață al dezvoltării sistemului, în funcție de circumstanțele particulare ale proiectului de tehnologie informațională (IT) și de metodologia de dezvoltare a sistemului utilizată.

Publicul țintă este managerul de proiect, echipa de proiect și echipa de dezvoltare. Anumite părți ale acestui document, cum ar fi interfața cu utilizatorul (UI), pot fi împărtășite cu clientul/utilizatorul și cu alte părți interesate ale căror contribuții/aprobări sunt necesare în UI.

## Prezentare generală și abordări de proiectare

Această secțiune descrie principiile și strategiile care vor fi utilizate ca ghiduri în momentul proiectării și implementării sistemului.

### Prezentare generală

Proiectul vizează dezvoltarea unui sistem inteligent de management al locurilor de parcare pentru a îmbunătăți eficiența, siguranța și experiența utilizatorilor în parcurile de mașini. Scopul nostru este să oferim o soluție inovatoare care să permită gestionarea și monitorizarea parcărilor în timp real, folosind tehnologii moderne

Abordarea de proiectare se concentrează pe dezvoltarea unei arhitecturi flexibile, scalabile și robuste, care să permită integrarea ușoară a diferitelor componente și tehnologii.

**Arhitectura Sistemului:**

* Sistemul va fi compus dintr-o serie de componente hardware și software interconectate, inclusiv senzori, camere, dispozitive IoT, server și interfețe de utilizator.

**Arhitectura Software-ulu**

Frontend-ul Web:

* Interfața de administrator va fi implementată ca o aplicație web, care va permite vizualizarea și gestionarea locurilor de parcare.
* Interfața va fi construită folosind tehnologii moderne de frontend, precum HTML, CSS și JavaScript, React, Typescript, și va comunica cu backend-ul prin intermediul unui REST API.

Frontend-ul Mobile (Flutter):

* Interfața de utilizator (UI) va fi implementată folosind framework-ul Flutter pentru a crea aplicații mobile native pentru platformele iOS și Android.
* Flutter oferă un mod eficient de a construi UI-uri frumoase și responsive, utilizând un singur cod sursă pentru ambele platforme.

Backend-ul:

* Backend-ul va gestiona logica aplicației și va comunica cu diferitele componente ale sistemului.
* Va fi dezvoltat folosind un cadru de dezvoltare web, in Golang, și va oferi servicii precum autentificarea utilizatorilor, gestionarea rezervărilor și monitorizarea stării parcării.

Bază de Date:

* Datele vor fi stocate într-o bază de date relațională PostgreSQL, pentru a asigura integritatea și consistența datelor.
* Containerul Docker va include și un serviciu de baza de date pentru a gestiona și a oferi acces la datele aplicației.

### Presupuneri/ Constrângeri/ Riscuri

#### Presupuneri

Software și Hardware asociat:

* Se presupune că sistemul va fi integrat cu hardware-ul necesar, cum ar fi camerele de supraveghere, senzorii IR și terminalele NFC pentru funcționalitățile de monitorizare și acces.
* Dependența de software poate include utilizarea anumitor biblioteci sau framework-uri pentru dezvoltarea aplicațiilor frontend și backend, precum și a serviciilor cloud pentru stocarea și gestionarea datelor.

Caracteristicile Utilizatorilor Finali:

* Se presupune că utilizatorii finali ai sistemului sunt șoferi și administratori de parcări, care au cunoștințe de bază în utilizarea aplicațiilor mobile și a interfeței web.
* Interfața utilizator va fi proiectată pentru a fi intuitivă și ușor de folosit, luând în considerare nevoile și preferințele utilizatorilor finali

Posibile Modificări ale Funcționalității:

* Este posibil ca funcționalitățile și caracteristicile sistemului să fie modificate sau extinse în viitor în funcție de feedback-ul utilizatorilor și de evoluția tehnologică.
* Sistemul trebuie să fie proiectat cu flexibilitate pentru a permite adăugarea ușoară a noi caracteristici și funcționalități prin actualizări software.

#### Constrângeri

Capacitatea și Performanța Hardware:

* Limitările hardware, cum ar fi puterea de procesare, capacitatea de stocare și memoria disponibilă, pot influența designul software-ului și funcționalitățile sistemului.
* capacitate hardware limitată poate duce la necesitatea optimizării codului și la restricționarea funcționalităților complexe care consumă multe resurse.

Cerințe de Interoperabilitate și Compatibilitate:

* Dacă sistemul trebuie să interopereze cu alte sisteme sau dispozitive externe, trebuie luate în considerare cerințele de interoperabilitate și compatibilitate.
* Aceste cerințe pot influența protocoalele de comunicație și interfețele utilizate în cadrul sistemului.

Comunicări de Rețea:

* Limitările rețelei, cum ar fi latimea de bandă, stabilitatea și timpul de răspuns, pot afecta comunicațiile între diferitele componente ale sistemului.
* Este important să se ia în considerare aceste limitări în proiectarea protocoalelor de comunicație și a mecanismelor de gestionare a erorilor.

#### Riscuri

Vulnerabilități de Securitate:

* Există riscul ca sistemul să fie expus la vulnerabilități de securitate, cum ar fi atacurile cibernetice sau accesul neautorizat la datele utilizatorilor.
* Strategii de reducere: Implementarea unor măsuri de securitate robuste, cum ar fi criptarea datelor, autentificarea puternică a utilizatorilor și actualizările regulate de securitate. Realizarea auditurilor de securitate și testarea penetrării pentru identificarea și remedierea vulnerabilităților.

Probleme de Performanță și Scalabilitate:

* Există riscul ca sistemul să nu fie capabil să gestioneze sarcini și solicitări mari, ceea ce poate duce la scăderi de performanță sau căderi ale sistemului.
* Strategii de reducere: Testarea și optimizarea continuă a performanței sistemului, inclusiv monitorizarea utilizării resurselor și identificarea și remedierea punctelor slabe. Implementarea unor strategii de scalabilitate, cum ar fi distribuirea în cluster și utilizarea serviciilor cloud scalabile.

Dependență de Tehnologii Externe:

* Există riscul ca sistemul să depindă de tehnologii externe sau terțiare, cum ar fi serviciile cloud sau bibliotecile de cod, care pot fi supuse unor erori sau întârzieri.
* Strategii de reducere: Identificarea și evaluarea riscurilor asociate cu dependența de tehnologii externe. Implementarea unor planuri de rezervă și de recuperare în cazul în care apare o problemă cu aceste tehnologii. Utilizarea serviciilor cloud și a bibliotecilor de cod cu un istoric solid și cu suport activ pentru a minimiza riscul.

## Considerațiii de proiectare

Este important să se evalueze resursele disponibile, inclusiv bugetul, timpul și expertiza echipei. Acest lucru va influența deciziile legate de arhitectură, tehnologii și scalabilitatea proiectului.

Este important să se stabilească și să se respecte standardele și reglementările relevante, cum ar fi standardele de securitate, reglementările GDPR și altele. Acest lucru poate avea un impact semnificativ asupra proiectării și implementării sistemului.

Este util să se evalueze tehnologiile și soluțiile existente disponibile pentru a determina cea mai potrivită pentru proiect. Acest lucru poate include cercetarea pieței, testarea prototipurilor și evaluarea costurilor și beneficiilor.

### Obiective și linii directoare (ghiduri)

Eficiența și Performanța:

* Accentul se pune pe dezvoltarea unei aplicații eficiente din punct de vedere al performanței, care să ofere o experiență rapidă și receptivă utilizatorilor. Acest lucru include optimizarea algoritmilor, gestionarea eficientă a resurselor și minimizarea timpilor de încărcare.

Simplicitate și Ușurință de Utilizare:

* Interfața utilizator va fi proiectată pentru a fi simplă, intuitivă și ușor de folosit, chiar și pentru utilizatorii neexperimentați. Se urmărește eliminarea oricăror elemente sau funcționalități care ar putea confunda sau încărca experiența utilizatorului.

Scalabilitate și Flexibilitate:

* Sistemul va fi proiectat pentru a fi scalabil și flexibil, pentru a putea gestiona creșterea volumului de utilizatori și de date în viitor. Se va acorda atenție utilizării tehnologiilor și a arhitecturilor care permit extensibilitatea și adaptabilitatea la schimbări.

### Metode de dezvoltare

Pentru designul sistemului și al software-ului, vom îmbina abordarea orientată pe obiecte (OOP) cu paradigma programării funcționale (FP). Această combinație ne va permite să obținem beneficiile ambelor paradigme și să abordăm problemele într-un mod flexibil și eficient.

Orientarea pe Obiecte (OOP):

* Se utilizeaza conceptele OOP pentru a organiza și structura componentele sistemului în jurul obiectelor și a interacțiunilor dintre acestea.
* Principiile OOP, cum ar fi încapsularea, moștenirea și polimorfismul, ne vor ajuta să creăm un design modular și extensibil.

Programarea Funcțională (FP):

* Se foloseste paradigma FP pentru a trata operațiile ca evaluarea funcțiilor matematice pure si crearea de componente functionale
* Funcțiile pure, imutabilitatea și tratarea datelor ca argumente de intrare și ieșire vor ajuta la scrierea codului mai clar, mai concis și mai predictibil.

### Strategii de arhitectură

Pentru sistemul de management al locurilor de parcare, următoarele decizii și strategii de design afectează organizarea generală a sistemului și structurile sale de nivel superior:

* Alegerea PostgreSQL pentru baza noastră de date ne oferă o bază de date relațională puternică și fiabilă. Cu suport pentru tranzacții și un set bogat de funcționalități, PostgreSQL este o opțiune excelentă pentru stocarea datelor despre locuri de parcare, rezervări și utilizatori.
* Utilizarea Docker permite containerizarea aplicației, inclusiv atât frontend-ul React, cât și backend-ul PostgreSQL și API-ul serverului. Aceasta facilitează distribuția aplicației noastre pe mai multe sisteme de operare și infrastructuri, asigurând portabilitate și consistență între medii de dezvoltare, testare și producție.
* React Query ne oferă un mod simplu și eficient de a gestiona starea aplicației React și de a face cereri către server. Folosind React Query, putem gestiona automat caching-ul datelor, invalidarea cache-ului și refetching-ul datelor, eliminând astfel nevoia de a scrie cod suplimentar pentru gestionarea stării

Planuri de dezvoltare ulterioara

**Implementarea Funcționalităților Avansate de Rezervare:**

Adaugarea mai multor opțiuni și funcționalități avansate pentru procesul de rezervare, cum ar fi rezervări recurente, notificări personalizate pentru rezervări și opțiuni de plată suplimentare.

**Extinderea Serviciilor de Raportare și Analiză:**

Extinderea serviciilor de raportare și analiză, oferind utilizatorilor și administratorilor instrumente mai puternice pentru analiza datelor, generarea de rapoarte personalizate și identificarea tendințelor in utilizarea parcarii.

## Arhitectura Sistemului și Proiectarea Arhitecturii

**Interfața Utilizatorului (Frontend):**

Componenta responsabilă de interacțiunea directă cu utilizatorii. Este dezvoltată folosind tehnologii precum React și interacționează cu backend-ul pentru a prelua și a afișa datele utilizatorilor, pentru a gestiona rezervările și pentru a oferi funcționalități de căutare și filtrare a locurilor de parcare

**Backend-ul Aplicației (Backend/API):**

Componenta care gestionează logica de afaceri a aplicației și comunică cu baza de date și alte servicii externe. Este dezvoltată folosind Golang și oferă un API REST pentru a permite interacțiunea cu frontend-ul și alte aplicații externe.

**Baza de Date PostgreSQL:**

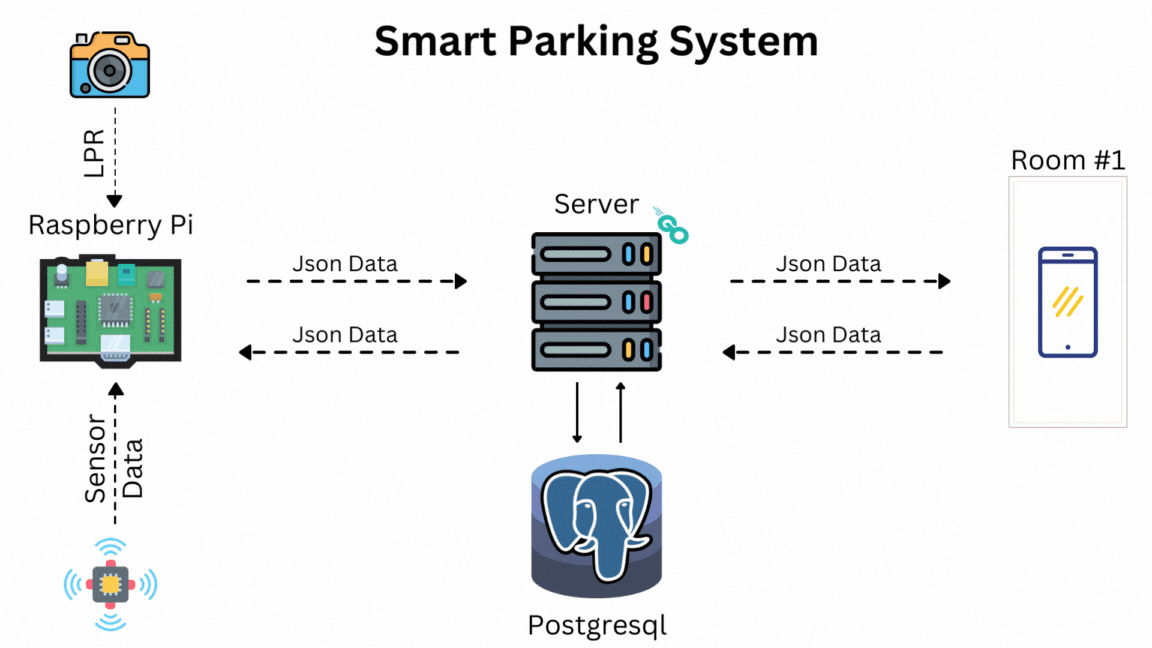
Componenta responsabilă de stocarea datelor despre locurile de parcare, rezervări și utilizatori. Este o bază de date relațională care este accesată și manipulată de backend pentru a efectua operațiile CRUD (Create, Read, Update, Delete) asupra datelor.

**Serviciile Externe:**

Sistemul poate interacționa cu diverse servicii externe, cum ar fi servicii de plată pentru procesarea plăților, servicii de navigație pentru a afișa locațiile parcajelor pe hărți și servicii de notificare pentru a trimite alerte și notificări către utilizatori.

Arhitectura este proiectată pentru a fi modulară și ușor de extins și de întreținut. Componentele sunt dezvoltate separat, permițând echipei să lucreze independent asupra anumitor părți ale sistemului fără a afecta restul aplicației. De asemenea, arhitectura este scalabilă, permițând adăugarea de noi funcționalități și integrarea cu noi servicii externe în viitor.

### Vedere logică



### Arhitectură hardware

Pentru sistemul nostru de management al locurilor de parcare, avem o arhitectură centralizată, cu un Raspberry Pi și senzori ca parte a infrastructurii hardware. Iată o descriere generală a hardware-ului și organizației sistemului:

**Raspberry Pi:**

Un singur Raspberry Pi este utilizat ca server central pentru a gestiona și monitoriza locurile de parcare. Raspberry Pi servește ca nod central pentru colectarea datelor de la senzori, procesarea acestora și furnizarea unei interfețe de comunicare cu alte componente ale sistemului.

**Senzori:**

Senzorii sunt amplasați la fiecare loc de parcare pentru a detecta dacă locul respectiv este ocupat sau liber. Acești senzori pot fi senzori infraroșu (IR) sau alți senzori de proximitate, care trimit semnale către Raspberry Pi pentru a indica starea de ocupare a locurilor de parcare.

Organizația generală a sistemului include Raspberry Pi-ul plasat central și senzorii distribuiți la fiecare loc de parcare. Senzorii transmit date către Raspberry Pi pentru a fi procesate și utilizate în aplicația de management al locurilor de parcare.

**Camera pentru Recunoașterea Automată a Numerelor de Înmatriculare (LPR):**

Camera LPR este amplasată strategic în zona de intrare/ieșire a parcării pentru a detecta și a înregistra numerelor de înmatriculare ale vehiculelor care intră și ies din parcare.

Datele captate de camera LPR sunt trimise către un api extern pentru a fi procesate și interpretate. Acest lucru poate fi realizat folosind un software specializat pentru LPR sau un model de învățare automată care recunoaște și interpretează numerelor de înmatriculare din imaginile captate.

**Tehnologia NFC (Near Field Communication):**

Dispozitive NFC sunt utilizate pentru accesul în parcare.

Atunci când dispozitivul NFC este apropiat de un cititor NFC amplasat în apropierea barierei de acces, informațiile de identificare sunt citite și procesate pentru a permite sau a refuza accesul în parcarea respectivă.

### Arhitectură software

Visual Studio Code (VS Code) este un editor de cod sursă dezvoltat de Microsoft,cunoscut pentru interfața sa simplă și performanța excelentă. Acesta vine cu suport integrat pentru numeroase limbaje de programare și oferă funcții precum evidențierea sintaxei ,completarea automată a codului și integrarea cu Git. Cu instrumente avansate pentru debbuging și o comunitate activă de dezvoltatori, VS Code este alegerea preferată pentru mulți programatori.

HTML, sau Hypertext Markup Language, este limbajul utilizat pentru structurarea și

crearea paginilor web. Folosind etichete precum <p> pentru paragrafe, <img> pentru imagini și

<a> pentru link-uri, HTML permite adăugarea de text, imagini și link-uri în paginile web. De

asemenea, oferă suport pentru crearea de liste, formulare pentru colectarea de date și elemente semantice pentru descrierea conținutului paginii. HTML este un standard recunoscut global și este compatibil cu majoritatea browserelor web moderne.

React este o bibliotecă JavaScript open-source pentru construirea interfețelor utilizator

(UI). Este dezvoltată și menținută de către Facebook, fiind utilizată pe scară largă pentru

dezvoltarea aplicațiilor web interactive și dinamice. Principalele caracteristici ale React includ:

Componente reutilizabile: React se bazează pe conceptul de componente, permițând

dezvoltatorilor să împartă interfața utilizator în componente reutilizabile și independente. Aceste

componente pot fi compuse și combinate pentru a crea interfețe complexe și dinamice.

Virtual DOM (Document Object Model): React utilizează un model virtual al DOM-ului

pentru a optimiza performanța și pentru a actualiza eficient interfața utilizator în funcție de

schimbările din aplicație. Comparând virtual DOM-ul cu DOM-ul real, React determină

modificările necesare și le aplică în mod eficient, fără a reafecta întreaga pagină.

JSX (JavaScript XML): React folosește JSX, o sintaxă similară cu HTML-ul, pentru a

descrie structura componentelor și interfața utilizator. JSX permite încorporarea de cod

JavaScript direct în markup-ul UI, facilitând dezvoltarea și înțelegerea componentelor.

Unidirectional Data Flow: React promovează un flux unidirecțional al datelor, ceea ce

înseamnă că datele sunt transmise de sus în jos prin ierarhia componentelor. Această abordare

simplifică gestionarea stării aplicației și face mai ușoară de înțeles relația între componentele UI. Comunitate activă și ecosistem bogat de biblioteci și instrumente: React beneficiază de o

comunitate vastă de dezvoltatori și contribuitori, iar ecosistemul său include o varietate de

biblioteci și instrumente care extind și îmbunătățesc funcționalitățile sale.

React este adesea utilizat împreună cu alte tehnologii și framework-uri, cum ar fi Redux

pentru gestionarea stării aplicației, React Router pentru navigare și Next.js pentru renderizare la

server și generarea paginilor statice. Această flexibilitate și puterea React-ului îl fac o alegere

populară pentru dezvoltarea aplicațiilor web moderne și scalabile.

React Query este o bibliotecă populară pentru gestionarea stării și a datelor în aplicațiile

React. Aceasta facilitează interacțiunea cu datele, inclusiv preluarea, actualizarea și gestionarea stării de încărcare a datelor în mod eficient. Iată câteva aspecte cheie despre React Query:

Încărcare datelor: React Query oferă un set de hook-uri simplificate pentru a prelua

datele din diverse surse, cum ar fi API-uri RESTful, GraphQL sau chiar din cache-ul local.

Aceste hook-uri gestionează automat starea de încărcare și starea de eroare asociate cererilor de

date.

Caching și invalidare: Biblioteca gestionează automat caching-ul datelor, astfel încât să

nu fie necesară o cerere suplimentară către server pentru datele care au fost deja preluate. De

asemenea, oferă metode pentru a invalida manual datele din cache când este necesar.

Reîmprospătare și refetching: React Query permite reîmprospătarea datelor în mod

automat sau manual, în funcție de nevoile aplicației. Acest lucru poate fi util pentru a asigura că

datele sunt mereu actualizate și consistente cu serverul.

Updatarea datelor: React Query oferă suport pentru efectuarea mutațiilor asupra datelor,

inclusiv adăugarea, actualizarea și ștergerea datelor. Aceste mutații pot fi gestionate în mod

optim prin intermediul hook-urilor specifice.

Gestionarea stării globale: React Query poate fi integrat cu ușurință în diverse soluții de

gestionare a stării globale, cum ar fi Context API sau biblioteci precum Redux, pentru a gestiona

datele comune în întreaga aplicație.

Optimizare performanță: Biblioteca este optimizată pentru performanță, reducând la

minim cererile suplimentare către server și gestionând eficient caching-ul datelor pentru a

asigura o experiență rapidă și fluidă pentru utilizatori.

Cascading Style Sheets (CSS) este un limbaj de stilizare utilizat pentru a controla

aspectul și prezentarea paginilor web create în HTML. CSS permite dezvoltatorilor să definească

formatarea și stilizarea elementelor HTML, inclusiv aspectul textului, culorile de fundal,

dimensiunile, marginile și alinierea elementelor, pentru a crea pagini web estetice și plăcute din

punct de vedere vizual.

Prin intermediul selecționării și aplicării de reguli de stil folosind selectori și proprietăți

CSS, dezvoltatorii pot să personalizeze aspectul și comportamentul fiecărui element individual

sau a grupurilor de elemente dintr-o pagină web. Acest lucru permite crearea unui design coerent

și atractiv pentru întreaga pagină web sau întregul site.

12

De asemenea, CSS oferă suport pentru concepte avansate precum layout-ul responsiv,

care permite adaptarea paginilor web la diferite dimensiuni de ecran și dispozitive, și animații și

tranziții, care adaugă interactivitate și dinamism paginilor web.

Prin separarea structurii și conținutului HTML de stilurile CSS, dezvoltatorii pot menține

un cod mai curat și mai ușor de întreținut, permițând modificări rapide și consistente ale

aspectului și prezentării paginilor web. CSS este un instrument esențial în crearea de experiențe

web moderne și atractive pentru utilizatori.

SCSS (Sassy CSS) și Tailwind CSS sunt două tehnologii utilizate pentru stilizarea

paginilor web în cadrul proiectelor de dezvoltare web.

SCSS este o extensie a limbajului CSS (Cascading Style Sheets) care adaugă

funcționalități suplimentare precum variabile, nesting, mixin-uri și funcții. Prin utilizarea SCSS,

dezvoltatorii pot organiza și gestiona mai eficient stilurile CSS ale aplicațiilor lor web, reducând

duplicarea și creșterea productivității. Principalele caracteristici ale SCSS includ:

Variabile: Permite definirea și utilizarea de variabile pentru culori, dimensiuni, fonturi

etc., facilitând actualizarea și consistența stilurilor.

Nesting: Permite înglobarea stilurilor CSS în cadrul altor selecții, facilitând organizarea

și structurarea codului.

Mixin-uri: Permite definirea și reutilizarea de blocuri de stiluri care pot fi apoi aplicate la

diferite elemente, reducând duplicarea și creșterea flexibilității.

Tailwind CSS este un framework CSS util pentru dezvoltarea rapidă a interfeței

utilizatorului. Acesta oferă un set extins de clase predefinite care permit stilizarea rapidă a

elementelor HTML, fără a fi nevoie de scrierea manuală a stilurilor CSS. Principalele

caracteristici ale Tailwind CSS includ:

Utility-first approach: Tailwind CSS adoptă o abordare de "utilitate întâi", oferind clase

predefinite pentru stiluri comune, cum ar fi margini, padding, culori, tipografie etc. Această

abordare permite construirea rapidă și flexibilă a interfeței utilizator fără a fi nevoie de stiluri

personalizate.

Configurabilitate: Tailwind CSS permite configurarea și personalizarea stilurilor prin

intermediul fișierului de configurare, permițând adaptarea framework-ului la nevoile și

preferințele proiectului.

Extensibilitate: Tailwind CSS poate fi extins prin adăugarea de plugin-uri sau prin

definirea de clase personalizate, permițând dezvoltatorilor să integreze stiluri specifice

proiectului fără a renunța la beneficiile oferite de framework.

TypeScript este un limbaj de programare open-source dezvoltat și întreținut de către

Microsoft. Este o supersetă a limbajului JavaScript, ceea ce înseamnă că este construit pe baza

JavaScript-ului și adaugă funcționalități suplimentare precum tipizarea statică și funcționalități

orientate pe obiecte. Principalele caracteristici ale TypeScript includ:

13

Tipizare statică: TypeScript permite specificarea tipurilor de date pentru variabile,

argumente de funcție și valori returnate. Aceasta oferă mai multă claritate și siguranță la scrierea

codului, identificând erorile de tip în timpul dezvoltării și facilitând refactorizarea și întreținerea

codului.

Orientat către obiecte: TypeScript oferă suport pentru caracteristici orientate pe obiecte

precum clase, interfețe, moștenire și module. Aceste caracteristici permit dezvoltatorilor să

creeze și să gestioneze structuri complexe de cod într-un mod organizat și ușor de înțeles.

ECMAScript compatibilitate: TypeScript este compatibil cu standardele ECMAScript,

permițând utilizarea funcționalităților JavaScript moderne și beneficiind de avantajele aduse de

noile actualizări ale limbajului.

Tooling și integrare cu IDE-uri: TypeScript este integrat cu diverse medii de dezvoltare

(IDE-uri) precum Visual Studio Code, Sublime Text și altele, oferind funcționalități avansate de

completare automată, verificare a tipurilor și navigare în cod.

Generare de cod JavaScript: TypeScript este transpilat în cod JavaScript standard înainte

de a fi executat în browser sau în medii de execuție JavaScript. Acest lucru înseamnă că

dezvoltatorii pot utiliza funcționalitățile avansate ale TypeScript în timpul dezvoltării și pot

genera cod JavaScript compatibil pentru a fi livrat în producție.

TypeScript este utilizat pe scară largă în dezvoltarea aplicațiilor web și a altor tipuri de

aplicații, fiind recunoscut pentru beneficiile aduse în ceea ce privește siguranța și eficiența

codului. Utilizarea TypeScript poate îmbunătăți calitatea și fiabilitatea aplicațiilor și poate

facilita colaborarea între membrii echipei de dezvoltare.

Baza de date PostgreSQL cu ORM (Object-Relational Mapping) reprezintă o combinație

puternică pentru dezvoltarea aplicațiilor web și a altor tipuri de aplicații care necesită stocarea și

gestionarea datelor într-un mod eficient și fiabil. PostgreSQL este un sistem de gestionare a

bazelor de date relaționale (RDBMS) puternic și fiabil, în timp ce ORM-ul oferă un set de

instrumente și funcționalități pentru a simplifica interacțiunea cu baza de date, abstractizând

operațiunile SQL și permitând manipularea datelor utilizând obiecte în cadrul limbajului de

programare. Principalele avantaje ale utilizării PostgreSQL cu ORM includ:

Abstracție a operațiilor SQL: ORM-ul permite dezvoltatorilor să lucreze cu obiecte și

clase în cadrul limbajului de programare, abstractizând detaliile operațiilor SQL necesare pentru

interacțiunea cu baza de date. Acest lucru face codul mai ușor de scris, de înțeles și de întreținut.

Mapping obiect-relațional (ORM): ORM-ul facilitează maparea obiectelor și claselor din

codul aplicației la tabelele și relațiile din baza de date. Acest lucru elimină necesitatea de a scrie

manual interogările SQL și de a gestiona conversia datelor între formatul obiect și formatul

relațional.

Gestionarea relațiilor și tranzacțiilor: ORM-urile oferă suport pentru definirea și

gestionarea relațiilor între diferitele entități din aplicație, precum și pentru gestionarea

tranzacțiilor în cadrul bazei de date. Acest lucru facilitează dezvoltarea aplicațiilor complexe și

asigură consistența și integritatea datelor.

Securitate și performanță: PostgreSQL este cunoscut pentru securitatea și performanța sa

ridicate, oferind funcționalități avansate precum controlul accesului la date, criptarea și

14

optimizarea interogărilor. ORM-urile pot exploata aceste caracteristici pentru a asigura

securitatea și performanța adecvate în cadrul aplicațiilor.

Compatibilitate cu diferite limbaje de programare: Majoritatea ORM-urilor sunt

compatibile cu diverse limbaje de programare, inclusiv JavaScript, Python, Java și altele. Acest

lucru permite dezvoltatorilor să utilizeze PostgreSQL cu ORM în cadrul proiectelor lor,

indiferent de limbajul de programare preferat.

### Arhitectura informațiilor

Informații despre Utilizatori:

* Nume, prenume
* Adresă de e-mail
* Număr de telefon
* Informații despre vehicule (marca, model, număr de înmatriculare)
* Date de autentificare (nume de utilizator, parolă)

Informații despre Rezervări:

* Data și ora rezervării
* Durata rezervării
* Locația rezervată (numărul locului de parcare)
* Statutul rezervării (activă, anulată, finalizată)

Informații despre Parcare:

* Disponibilitatea locurilor de parcare (ocupate, libere)
* Informații despre locația parcajului (adresă, coordonate GPS)

Informații despre Accesul în Parcare:

* Date de identificare NFC sau numere de înmatriculare pentru accesul în parcare

### Arhitectura de comunicații interne

**Arhitectura de Comunicații:**

Arhitectura de comunicații va fi centralizată, cu toate componentele sistemului conectate la un singur punct central, care poate fi un router sau un switch.

**Componente ale Rețelei:**

* Router/Switch Central: Un dispozitiv central va fi folosit pentru a interconecta toate componentele sistemului. Acesta va gestiona rutele de comunicație și va dirija traficul între diferitele componente ale sistemului.
* Raspberry Pi: Dispozitivul Raspberry Pi va fi conectat direct la router/switch central și va servi ca nod central pentru procesarea datelor și gestionarea comunicațiilor între diversele componente ale sistemului.
* Camere LPR și Senzori: Camerele LPR și senzorii de parcare vor fi conectați la Raspberry Pi prin intermediul unui conector special.
* Dispozitive NFC: Cititoarele NFC pentru acces vor fi conectate la Raspberry Pi

**Estimări ale Resurselor:**

Capacitatea rețelei de comunicații depinde de volumul de date transferate între componentele sistemului și de frecvența actualizării datelor. Pentru o parcare de dimensiuni medii, o rețea Ethernet standard sau Wi-Fi ar trebui să fie suficientă pentru a gestiona traficul de date generat de camerele LPR, senzori și dispozitivele NFC. Se recomandă evaluarea necesităților specifice și dimensionarea corespunzătoare a rețelei pentru a asigura o performanță optimă.

### Diagrama de arhitectură a sistemului

Structura generală și integrată a sistemului nostru de management al locurilor de parcare este rezultatul combinației între hardware-ul, software-ul, rețeaua de comunicații și fluxul de informații descrise anterior. Iată cum arată această structură integrată:

**Hardware:**

Raspberry Pi: Dispozitivul central care gestionează și monitorizează locurile de parcare, procesând datele de la senzori și camerele LPR și controlând accesul utilizatorilor prin dispozitivele NFC.

Camere LPR: Detectează și înregistrează numerele de înmatriculare ale vehiculelor care intră și ies din parcarea noastră.

Senzori de Parcare: Detectează dacă locurile de parcare sunt ocupate sau libere.

Dispozitive NFC: Folosite pentru accesul în parcarea utilizatorilor.

**Software:**

Aplicație Frontend (Flutter): Interfața cu utilizatorul care permite utilizatorilor să vizualizeze locurile de parcare disponibile, să facă rezervări și să își gestioneze profilurile.

Aplicație Backend (Golang): Gestionarea datelor și logicii de afaceri, inclusiv procesarea rezervărilor, gestionarea utilizatorilor și a locurilor de parcare.

Bază de Date (PostgreSQL): Stocarea datelor despre utilizatori, rezervări, locuri de parcare și accesul în parcări.

**Comunicații:**

Conexiunea între Raspberry Pi și camerele LPR, senzorii de parcare și dispozitivele NFC se realizează prin intermediul unei rețele locale (LAN), folosind conexiuni Ethernet sau Wi-Fi, în funcție de disponibilitatea infrastructurii de rețea.

Conexiunea între aplicația frontend și backend se realizează prin intermediul unui protocol de comunicație, cum ar fi HTTP sau WebSocket.

**Informații:**

Datele despre utilizatori, rezervări, locuri de parcare și acces sunt stocate în baza de date PostgreSQL și sunt accesate și procesate de către aplicațiile frontend și backend.

## Proiectarea sistemului

**Tabela cars (Mașini):**

Dicționar de date:

* ID: ID-ul unic al mașinii.
* CreatedAt: Data și ora creării înregistrării.
* UpdatedAt: Data și ora ultimei actualizări a înregistrării.
* DeletedAt: Marcator pentru ștergerea logică a înregistrării.
* CarBrand: Marcă/model a mașinii.
* CarModel: Modelul mașinii.
* CarPlate: Numărul de înmatriculare al mașinii (unic).
* CarYear: Anul mașinii.
* IsParked: Indică dacă mașina este parcata sau nu.
* IsElectric: Indică dacă mașina este electrică sau nu.
* UserID: ID-ul utilizatorului asociat mașinii.
* IsSelected: Indică dacă mașina este selectată sau nu (opțional).

**Tabela reservations (Rezervări):**

Dicționar de date:

* ID: ID-ul unic al rezervării.
* CreatedAt: Data și ora creării înregistrării.
* UpdatedAt: Data și ora ultimei actualizări a înregistrării.
* DeletedAt: Marcator pentru ștergerea logică a înregistrării.
* Status: Statutul rezervării.
* UserID: ID-ul utilizatorului care a efectuat rezervarea.
* SpotID: ID-ul locului de parcare rezervat.
* StartTime: Data și ora începutului rezervării.
* EndTime: Data și ora sfârșitului rezervării.
* Price: Prețul rezervării.
* CarID: ID-ul mașinii asociate rezervării.
* IsPaid: Indică dacă rezervarea a fost plătită sau nu

**Tabela spots (Locuri de parcare):**

Dicționar de date:

* ID: ID-ul unic al locului de parcare.
* CreatedAt: Data și ora creării înregistrării.
* UpdatedAt: Data și ora ultimei actualizări a înregistrării.
* DeletedAt: Marcator pentru ștergerea logică a înregistrării.
* Label: Eticheta/denumirea locului de parcare.
* IsForDisabledPeople: Indică dacă locul de parcare este destinat persoanelor cu dizabilități sau nu.
* IsForElectricCar: Indică dacă locul de parcare este destinat mașinilor electrice sau nu.
* IsLocked: Indică dacă locul de parcare este blocat sau nu.
* ConfortLevel: Nivelul de confort al locului de parcare.
* IsEmpty: Indică dacă locul de parcare este liber sau nu.
* ParkingLotID: ID-ul parcării asociate locului de parcare.
* CarsParkedCount: Numărul de mașini parcate pe locul respectiv

**Tabela feedback (Feedback):**

Dicționar de date:

* ID: ID-ul unic al feedback-ului.
* CreatedAt: Data și ora creării înregistrării.
* UpdatedAt: Data și ora ultimei actualizări a înregistrării.
* DeletedAt: Marcator pentru ștergerea logică a înregistrării.
* Rating: Evaluarea acordată.
* RatingText: Comentariu asociat evaluării.
* ParkingLotID: ID-ul parcării asociate feedback-ului.
* UserID: ID-ul utilizatorului care a lăsat feedback-ul.

**Tabela users (Utilizatori):**

Dicționar de date:

* ID: ID-ul unic al utilizatorului.
* CreatedAt: Data și ora creării înregistrării.
* UpdatedAt: Data și ora ultimei actualizări a înregistrării.
* DeletedAt: Marcator pentru ștergerea logică a înregistrării.
* Username: Numele de utilizator al utilizatorului (unic).
* PasswordHash: Hash-ul parolei utilizatorului.
* Email: Adresa de email a utilizatorului (unică).
* PhoneNumber: Numărul de telefon al utilizatorului.

### Proiectarea bazei de date

**Tabela users (Utilizatori):**

* ID: ID-ul unic al utilizatorului (INTEGER, PRIMARY KEY).
* username: Numele de utilizator al utilizatorului (VARCHAR, UNIQUE, NOT NULL).
* password\_hash: Hash-ul parolei utilizatorului (VARCHAR, NOT NULL).
* email: Adresa de email a utilizatorului (VARCHAR, UNIQUE, NOT NULL).
* phone\_number: Numărul de telefon al utilizatorului (VARCHAR).
* created\_at: Data și ora creării înregistrării (TIMESTAMP, DEFAULT NOW()).
* updated\_at: Data și ora ultimei actualizări a înregistrării (TIMESTAMP).

**Tabela cars (Mașini):**

* ID: ID-ul unic al mașinii (INTEGER, PRIMARY KEY).
* car\_brand: Brandul mașinii (VARCHAR).
* car\_model: Modelul mașinii (VARCHAR).
* car\_plate: Numărul de înmatriculare al mașinii (VARCHAR, UNIQUE, NOT NULL).
* car\_year: Anul fabricației mașinii (VARCHAR).
* is\_parked: Indicator pentru a verifica dacă mașina este parcată sau nu (BOOLEAN, DEFAULT FALSE).
* is\_electric: Indicator pentru a verifica dacă mașina este electrică sau nu (BOOLEAN, DEFAULT FALSE).
* user\_id: ID-ul utilizatorului asociat mașinii (INTEGER, FOREIGN KEY către users).
* created\_at: Data și ora creării înregistrării (TIMESTAMP, DEFAULT NOW()).
* updated\_at: Data și ora ultimei actualizări a înregistrării (TIMESTAMP).

**Tabela reservations (Rezervări):**

* ID: ID-ul unic al rezervării (INTEGER, PRIMARY KEY).
* status: Starea rezervării (VARCHAR sau ENUM).
* user\_id: ID-ul utilizatorului care a făcut rezervarea (INTEGER, FOREIGN KEY către users).
* spot\_id: ID-ul locului de parcare rezervat (INTEGER, FOREIGN KEY către spots).
* start\_time: Data și ora de început a rezervării (TIMESTAMP).
* end\_time: Data și ora de sfârșit a rezervării (TIMESTAMP).
* price: Prețul rezervării (DECIMAL).
* car\_id: ID-ul mașinii asociate rezervării (INTEGER, FOREIGN KEY către cars).
* is\_paid: Indicator pentru a verifica dacă rezervarea este plătită sau nu (BOOLEAN, DEFAULT FALSE).
* created\_at: Data și ora creării înregistrării (TIMESTAMP, DEFAULT NOW()).
* updated\_at: Data și ora ultimei actualizări a înregistrării (TIMESTAMP).

**Tabela feedback (Feedback):**

* ID: ID-ul unic al feedback-ului (INTEGER, PRIMARY KEY).
* rating: Evaluarea acordată (INTEGER sau BYTE).
* rating\_text: Comentariul asociat evaluării (TEXT).
* parking\_lot\_id: ID-ul parcării pentru care se dă feedback (INTEGER, FOREIGN KEY către parking\_lots).
* user\_id: ID-ul utilizatorului care a dat feedback (INTEGER, FOREIGN KEY către users).
* created\_at: Data și ora creării înregistrării (TIMESTAMP, DEFAULT NOW()).
* updated\_at: Data și ora ultimei actualizări a înregistrării (TIMESTAMP).

#### Obiecte de date și structuri de date rezultante

Instrucțiuni: Pentru fiecare obiect de date funcțional, specificați structura/ile de date care va/vor fi utilizată/e pentru a stoca și procesa datele. Descrieți orice structuri de date care sunt o parte importantă a sistemului, inclusiv structurile de date majore care sunt transmise între componente.

#### Fișiere și baze de date

Instrucțiuni: Creați un model de date fizic care descrie stocarea și manipularea datelor în cadrul arhitecturii sistemului. Descrieți structurile de fișiere și locațiile acestora.

##### Baze de date

Instrucțiuni: Furnizați proiectarea detaliată a fișierelor DBMS.

##### Fișiere non-DBMS

Instrucțiuni: Furnizați descrierea detaliată a tuturor fișierelor non-DBMS și includeți o descriere narativă a utilizării fiecărui fișier care identifică dacă fișierul este utilizat pentru intrare, ieșire sau ambele, și dacă fișierul este un fișier temporar. De asemenea, oferiți o indicație a modulului care citește și scrie fișierul și includeți structurile de fișiere (faceți referire la dicționarul de date). În funcție de caz, informațiile despre structura fișierului ar trebui să includă următoarele:

* Structurile de înregistrare, cheile de înregistrare sau indexele și elementele de date referite în înregistrări
* Lungimea înregistrării (lungime fixă sau maximă variabilă) și factorii de blocare
* Estimarea dimensiunii fișierului sau volumului de date din fișier
* Definiția frecvenței de actualizare a fișierului (dacă fișierul face parte dintr-un sistem bazat pe tranzacții online, furnizați numărul estimat de tranzacții per unitate de timp și media statistică, modul și distribuția acestor tranzacții.)
* Specificații de backup și recuperare

### Conversii de date

Instrucțiuni: Inserați orice documente care descriu conversiile de date necesare sau furnizați o referință către locul unde sunt stocate.

### Interfețe utilizator

Instrucțiuni: Furnizați o descriere a fiecărei clase de utilizatori sau roluri asociate cu sistemul. O clasă de utilizatori se diferențiază în funcție de modurile în care utilizatorii interacționează cu sistemul sau situația propusă. Factorii care disting o clasă de utilizatori includ responsabilitățile comune, nivelurile de competență, activitățile de lucru și modurile de interacțiune cu sistemul. În acest context, un utilizator este oricine interacționează cu sistemul propus, inclusiv utilizatori operaționali, personal de introducere a datelor, operatori de sistem, personal de suport operațional, cei care întrețin sistemul și formatori. Pentru fiecare clasă de utilizatori, furnizați estimări ale numărului total de utilizatori anticipați, un număr maxim de utilizatori simultani și numărul de utilizatori externi.

#### Intrări

Instrucțiuni: Furnizați o descriere a mijloacelor de intrare folosite de utilizator/operator pentru a furniza informații sistemului.

Arătați o mapare către fluxurile de date la nivel înalt (de exemplu, ecranele de introducere a datelor, cititoarele optice de caractere, scanerele de coduri de bare, etc.). Dacă este cazul, tipurile de înregistrări de intrare, structurile de fișiere și structurile de baze de date furnizate în secțiunea pentru Proiectarea datelor pot fi referite. Includeți definițiile elementelor de date sau faceți referire la dicționarul de date.

Furnizați aspectul tuturor ecranelor de intrare de date sau interfețe grafice (GUI-uri) (de exemplu, ferestre). Definiți toate elementele de date asociate fiecărui ecran sau GUI sau faceți referire la dicționarul de date. Furnizați criterii de editare pentru elementele de date, inclusiv valori specifice, interval de valori, obligatoriu/opțional, valori alfanumerice și lungime. Discutați controalele de introducere a datelor pentru a preveni trecerea peste editare. Discutați mesajele diverse asociate cu intrările utilizatorului/operatorului.

#### Ieșiri

Instrucțiuni: Ieșirile sistemului includ rapoarte, ecrane de afișare a datelor și interfețe grafice, rezultatele interogărilor, etc. Fișierele de ieșire descrise în secțiunea pentru Proiectarea Datelor pot fi referite. Ar trebui să fie furnizate următoarele, dacă este potrivit:

• Identificarea codurilor și numelor pentru rapoarte și ecranele de afișare a datelor

• Descrierea conținutului raportului și al ecranului (furnizați o reprezentare grafică a fiecărei structuri și definiți toate elementele de date asociate cu structura sau faceți referire la dicționarul de date)

• Descrierea scopului ieșirii, inclusiv identificarea utilizatorilor principali

• Descrierea oricăror restricții de acces sau considerații de securitate

### Proiectarea interfețelor cu utilizatorul

Instrucțiuni: Introduceți interfețele proiectate împreună cu descrierea lor.

## Scenarii de utilizare

Instrucțiuni: Descrieți funcționalitatea generală a sistemului din perspectiva utilizatorilor și furnizați un flux de execuție sau operațional al sistemului prin scenarii operaționale care oferă descrieri pas cu pas despre modul în care sistemul propus ar trebui să funcționeze și să interacționeze cu utilizatorii săi și interfețele externe într-un set dat de circumstanțe. Scenariile leagă împreună toate părțile sistemului, utilizatorii și alte entități prin descrierea modului în care interacționează, și pot fi folosite și pentru a descrie ce nu ar trebui să facă sistemul.

Scenariile de utilizare ar trebui să fie descrise pentru toate modurile operaționale, tranzacțiile și toate clasele de utilizatori identificate pentru sistemul propusFiecare scenariu ar trebui să includă evenimente, acțiuni, stimuli, informații și interacțiuni în măsura în care este necesar pentru a oferi o înțelegere cuprinzătoare a aspectelor operaționale ale sistemului propus.

## Proiectare de detaliu

Instrucțiuni: Furnizați informațiile necesare echipei de dezvoltare a sistemului pentru a construi și integra efectiv componentele hardware, pentru a codifica și integra componentele software, și pentru a interconecta segmentele hardware și software într-un produs funcțional.

În plus, adresați procedurile detaliate pentru combinarea pachetelor separate externe într-un singur sistem.

### Proiectare hardware de detaliu

Instrucțiuni: Furnizați suficiente informații detaliate despre fiecare dintre componentele hardware individuale pentru a construi și/sau achiziționa corect toate componentele hardware pentru sistem (sau pentru a integra articole COTS). Dacă există multe componente sau dacă documentația componentelor este extensivă, plasați-o într-un anex. Adăugați diagrame și informații suplimentare, dacă este necesar, pentru a descrie fiecare componentă și funcțiile sale adecvat. Ar trebui să fie urmate practici standard din industrie pentru specificațiile componentelor.

Pentru componente externe, identificați furnizorul specific și denumirile și numerele de model corespunzătoare. Includeți următoarele informații în proiectele detaliate ale componentelor, după caz:

* Cerințe de intrare de alimentare pentru fiecare componentă
* Impedanțe și stări logice ale semnalului
* Specificații ale conectorilor (serial/paralel, 11 pini, mascul/femelă, etc.)
* Specificații ale spațiului de memorie și/sau stocare
* Cerințe ale procesorului (viteză și funcționalitate)
* Reprezentare grafică care prezintă numărul de articole hardware (de ex., servere, dispozitive I/O, monitoare, imprimante etc.), și poziționarea relativă a componentelor între ele

Acestea pot fi introduse ca anexe ale acestui document

### Proiectare software de deatliu

Instrucțiuni: Furnizați o descriere detaliată pentru fiecare serviciu software de sistem care abordează următoarele atribute ale serviciului software. O mare parte din informațiile care apar în această secțiune ar trebui să fie conținute în antete/prologuri și secțiunile de comentarii ale codului sursă pentru fiecare componentă, subsistem, modul și subrutină. În cazul în care este așa, această secțiune poate consta în mare parte din referințe sau extrase din diagramele anotate și codul sursă. Orice diagrame sau extrase din codul sursă referite ar trebui furnizate la orice revizii de proiectare.

* Identificator serviciu - Identificatorul unic și/sau numele serviciului software
* Clasificare - Tipul de serviciu (de exemplu, aplicație, serviciu de date, etc.)
* Definiție - Scopul specific și semnificația semantică a serviciului
* Cerințe - Cerințele specifice funcționale sau nonfuncționale pe care serviciul le satisface
* Structuri de date interne - Structurile de date interne pentru serviciu
* Constraingeri - Orice relevante, presupuneri, limite sau constrângeri pentru serviciu (aceasta ar trebui să includă constrângerile de timp, de stocare sau de stare a serviciului și ar putea include reguli pentru interacțiunea cu serviciul (cuprinzând precondiții, postcondiții, invariante, alte constrângeri privind valorile de intrare sau ieșire și valorile locale sau globale, formatele datelor și accesul la date, sincronizarea, excepțiile, etc.))
* Compoziție - O descriere a utilizării și semnificației subserviciilor care fac parte din serviciu
* Utilizatori/Interacțiuni - O descriere a colaborării serviciului cu alte servicii (ce alte servicii utilizează această entitate? ce alte servicii utilizează această entitate (inclusiv orice efecte secundare pe care le-ar putea avea acest serviciu asupra altor părți ale sistemului)? acest lucru include metoda de interacțiune, precum și interacțiunea în sine. Designurile orientate pe obiecte ar trebui să includă o descriere a oricăror sub-clase, super-clase și meta-clase cunoscute sau anticipate)
* Procesare - O descriere precisă a modului în care serviciul își îndeplinește responsabilitățile (aceasta ar trebui să cuprindă o descriere a oricăror algoritmi utilizați; modificări ale stării; complexitate relevantă în timp sau spațiu; concurență; metode de creare, inițializare și curățare; și gestionarea condițiilor excepționale)
* Interfețe/Exporturi - Setul de servicii (resurse, tipuri de date, constante, subrutine și excepții) pe care serviciul le oferă (definiția sau declarația precisă a fiecărui astfel de element ar trebui să fie prezentă, împreună cu comentarii sau adnotări care descriu semnificațiile valorilor, parametrilor etc.; pentru fiecare element de serviciu descris, includeți sau furnizați o referință în discuția sa la o descriere a atributelor sale importante ale serviciului software (Identificator Componentă, Clasificare, Limbaj, Estimare SLOC, Definiție, Responsabilități, Cerințe, Structuri de Date Interne, Constraingeri, Compoziție, Utilizări/Interacțiuni, Resurse, Procesare și Interfețe/Exporturi))
* Raportare Design și Integrare - Dacă este inclus, furnizați detalii despre traficul și volumele de date.

### Proiectare detaliată de securitate

Instrucțiuni: Furnizați informații detaliate pentru fiecare dintre componentele individuale de securitate hardware. Specificați elementele de mai jos, după cum este necesar.

• Autentificare

• Autorizare

• Jurnalizare(arhivare) și auditare

• Criptare

• Utilizarea porturilor de rețea

• Detectare și prevenire a intruziunilor (în special dacă este găzduită într-un centru de date care nu aparține de CMS).

### Proiectare de detaliu pentru performanța sistemului

Instrucțiuni: Furnizați o reprezentare grafică cu informații detaliate pentru fiecare dintre componentele hardware individuale de performanță și fiabilitate, pentru a include elementele de mai jos:

• Cerințe/estimări de capacitate și volum

• Așteptări de performanță

• Cerințe de disponibilitate

• Proiectare de performanță pentru a îndeplini cerințele de capacitate

• Proiectare de fiabilitate pentru a îndeplini cerințele de disponibilitate

• Proiectare de backup, recuperare și arhivare

Identificați punctele unice de eșec și, dacă este relevant, descrieți proiectarea de disponibilitate ridicată (de exemplu, clustering).

### Proiectare detaliată a comunicațiilor interne (între componente)

Instrucțiuni: Dacă sistemul include mai mult de o componentă, este posibil să existe o cerință pentru comunicații interne pentru a schimba informații, a furniza comenzi sau a susține funcțiile de intrare/ieșire. Furnizați suficiente informații detaliate despre proiectarea comunicațiilor pentru a construi și/sau a achiziționa corect componentele de comunicații pentru sistem. Includeți următoarele informații în proiectările detaliate ale componentelor, după caz:

* Numărul de servere și clienți care urmează să fie incluși pe fiecare rețea de zonă
* Specificații pentru cerințele de sincronizare și control al busului
* Format(e) pentru datele schimbate între componente
* Reprezentare grafică a conectivității între componente, arătând direcția fluxului de date (dacă este aplicabil), și distanțele aproximative între componente (informațiile ar trebui să furnizeze suficient detaliu pentru a susține achiziționarea hardware-ului pentru finalizarea instalării într-o anumită locație)
* Topologia LAN

## Controale pentru verificarea integrității sistemului

Instrucțiuni: Furnizați specificații de proiectare pentru următoarele nivele minime de control și orice controale suplimentare adecvate sau necesare:

* Securitate internă pentru a restricționa accesul la datele critice doar pentru acele tipuri de acces necesare de către utilizatori/operatori
* Proceduri de audit pentru a îndeplini cerințele de control, raportare și perioade de reținere pentru rapoartele operaționale și de management
* Piste de auditare a aplicațiilor pentru a audita dinamic accesul la recuperare la datele critice designate
* Tabele standard care urmează să fie utilizate sau solicitate pentru validarea câmpurilor de date
* Procese de verificare pentru adăugarea, ștergerea sau actualizarea datelor critice
* Capacitatea de a identifica toate informațiile de auditare prin identificarea utilizatorului, identificarea terminalului de rețea, dată, oră și datele accesate sau modificate.

Anexa A: Gestiunea modificărilor documentului

Instrucțiuni: Furnizați informații despre modul în care dezvoltarea și distribuția documentului va fi controlată și urmărită. Utilizați tabelul de mai jos pentru a furniza numărul de versiune, data versiunii, autorul/deținătorul versiunii și o scurtă descriere a motivului pentru crearea versiunii revizuite.

Tabel 1 – Înregistrarea modificărilor asupreaa documentului curent

| versiune | Data | Autorul/Deținătorul | Descriere |
| --- | --- | --- | --- |
| <X.X> | <ZZ/LL/AAAA> | <nume autor> | <Descrierea modificării> |
| <X.X> | <ZZ/LL/AAAA> | <nume autor> | <Descrierea modificării> |
| <X.X> | <ZZ/LL/AAAA> | <nume autor> | <Descrierea modificării> |

Anexa B: Acronime

*Instrucțiuni: Furnizați o listă de acronime și traduceri literale asociate utilizate în cadrul documentului. Enumerați acronimele în ordine alfabetică folosind un format tabular, așa cum este ilustrat mai jos.*

Tabel 2 - Acronime

| Acronim | Forma completă |
| --- | --- |
| <Acronim> | <Forma completă> |
| <Acronim> | <Forma completă> |
| <Acronim> | <Forma completă> |

Anexa C Documente la care se face referire

*Instrucțiuni: Sintetizați relația acestui document cu alte documente relevante. Furnizați informații de identificare pentru toate documentele folosite pentru a ajunge la și/sau referite în acest document (de exemplu, documente conexe și/sau asociate, documente prealabile, documentație tehnică relevantă, etc.).*

Tabel 3 – Documente la care se facce referire

| Nume document | Locație sau URL | Dată emitere document |
| --- | --- | --- |
| < Nume document > | <Locație sau URL> | <ZZ/LL/AAAA> |
| < Nume document > | <Locație sau URL> | <ZZ/LL/AAAA> |
| < Nume document > | <Locație sau URL> | <ZZ/LL/AAAA> |