UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**PROIECT DE DIPLOMĂ**

Coordonator științific: Șef lucr. Dr. Inf. Pitic Antoniu Gabriel

Absolvent: Andreiu Dan

Specializarea: Calculatoare

- Sibiu, 2024 -

UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**Aplicație pentru pontare și împărțirea task-urilor unui proiect în echipă**

Coordonator științific: Șef lucr. Dr. Inf. Pitic Antoniu Gabriel

Absolvent: Andreiu Dan

Specializarea: Calculatoare

- Sibiu, 2024

Cuprins

[1 Prezentarea temei 4](#_Toc170255113)

[1.2 Descrierea temei 4](#_Toc170255114)

[1.3 Scop și obiective 4](#_Toc170255115)

[1.4 Tehnologii utilizate 5](#_Toc170255116)

[1.5 Organizarea lucrării 5](#_Toc170255117)

[2 Considerente teoretice 7](#_Toc170255118)

[2.1 .NET Core 1.0 7](#_Toc170255119)

[2.1.1 Istorie 7](#_Toc170255120)

[2.1.2 Portabilitate 7](#_Toc170255121)

[2.1.3 Diferențe față de .NET 8 7](#_Toc170255122)

[2.2 C# 8](#_Toc170255123)

[2.2.1 Structura unui program 8](#_Toc170255124)

[2.2.2 Variabile. Constante. Expresii 8](#_Toc170255125)

[2.2.3 Instrucțiuni condiționate 9](#_Toc170255126)

[2.2.4 Instrucțiuni repetitive 10](#_Toc170255127)

[2.3 Angular 2 11](#_Toc170255128)

[2.3.1 Istorie 11](#_Toc170255129)

[2.3.2 Diferențe față de Angular 17 11](#_Toc170255130)

[2.4 TypeScript 12](#_Toc170255131)

[2.4.1 Introducere 12](#_Toc170255132)

[2.4.2 Structura unui program 12](#_Toc170255133)

[2.4.3 Variabile. Constante. Expresii 13](#_Toc170255134)

[2.5 Source control 14](#_Toc170255135)

[2.5.1 Git 14](#_Toc170255136)

[2.5.2 Fork 15](#_Toc170255137)

[2.5.3 GitHub 16](#_Toc170255138)

[2.6 SQLite 18](#_Toc170255139)

[2.6.1 Introducere 18](#_Toc170255140)

[2.6.2 Forme normale 18](#_Toc170255141)

[2.7 Figma Mockups 19](#_Toc170255142)

[3 Descrierea formală a aplicației 21](#_Toc170255143)

[3.1 Actori 21](#_Toc170255144)

[3.2 Mockups 21](#_Toc170255145)

[3.2.1 Landing 21](#_Toc170255146)

[3.2.2 Login 22](#_Toc170255147)

[3.2.3 Registration 23](#_Toc170255148)

[3.2.4 Dashboard 23](#_Toc170255149)

[3.2.5 Tasks board 24](#_Toc170255150)

[3.2.6 Task 24](#_Toc170255151)

[3.2.7 Backlog 25](#_Toc170255152)

[3.2.8 Calendar 25](#_Toc170255153)

[3.3 Arhitectura aplicației 26](#_Toc170255154)

[4 Detalii de implementare 30](#_Toc170255155)

[4.1 Descrierea bazei de date 30](#_Toc170255156)

[4.1.2 Implementarea bazei de date 34](#_Toc170255157)

[4.2 Use Cases 35](#_Toc170255158)

[4.2.1 Autentificare 35](#_Toc170255159)

[4.2.2 Înregistrare 39](#_Toc170255160)

[4.2.3 Dashboard 41](#_Toc170255161)

[4.2.4 Vizualizarea proiectelor și crearea unui nou proiect 42](#_Toc170255162)

[4.2.5 Vizualizare sprint-uri și crearea unui nou sprint 43](#_Toc170255163)

[4.2.6 Crearea unei noi sarcini 45](#_Toc170255164)

[4.2.7 Vizualizarea statusului pentru sprintul activ 45](#_Toc170255165)

[4.2.8 Pontaj 47](#_Toc170255166)

[4.2.9 Vizualizarea unei sarcini 49](#_Toc170255167)

[4.2.10 Vizualizarea informațiilor personale 50](#_Toc170255168)

[4.2.11 Logout 51](#_Toc170255169)

[4.3 Descrierea API-ului 52](#_Toc170255170)

[4.3.1 User Controller 52](#_Toc170255171)

[4.3.2 Project Controller 54](#_Toc170255172)

[4.3.3 Backlog Controller 58](#_Toc170255173)

[4.3.4 Sprint Controller 59](#_Toc170255174)

[4.3.5 Task Controller 65](#_Toc170255175)

[4.3.6 Timesheet Controller 69](#_Toc170255176)

[5 Concluzii și dezvoltări ulterioare 72](#_Toc170255177)

[5.1 Concluzii 72](#_Toc170255178)

[5.2 Dezvoltări ulterioare 72](#_Toc170255179)

[Bibliografie 74](#_Toc170255180)

# 1 Prezentarea temei

**1.1 Introducere**

Una din principalele probleme în dezvoltarea de aplicații software pentru clienți este nivelul major de incertitudine când vorbim despre livrarea unui produs care să atingă calitatea dorită de client în contrast cu simpla îndeplinire a obiectivelor de buget și timp.

Termenul Agile se referă la o metodologie pentru dezvoltarea de aplicații software care anticipă nevoia de flexibilitate și aplică un nivel de pragmatism pentru produsul final livrat clientului.

Această metodologie înceară să rezolve problema menționată prin comunicarea frecventă între echipa de dezvoltare a aplicației și client, precum și prin adaptarea la schimbarea planului de implementare. Astfel, orice incertitudine a programatorilor în legăură cu o componentă a aplicației poate fi discutată cu clientul înainte de implementare. Așadar, prin aceste informări continue, se asigură că fiecare componentă a aplicației se află în pragul de calitate dorit de client în momentul livrării.

## 1.2 Descrierea temei

Tema aleasă, cu titlul „Aplicație colaborativă pentru gestionarea proiectelor Agile” își dorește implementarea unei aplicații utile echipelor de dezvoltare a aplicaților software pentru clienți care să ușureze folosirea metodologiilor Agile pentru o calitate ridicată a produsului livrat. De asemenea, în această temă se dorește implementarea unui mecanism de pontaj semi-automat.

## 1.3 Scop și obiective

Scopul principal este acela de a determina un model intuitiv pentru așezarea în pagină a componentelor necesare pentru dezvoltarea unei aplicații după modelul Agile, precum și un model de calendar ușor de folosit pentru mecanismul de pontaj, apoi realizarea unei aplicații web conectată cu o interfață de programare a aplicației pentru acestea.

Pentru a atinge acest scop ne-am propus următoarele obiective principale:

* Determinarea tehnologiilor adecvate pentru implementarea aplicațiilor
* Descrierea formală a aplicației, pentru o mai bună organizare a implementării
* Dezvoltarea interfeței de programare a aplicației
* Implementarea aplicației web
* Testarea în mediul de producție
* Realizarea documentației tehnice

## 1.4 Tehnologii utilizate

Pentru crearea interfeței de programare a aplicației am folosit următoarele tehnologi Microsoft:

* ASP.NET MVC – pentru a realiza conexiunea între cererile de tip HTTP venite de la aplicația web și baza de date a aplicației împreună cu logica principală
* C# – pentru implementarea logicii aplicației, a modelelor și a bazei de date
* SQLite – pentru salvarea datelor

Pentru dezvoltarea aplicației web am folosit tehnologiile următoare:

* TypeScript – pentru modelarea datelor provenite din interfața de programare a aplicației și realizarea cererilor de tip HTTP către aceasta
* SCSS + HTML5 – pentru design-ul site-ului
* Brave debug mode – pentru depanare

Alte tehnologii utilizate:

* Postman – pentru testarea cererilor de tip HTTP către interfața de programare a aplicației
* DBeaver – pentru accesarea facilă a datelor din baza de date

## 1.5 Organizarea lucrării

Lucrarea este structurată în 5 capitole. Primul capitol este unul introductiv. Al doilea capitol descrie pe scurt tehnologiile utilizate în implementarea aplicației și diferitele instrumente și biblioteci care ne-au permis să descriem formal, să implementăm, să testăm și să gestionăm sursele în toate etapele dezvoltării aplicației.

Capitolul 3 este o descriere formală a aplicației, cuprinzând cazurile de utilizare principale, descrise conform standardelor în vigoare, Mockups și arhitectură.

“Detalii de implementare” este un capitol care prezintă o parte dintre provocarile cu care ne-am confruntat, cuprinzând capturi de ecran ale aplicației și o descriere a utilizării aesteia, dar și bucăti de cod și explicarea acestora.

Ultimul capitol, Concluzii și dezvoltări ulterioare, conține două părti. Prima reprezintă o privire de ansamblu asupra ce am reusit să realizăm, iar a doua parte planurile de viitor.

# 2 Considerente teoretice

## 2.1 .NET Core 1.0

### 2.1.1 Istorie

În data de 27 iunie 2016 Microsoft a lansat o nouă versiune pentru .NET Framework, numită .NET Core 1.0. Această versiune urma să devină fundația tuturor platformelor viitoare .NET, aducând flexibilitate, performanță și suport extins pentru diferite platforme.

.NET Core 1.0 a fost conceput cu suport puternic pentru dezvoltarea de microservicii și aplicații cloud-native, precum și o platformă de dezvoltare gratuită, open-source, care putea rula pe sistemele de operare Windows, macOS si Linux.

Ultima versiune, .NET Core 1.0.16, apărută în data de 14 mai 2019, a adresat vulnerabilitățile de securitate existente. Suportul pentru această versiune de .NET Core a fost încheiat de către Microsoft în data de 27 iunie 2019, la scurt timp după apariția ultimei versiuni.

### 2.1.2 Portabilitate

În .NET Core 1.0, portabilitatea a reprezentat punctul forte în dezvoltarea acestui nou framework. Aceasta a format noi oportunități pentru dezvoltatori și a ajutat la consolidarea .NET Core ca o platformă modernă care poate face față diversității cerințelor de dezvoltare a aplicațiilor software.

Caracteristicile cheie ale portabilității în .NET Core 1.0 sunt posibilitatea de a scrie și a rula aplicații pe mai multe sisteme de operare, precum și pe diferite arhitecturi hardware, facilitând dezvoltarea de aplicații pentru dispozitive mobile sau servere. De asemenea, .NET Core 1.0 a fost conceput să fie modular, introducând doar părțile necesare ale framework-ului în aplicație, ceea ce facilita adaptarea la diferite cerințe. Interoperabilitatea si suportul pentru cloud au adus și ele un beneficiu major asupra portabilității în .NET Core 1.0.

### 2.1.3 Diferențe față de .NET 8

În anul 2019, Microsoft a lansat .NET Framework 4.8.1, aceasta fiind ultima versiune de .NET Framework existentă. Astfel, din anul 2020 .NET Core a trecut la denumirea simplificată .NET, prima versiune cu această denumire fiind .NET 5, care este succesoarea versiunii .NET Core 3.1.

Cea mai recentă versiune de .NET, apărută în data de 14 noiembrie 2023 este .NET 8. Această versiune, comparativ cu .NET Core 1.0, menține caracteristicile principale de portabilitate, funcționalități și interfețe de program a aplicațiilor, dar vine cu îmbunătățiri semnificative. Printre multele îmbunătățiri, ies în evidență interoperabilitatea cu alte tehnologii și limbaje de programare, optimizări semnificative în performanță și scalabilitate, precum și îmbunătățiri la nivelul de securitate.

Pe lângă îmbunătățirile aduse caracteristicilor existente, .NET 8 a introdus și caracteristici noi precum suport pentru machine learning și edge computing. Totodată cu actualizarea versiunilor de .NET, Microsoft a actualizat și mediile de dezvoltare pentru .NET oferind o experiență îmbunătățită pentru programatori. (Microsoft, ASP.NET, 2024)

## 2.2 C#

### 2.2.1 Structura unui program

Structura unui program C# conține unul sau mai multe fișiere format din mai multe părți esențiale. Aceste părți sunt, în mod normal, organizate într-un mod specific pentru a asigura un cod ușor de citit și funcțional.

În partea de sus a fișierului se găsesc importurile folosite pentru a accesa clase și metode din biblioteci. Apoi, fișierele pot conține namespace-uri care ajută la împărțirea codului. Un namespace conține tipuri precum clase, structuri, interfețe, enumerații sau metode delegate. Clasa, la rândul ei, conține variabile, metode și proprietăți.

### 2.2.2 Variabile. Constante. Expresii

Variabilele în C# reprezintă o locație din memorie care poate fi rescrisă și refolosită de oricâte ori este nevoie. Fiecare variabilă trebuie să aibă un tip care determină ce valori pot fi stocate în aceasta. Tipurile de variabile predefinite sunt “decimal”, “integral”, “floating point”, “boolean” și “nullable”, dar un utilizator își poate defini propriul tip de variabilă precum o clasă. În C#, sintaxa de definire a unei variabile este formată din tipul de dată în partea stângă, urmat de lista de variabile separate de virgulă dacă sunt mai mult de una. Variabila poate fi inițializată în declararea acesteia.

int sum = 4;

Constantele sunt valori care nu pot fi modificate pe parcursul execuției programului, odată ce acestea au fost declarate și inițializate. De regulă sunt definite folosind cuvântul “const” și trebuie inițializate la momentul declarării. Doar tipurile de date predefinite în C# pot fi declarate ca și constante. Astfel, tipurile de date definite de utilizator, precum clase, structuri și vectori nu pot fi constante și este necesară folosirea cuvântului cheie “readonly”.

const string TaskId = "TaskId";

readonly IUnitOfWork unitOfWork;

Expresiile sunt secvențe de operatori și operanzi evaluate pentru a produce o valoare. Acestea sunt folosite, de obicei, în asignări de variabile, instrucțiuni condiționale, instrucțiuni repetitive sau alte operații care necesită evaluarea unui calcul matematic sau a unei logici.

### 2.2.3 Instrucțiuni condiționate

În C#, intrucțiunile condiționate sunt folosite pentru a controla execuția programului prin rularea diferitelor blocuri de cod în funcție de condiția sau expresia folosită. Aceste instrucțiuni sunt cruciale în dezvoltarea unui program dinamic.

Instrucțiunea condiționată “if” execută blocul de cod atunci când condiția este evaluată ca și adevărat. Această instrucțiune poate lua și forma de “if-else” în care execută codul din blocul else atunci când expresia este falsă.

if (sum > 10)

{

//bloc de cod

}

else

{

//alt bloc de cod

}

Setarea mai multor condiții de “if” succesive poate fi realizată folosind instrucțiunea “else if” după instrucțiunea inițială “if”, iar fiecare instucțiune de tipul „else if” o să aibă condiția ei proprie, apoi la final se găsește instrucțiunea “else”.

if (sum > 10)

{

//bloc de cod

}

else if (sum < 0)

{

//bloc de cod

}

else

{

//bloc de cod

}

Ultimul tip de instrucțiune condiționată disponibil în C# este instrucțiunea “switch” care conține mai multe cazuri posibile de evaluat pentru una sau mai multe variabile și un caz default care se execută când niciun caz anterior nu a fost executat. Cazurile acestei instrucțiuni pot avea și condiții suplimentare numite gărzi.

switch ((a, b))

{

case ( > 0, > 0) when a == b:

//bloc de cod

break;

case ( > 0, > 0):

//bloc de cod

break;

default:

//bloc de cod

break;

}

### 2.2.4 Instrucțiuni repetitive

Cât timp condiția de oprire din instrucțiune nu este evaluată ca și adevărat, instrucțiunile repetitive au obligația de a executa un bloc de cod. În C# se regăsesc patru tipuri de intrucțiuni repetitive.

Instrucțiunea “for” execută blocul de cod din aceasta cât timp expresia definită este adevărată. Comparativ cu restul instrucțiunilor repetitive din C#, aceasta poate conține un pas și o definire sau inițializare a unei variabile în aceasta. Toate secțiunile instrucțiunii “for” sunt opționale.

for(int i = 0; i < 10; i++)

{

//bloc de cod

}

“Foreach” este o intrucțiune repetitivă care execută blocul de cod pentru fiecare element dintr-o colecție și iterează prin aceasta.

foreach(char caracter in nume)

{

//bloc de cod

}

Instrucțiunea repetitivă “do” rulează blocul de cod cel puțin o dată după care verifică condiția de oprire.

do

{

//bloc de cod

}while (username.Length < 10);

Instrucțiunea “while” este asemănătoare cu instrucțiunea “do”, doar că aceasta verifică condiția de oprire înainte de a intra în blocul de cod, astfel existând șansa ca blocul respectiv să nu fie executat niciodată. (Microsoft, C#, 2024)

while (username.Length < 10)

{

//bloc de cod

}

## 2.3 Angular 2

### 2.3.1 Istorie

Angular este un framework gratuit și open-source bazat pe limbajul de programare TypeScript pentru aplicațiile web formate dintr-o singură pagină. Acesta rulează prin Node.js și este dezvoltat de Google împreună cu alți dezvolatori individuali ai comunității. În anul 2016 echipa de programatori a lansat versiunea Angular 2, versiune care a venit ca o remodelare completă a versiunii anterioare numită „AngularJS” astfel modificând și numele noii versiuni.

În prezent echipa Angular nu mai oferă suport pentru versiunea Angular 2, iar singurele versiuni care dispun de suport pe o perioadă îndelungată sunt Angular 16, Angular 17 și Angular 18.

### 2.3.2 Diferențe față de Angular 17

În prezent, Angular primește îmbunătățiri de două ori pe an, fiecare versiune continuând să îmbunătățească performanța, să simplifice dezvoltarea aplicațiilor și sa aducă noi calități. Astfel, diferențele dintre Angular 17 și Angular 2 sunt resimțibile, unele dintre acestea fiind funcționalitatea și bibliotecile. În Angular 17 sunt existente componentele standalone care reduc dependințele, acest tip de componentă devenind cel implicit la crearea unui nou proiect în această versiune.

Încă din versiunea Angular 9 a fost introdus compilatorul Ivy pentru a îmbunătăți timpul de dezvoltare și a reduce dimensiunea bundel-urilor, acesta fiind îmbunătățit în continuare până la versiunea Angular 17.

Versiunea Angular 17 a beneficiat de îmbunătățiri de performanță și prin adăugarea tehnicilor de “hydration” care permite ca pagina să fie vizibilă si funcțională rapid, apoi să devină interactivă fără a mai fi necesară reîncărcarea acesteia. Altă tehnică care îmbunătățește performanța este „Deferred Loading” care reduce timpul de încărcare al aplicației prin întârzierea încărcării componentelor.

Pe lângă sporirea performanței, versiunea Angular 17 a obținut și îmbunătățiri majore asupra stabilității, migrării și a ecosistemului. (Wikipedia, Angular (web framework), 2024)

## 2.4 TypeScript

### 2.4.1 Introducere

TypeScript este un limbaj de programare dezvoltat de către Microsoft în anul 2012. Acesta este bazat pe limbajul de programare JavaScript căruia îi adaugă tipuri de date statice opționale. Orice cod scris în TypeScript este tradus automat în JavaScript pentru a fi rulat. Astfel, orice cod scris în JavaScript este valid și în TypeScript, dar poate avea probleme la verificarea tipurilor de date.

### 2.4.2 Structura unui program

Un program realizat folosind limbajul de programare TypeScript conține mai multe fișiere generate automat. Unul din fișiere este “tsconfig.json” care conține opțiunile pentru compilatorul TypeScript. Aici se pot modifica opțiunile de compilare cum ar fi spre exemplu configurarea pentru directorul de ieșire pentru fișierele compilate sau fișierele incluse în program.

Fișierul “index.ts” este punctul de intrare al aplicației. De obicei aici se importă și se utilizează funcționalitățile definite în alte module sau pachete. Sintaxa de adăugare a unui modul sau pachet este formată din cuvântul cheie “import”, apoi între acolade se găsesc toate pachetele sau modulele de care este nevoie și pot fi găsite la calea dată. După ce enumerăm pachetele sau modulele dorite se găsește cuvântul cheie “from” urmat de calea unde se găsesc pachetele respective.

import { Component } from '@angular/core';

În continuare se află codul sursă care urmează să fie executat. Acesta poate cuprinde declarări de funcții și variabile, apeluri de funcții sau modificări ale variabilelor.

Structura programului mai poate conține alte fișiere “\*.ts” care pot fi importate în fișierul “index.ts”. Aceste fișiere pot fi folosite pentru a împărți codul în module cu scopul de a rezulta un cod mai curat. Pentru a putea folosi o clasă sau o funcție din alt fișier se folosește cuvântul cheie export anterior definiției acesteia.

export class AppRoutingModule { }

### 2.4.3 Variabile. Constante. Expresii

În TypeScript există două cuvinte cheie diferite folosite pentru declararea variabilelor. Declarația variabilelor folosind “var” este modul tradițional provenit din JavaScript.

var a = 10;

Folosind acest mod de declarare a variabilei în interiorul unei funcții face ca variabila să fie vizibilă în tot blocul de cod din funcție.

function sum(a: number, b: number) {

if (a + b > 10) {

var x = 10;

}

return x;

}

Acest lucru poate fi o problemă, deoarece declararea mai multor variabile cu același nume folosind cuvântul cheie “var” nu generează o eroare. Prin urmare, pot apărea o mulțime de greșeli de tip de dată sau de rescriere a unei variabile.

Introducerea cuvântului cheie “let” a avut ca și scop rezolvarea problemelor avute în momentul folosirii cuvântului cheie “var”. Sintaxa de definire a unei variabile folosind “let” este la fel ca și cea pentru definirea unei variabile folosind “var”.

let a = 10;

Variabilele declarate cu “let”, comparativ cu cele declarate cu “var”, sunt vizibile doar în interiorul blocului în care au fost declarate. Astfel, folosirea variabilei în afara blocului în care a fost declarată o să genereze o eroare.

function sum(a: number, b: number) {

if (a + b > 10) {

let x = 10;

}

return x; //eroare de existență

}

Acest mod de declarare a variabilelor elimină și posibilitatea anterioară de a declara o variabilă cu același nume de mai multe ori, generând o eroare în acest caz. Astfel, “let” elimină posibila rescriere a variabilelor din greșeală și erorile de tip de dată. De asemenea, o variabilă nu poate fi definită folosind cuvântul cheie “var” dacă a fost anterior definită folosind cuvântul cheie “let”.

Pentru a declara o variabilă constantă în TypeScript trebuie să folosim cuvântul cheie “const”. Acest mod de declarare a unei variabile este foarte asemănător cu declararea de tip “let”, doar că variabilele nu pot fi reasignate. În cazul obiectelor sau vectorilor definiți folosind “const” referința acestora nu poate fi schimbată, dar conținutul lor poate fi modificat.

const person = {

name: "Alice",

age: 30

};

person.age = 31;

Pentru a remedia această problemă TypeScript oferă posibilitatea de a seta membrii unui obiect ca readonly.

În TypeScript expresiile sunt fragmente de cod care produc o valoare. Cel mai simplu model de expresie disponibil în TypeScript este expresia de atribuire. Aceasta este folosită atunci când atribuim o valoare unei proprietăți. Apoi expresiile aritmetice care sunt folosite pentru a efectua operații matematice. Expresii logice, folosite pentru a evalua condiția logică și relațională.

Un tip de expresie mai special regăsit în TypeScript este expresia de definire a unui tip de funcție. Sunt similare cu expresiile lambda și sunt folosite pentru a îmbunătăți documentația codului și a spori siguranța tipului de date. (TypeScript, 2024)

type MathOperation = (a: number, b: number) => number;

const add: MathOperation = (a, b) => a + b;

## 2.5 Source control

### 2.5.1 Git

Git a fost creat de către Linus Torvalds și alți dezvoltatori cu scopul de a ușura dezvoltarea sistemului de operare Linux în anul 2005. Scopul principal al acestui sistem este de a menține controlul versiunilor fișierelor distribuite, dar și de a garanta integritatea datelor și viteza. Acesta reușește să aibă un sistem distribuit prin faptul că fiecare utilizator care lucrează la un proiect are o copie locală cu tot proiectul și branch-urile active, precum și istoricul versiunilor anterior dezvoltate.

Un branch este o copie a proiectului în care un dezvoltator poate face modificări în mod izolat pentru a experimenta sau a adăuga noi funcționalități. Aceste branch-uri pot fi apoi adăugate la branch-ul principal folosind “merge” fără a pierde istoricul modificărilor. În cazul în care mai mulți utilizatori au adăugat branch-urile proprii la branch-ul principal, iar ulterior un utilizator vrea să adauge branch-ul lui care nu conține noile modificări Git verifică dacă modificările acestui utilizator interferează cu noile modificări din branch-ul principal și crează un conflict în acest caz fără a mai adăuga branch-ul utilizatorului. Astfel utilizatorul trebuie să rezolve conflictul prin aducerea modificărilor din branch-ul principal în cel propriu și alegerea modificărilor pe care dorește să le mențină înainte de a încerca din nou funcția de “merge”, asigurânudu-se integritatea datelor.

Pentru a salva modificările efectuate asupra proiectului într-o nouă versiune utilizăm “commit”. Pentru a realiza un commit este necesar să adăugăm și un mesaj. Prin această comandă, modificările sunt salvate local urmând să se folosească comanda “push” pentru a salva toate modificările remote. Prin folosirea acestor comenzi, modificările sunt salvate în branch-ul în care se află utilizatorul.

În prezent, Git a devenit cel mai popular sistem de control al versiunii, distribuit cu peste 90% din dezvoltatori folosind-ul ca și sistem de control de versiune principal, fiind folosit în servicii precum GitHub, Bitbucket, SourceForge si GitLab. Acesta poate fi folosit fie prin interfețe grafice, fie de la linia de comandă, principalele comenzi fiind (GitHub I. , 2024) (Conservancy, 2020):

* git init – inițializează un repository nou
* git clone [URL] – copiază local un repozitory de la URL
* git add [file] – adaugă un fișier în directorul în care se lucrează
* git commit -m [commit message] – salvează modificările curente adăugate în istoricul repository-ului
* git push – trimite modificările locale într-un repository remote
* git pull – aduce modificările din repository-ul remote și execută comanda “merge”
* git merge – combină modificările dintr-un branch cu cele din alt branch

### 2.5.2 Fork

Fork este o aplicație pentru desktop care oferă o interfață grafică pentru Git, disponibilă pe sistemele de operare Windows și macOS. Aceasta oferă posibilitatea de a folosi comenzile de bază din git mult mai ușor, schimbarea rapidă între repository-uri, vizualizarea istoricului de commit-uri și a branch-urilor publicate în repository-ul remote.

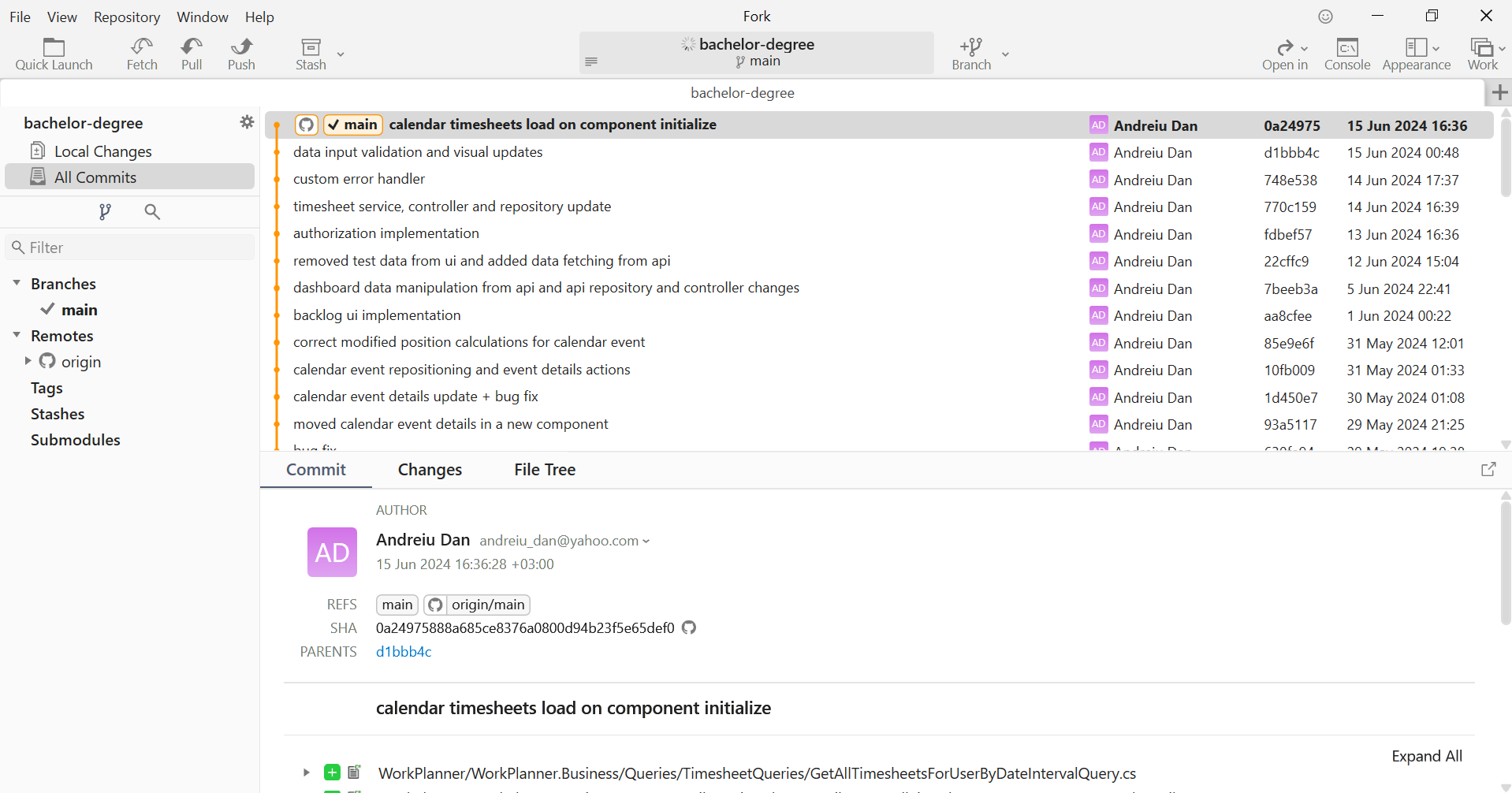


Figura 2.5.2.1 Interfață grafică Fork

Pe lângă caracteristicile principale enumerate, Fork oferă și un mod intuitiv și ușor de rezolvare a conflictelor de merge, recuperarea commit-urilor pierdute și funcția de rebase interactiv. (Fork, 2020)

### 2.5.3 GitHub

GitHub este o platformă care permite dezvoltatorilor de aplicații software să își găzduiască codul oferind controlul versiunilor și posibilitatea de colaborare cu alte persoane de oriunde. Acesta folosește software-ul Git pentru controlul versiunilor distribuit, de asemenea oferă și caracteristici precum accesul controlat la repository-uri prin setarea permisiunilor pentru colaboratori.

Serviciul GitHub, apărut în anul 2008, a fost dezvoltat de către Chris Wanstrath, P. J. Hyett, Tom Preston-Werner, și Scott Chacon folosind un framework pentru aplicații web server-side scris în limbajul de programare Ruby numit “Ruby on Rails”. Acesta este în prezent deținut de către Microsoft din anul 2018, dar operează independent.

O caracteristică principală oferită de GitHub este managementul de proiect, util mai ales în momentul colaborării cu mai multe persoane. Din această caracteristică face parte sistemul de “Pull Requests” care permite utilizatorilor să propună modificări asupra codului modificat de un dezvoltator înaine ca acesta să fie introdus în branch-ul principal al proiectului. O altă unealtă utilă de management de proiect este “Issues” prin care se pot identifica ușor funcționalitățile care trebuie să fie implementate sau bug-urile din cod. “Projects” oferă o simplificare în gestionarea sarcinilor și organizarea proiectelor prin opțiunea de a grupa sarcinile din proiect în funcție de stagiul în care se află, crearea unui plan pentru diferite stagii ale proiectului și vizualizarea unui istoric cu sarcinile, momentul începerii sarcinilor, momentul finalizării sarcinilor și momentele cheie în dezvoltarea proiectului. De asemenea, GitHub oferă și un spațiu pentru documentarea proiectului numit “Wiki”.

Vizualizarea și navigarea prin cod este una facilă și intuitivă, acesta oferind unelte pentru analiză de cod și adăugarea de comentarii la linia dorită, vizualizarea grafică a branch-urilor și a commit-urilor, precum și vizualizarea diferențelor între două versiuni ale programului.

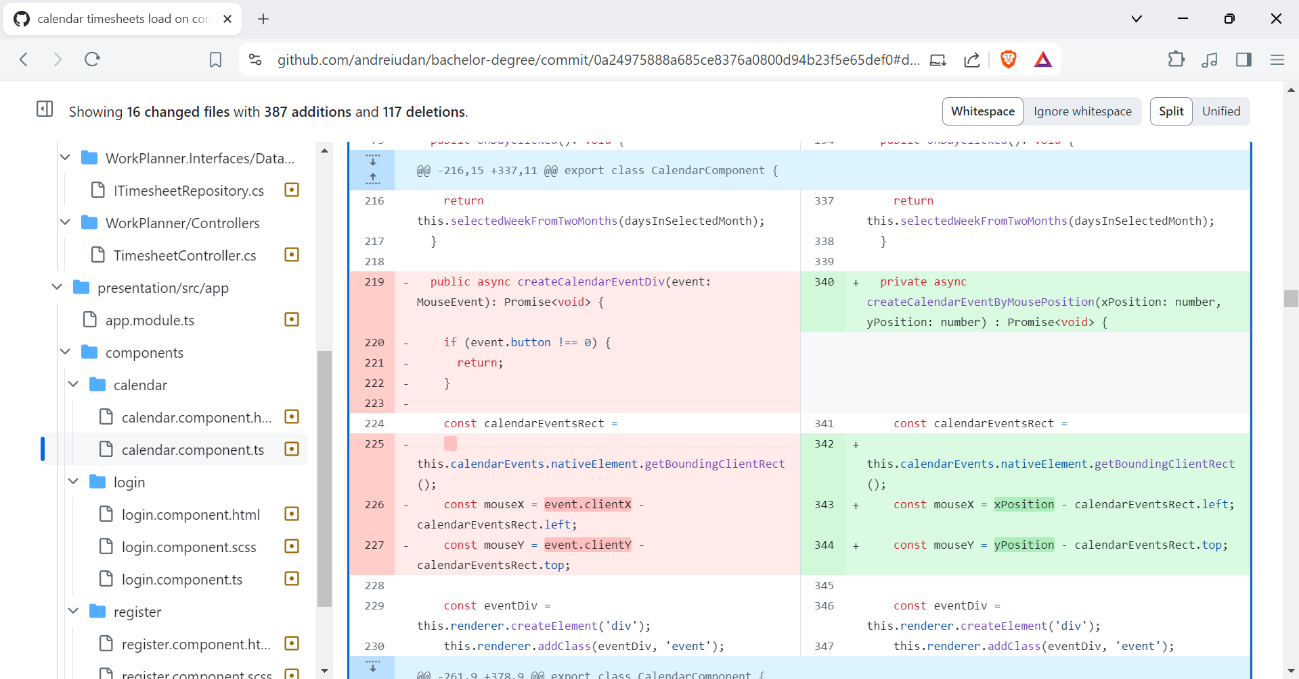


Figura 2.5.3.1 Vizualizarea modificărilor în GitHub

O altă caracteristică a GitHub este integrarea și automatizarea realizată prin GitHub Actions. Caracteristicile principale a acestora sunt automatizările complete, evenimente declanșatoare, job-uri și etape. Aceste job-uri conțin mai multe etape care execută acțiuni individuale precum un script. Astfel, GitHub Actions poate fi folosit pentru integrare continuă dacă utilizatorul dorește rularea testelor unitare la fiecare commit, pentru livrare continuă dacă se dorește automatizarea procesului de deploy sau pentru automatizarea proceselor repetitive. (GitHub, 2024)

## 2.6 SQLite

### 2.6.1 Introducere

În anul 2000, Dwayne Richard Hipp a lansat prima versiune de SQLite cu stocare bazată pe GNU Database Manager. Aceasta este o librărie scrisă în limbajul de programare C și este în prezent inclusă în marea majoritate a sistemelor de operare.

De-a lungul anilor SQLite a primit o mulțime de actualizări, unul din cele notabile fiind lansarea versiunii SQLite 2, moment în care a fost schimbat modul de stocare. Apoi, din anul 2004, a fost lansată versiunea SQLite 3.0 care a adus multe îmbunătățiri și care încă este de versiunea principală. Această versiune a primit la rândul ei îmbunătățiri în mod constant, iar în anul 2018 această librărie a adoptat o conduită de cod bazată pe etica codului. În prezent, cea mai recentă versiune de SQLite este 3.46.0 lansată în anul 2024 care a venit cu multe optimizări și noi trăsături. (SQLite, 2024)

### 2.6.2 Forme normale

Pentru a putea proiecta o bază de date relațională corectă este nevoie să aplicăm un set de reguli de proiectare și concepte care ajută la eliminarea redundanțelor și asigură integritatea datelor. Un concept esențial este setul de forme normale. Fiecare formă normală vine cu propriile sale criterii și mărește integritatea datelor, dar și complexitatea interogărilor.

**Prima formă normală**

Această formă este realizată în momentul în care câmpurile unui tabel conțin doar o singură valoare și fiecare intrare în tabelă este unică. Încălcarea acestei forme normale poate avea loc, spre execmplu, atunci când un câmp conține un vector de elemente.

**A doua formă normală**

Pentru a îndeplini a doua formă normală un tabel trebuie în primul rând să îndeplinească condițiile primei forme normale. Apoi trebuie să se asigure că toate coloanele care nu fac parte din cheia primară depind de totalitatea acesteia.

**A treia formă normală**

Dacă într-o tabelă, toate coloanele non-cheie depind direct de cheia primară și nimic altceva, iar aceasta îndeplinește cerințele pentru a doua formă normală, atunci acea tabelă este în a treia formă normală.

**Forma normală Boyce-Codd**

Se mai numește și forma normală 3.5. Această formă normală este o remodelare mai restrictivă a formei normale trei care ajută la eliminarea anumitor redundanțe bazate pe dependințe funcționale. De asemenea, tabela trebuie să îndeplinească condițiile pentru a treia formă normală ca să poată fi aplicată.

**A patra formă normală**

Această formă normală rezolvă redundațele apărute în tabelă prin independențele dintre câmpuri care formează cheia primară. Soluția pentru rezolvarea acestei probleme este descompunerea tabelului principal în mai multe tabele, separând câmpurile independente. Pentru a putea aplica aceasă formă normală este necesară îndeplinirea condițiilor formei normale Boyce-Codd. (GeeksforGeeks, 2024)

## 2.7 Figma Mockups

Figma este un instrument folosit pentru crearea design-ului interfeței grafice și a experienței utilizatorului. Acesta a fost lansat în anul 2015 ca și un program exclusiv bazat pe invitații. Apoi, după aproape un an, Figma a fost lansat public pentru prima dată. De atunci Figma primește îmbunătățiri în mod constant care ușurează dezvoltarea colaborativă de proiecte precum și implementarea acestora.

O caracteristică importantă adusă acestui instrument este modul pentru dezvoltatori, lansat în anul 2023. Acest mod traduce schița creată a aplicației în cod sursă în mod rapid. Momentan acest mod este gratuit doar pentru testarea acestuia pe un număr limitat de pagini și proiecte.

O altă calitate importantă este modelul de creare a prototipurilor. Acest mod poate fi folosit pentru a îmbunătăți experiența utilizatorului cu design-ul creat. Aici, dezvoltatorul design-ului poate adăuga navigare între pagini și animații pentru a simula navigarea în proiectul final după implementare. Crearea acestui prototip facilitează și comunicarea dintre dezvoltatorul design-ului interfeței grafice și dezvoltatorul aplicației propriu-zise.

Figma pune la dispoziție și un istoric al modificărilor aduse fișierelor, astfel permițând revenirea la versiuni anterioare dacă este necesar. Această caracteristică este necesară mai ales atunci când proiectul este dezvoltat de mai multe persoane.

Crearea unui proiect în echipă folosind Figma a fost ușurat de către dezvoltatorii aplicației printr-o mulțime de caracteristici. Figma oferă abilitatea mai multor designeri de a lucra simultan pe același, vizualizând modificările în timp real. De asemenea este ofertită posibilitatea de a adăuga comentarii direct pe design folosind sticky notes. Această calitate facilitează comunicarea și revizuirea între membrii echipei. (Wikipedia, Figma, 2024) (Figma, 2024)

# 3 Descrierea formală a aplicației

Acest capitol prezintă o serie de aspecte pe care le-am luat în considerare înainte de a începe dezvoltarea propriu-zisă a aplicației.

## 3.1 Actori

Aplicația include trei tipuri de actori.

**Vizitator**

Acest actor este o persoană care nu este autentificată sau nu deține un cont. El poate accesa paginile de întâmpinare, pagina de autentificare sau pagina de înregistrare.

**Utilizator**

Acesta este actorul principal al aplicației. Este reprezentat de o persoană care s-a înregistrat și autentificat în aplicație. Utilizatorii au acces la toate funcțiile și paginile puse la dispoziție și pot crea, modifica sau șterge date din baza de date.

**Sistem**

Acest actor este reprezentat de componentele automate ale aplicației care interacționează cu utilizatorii. Acesta se ocupă de trimiterea automată a unui mail care conține un link de verificare atunci când un utilizator a creat un nou cont.

## 3.2 Mockups

Înainte de a începe implementarea aplicației am realizat o serie de schițe pentru o parte din paginile principale ale aplicației care prezintă aspectul dorit al aplicației. Aceste schițe, precum și un prototip care simulează experiența utilizatorului cu aplicația au fost realizate folosind Figma.

### 3.2.1 Landing

Este pagina de bun venit a aplicației. Aici îi sunt oferite câteva beneficii utilizatorului care vizitează pagina pentru prima dată.

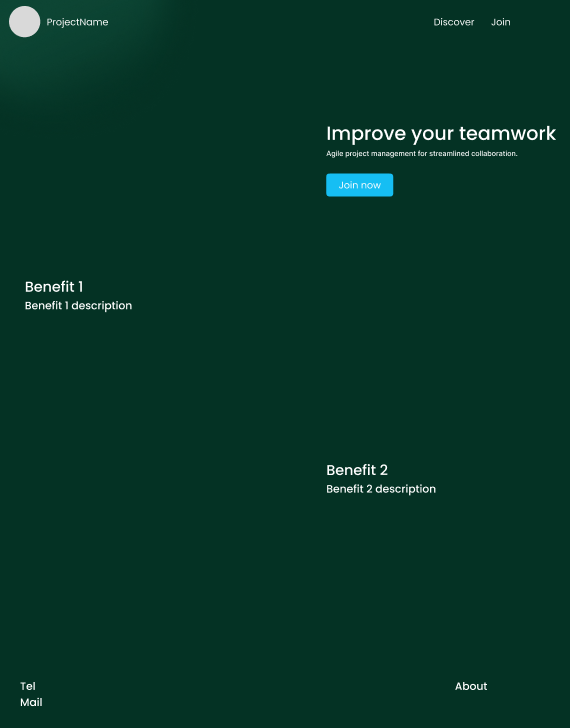


Figura 3.2.1.1 Mockup pagina de landing

### 3.2.2 Login

Este pagina din care utilizatorul se poate autentifica.

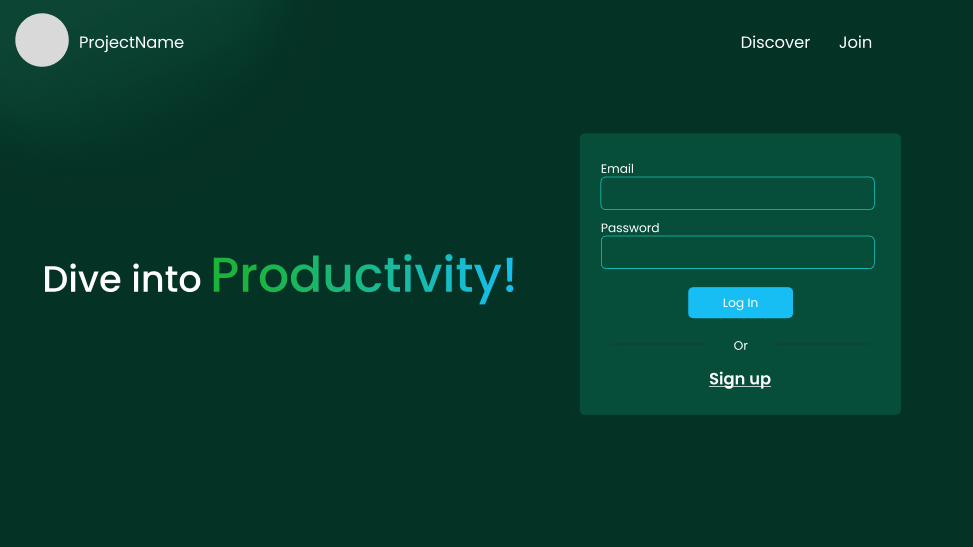


Figura 3.2.2.1 Mockup pagina de login

### 3.2.3 Registration

Este pagina din intermediul căreia utilizatorul poate să își creeze un nou cont.

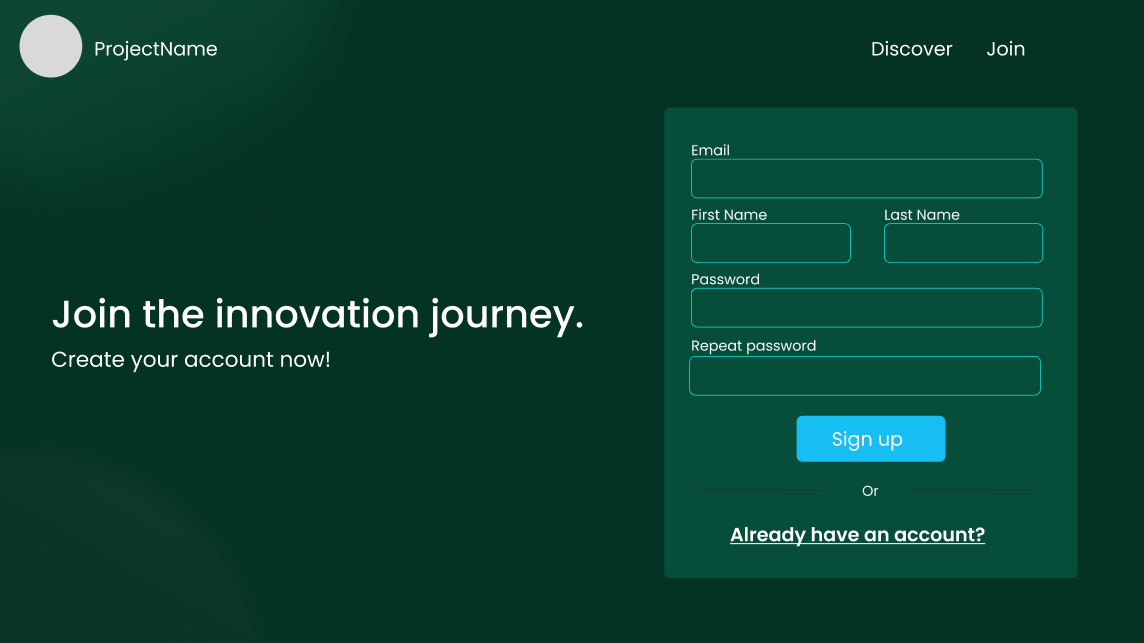


Figura 3.2.3.1 Mockup pagina de register

### 3.2.4 Dashboard

Este pagina unde utilizatorul poate vedea informații despre proiectul selectat și sprint-ul activ, precum și comparații între sprint-uri.

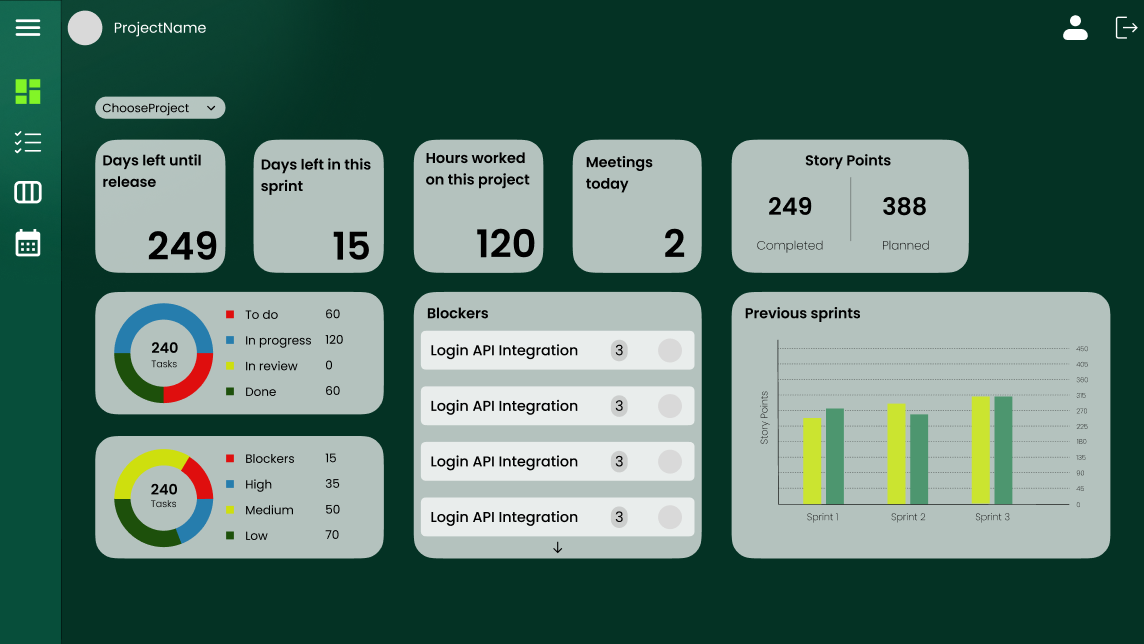


Figura 3.2.4.1 Mockup pagina de dashboard

### 3.2.5 Tasks board

Este pagina unde sunt afișate sarcinile din sprint-ul activ, grupate în funcție de stagiul în care se află.

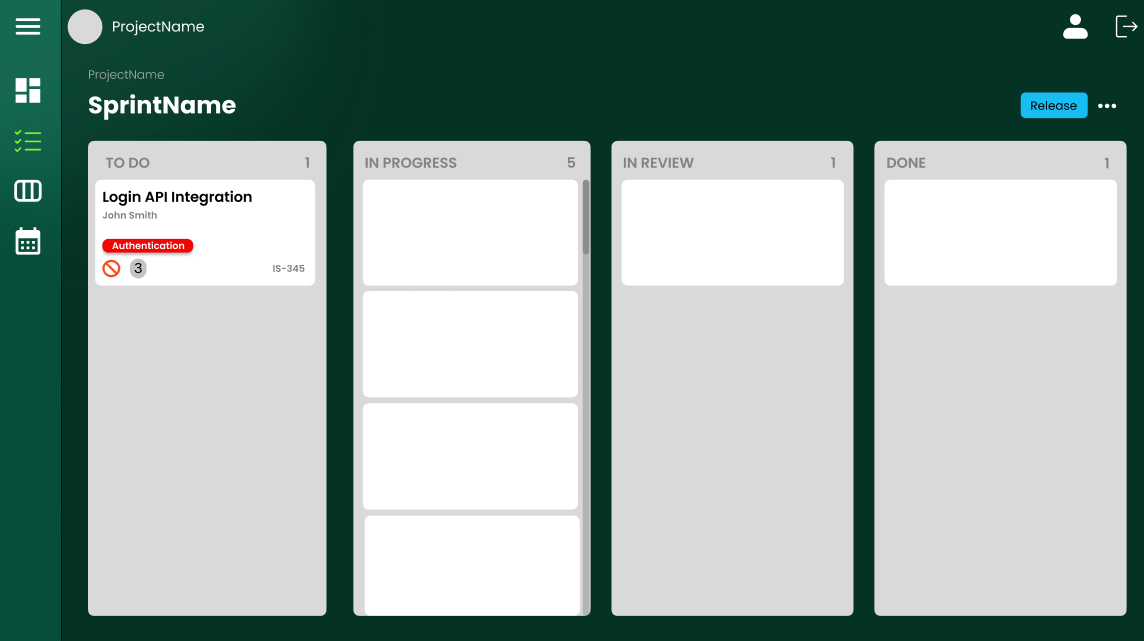


Figura 3.2.5.1 Mockup pagina task board

### 3.2.6 Task

Aici utilizatorul poate vedea toate detaliile unei sarcini împreună cu sub sarcinile acesteia și status-ul lor.



Figura 3.2.6.1 Mockup pagina unui task

### 3.2.7 Backlog

Este pagina în care se găsesc sprint-urile unui proiect, precum și backlog-ul acestuia împreună cu task-urile din acestea.

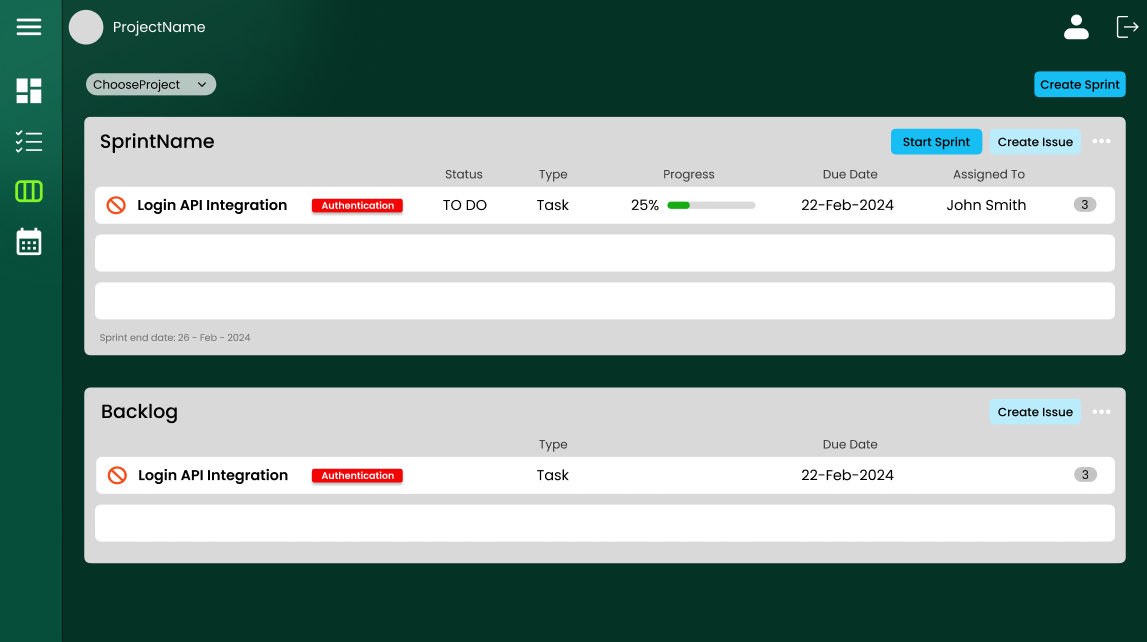


Figura 3.2.7.1 Mockup pagina backlog

### 3.2.8 Calendar

Aici este prezentat calendarul în care utilizatorul își poate trece pontajul.

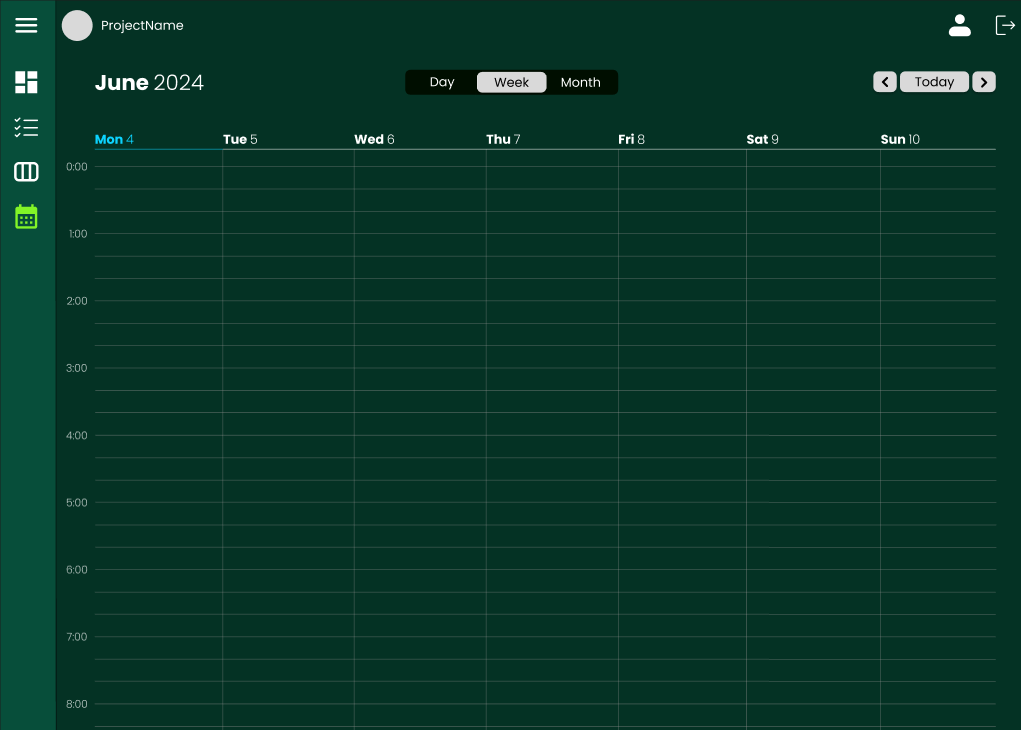


Figura 3.2.8.1 Mockup pagina pentru calendar

## 3.3 Arhitectura aplicației

Pentru arhitectura aplicației am urmărit modelul de arhitectură curată propus de Robert C Martin. Acest tip de arhitectură se focusează pe separarea clară a responsabilităților pentru a ușura testarea, mentenanța și scalabilitatea aplicațiilor. Principiul de bază al acestei arhitecturi este independența față de sursele exterioare precum interfața utilizatorului, baza de date sau framework-uri folosite. (Martin, 2012)

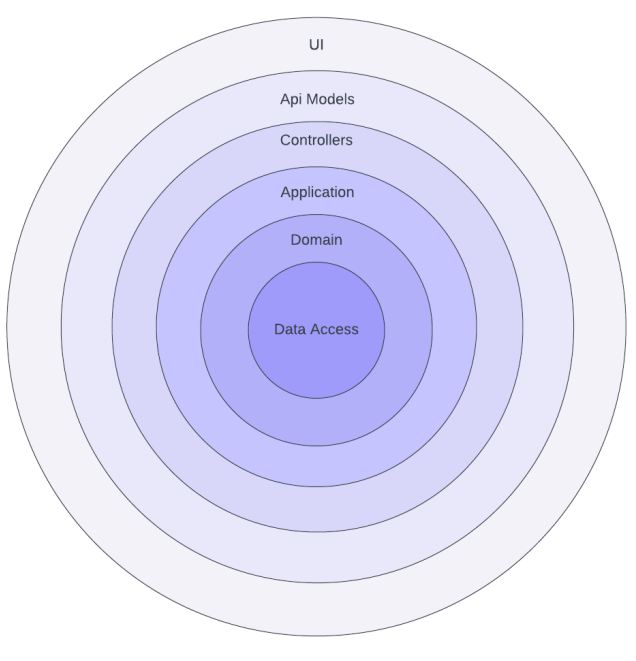


Figura 3.4.1 Model de arhitectură curată tip ceapă

Arhitectura aplicației dezvoltată este formată în ansamblu din două aplicații care comunică între ele prin protocolul HTTP. Una din aplicații conține pagina web cu care interacționează utilizatorul și pe care sunt afișate datele. Iar cea de a doua aplicație este un REST API care primește cereri de la aplicația web și le interpretează. Acest API deține legătura cu baza de date și modelează toate datele cerute sau primite de la pagina web.

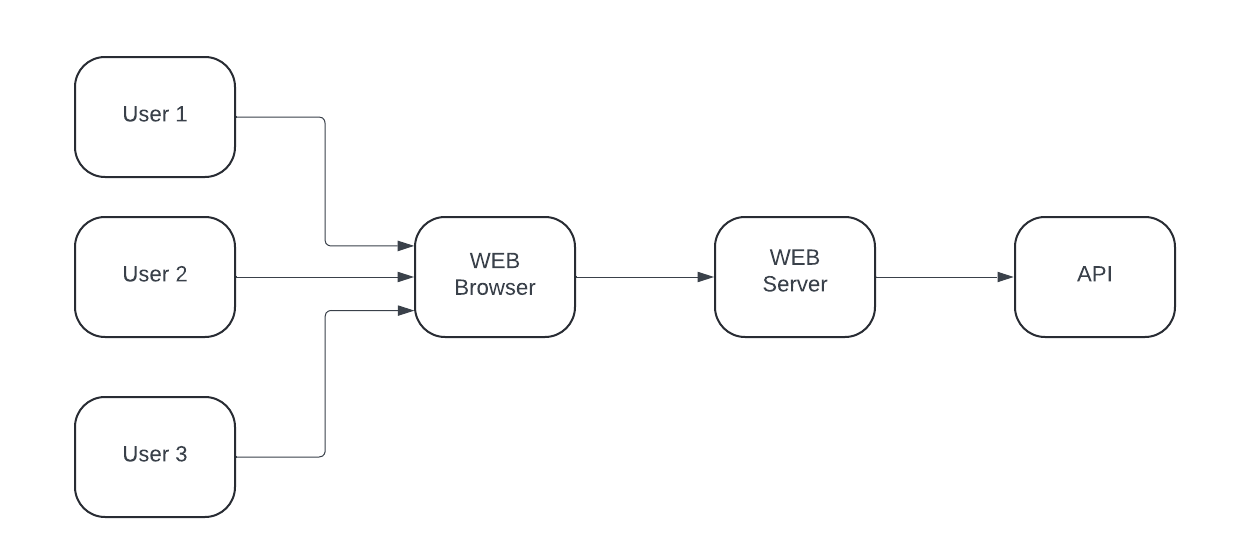


Figura 3.4.2 Arhitectura aplicației dezvoltate

API-ul este format la rândul lui din următoarea arhitectură.

**Nivelul Api**

Acesta este nivelul cel mai din exterior al arhitecturii API-ului. Este punctul de legătură dintre aplicația web și baza de date. Aici datele primite sunt transmise către nivelul Business care rezolvă cererea efectutată. De asemenea, prin acest nivel datele sunt trimise înapoi la aplicația web.

În acest nivel se realizează și o serie de validări a cererilor HTTP și a datelor primite. Dacă validarea a avut loc cu succes, datele sunt trimise către controller. În caz contrar, este trimisă o listă cu erorile care au împiedicat validarea datelor.

**Nivelul Business**

Aici sunt procesate informațiile valide primite de la pagina web, precum și datele aduse din baza de date pentru a fi trimise către pagina web.

Pentru a ușura dezvoltarea și a evita legătura strânsă dintre clase am folosit modelul de proiectare ”mediator”. Astfel, legătura dintre cererile din nivelul Api și logica aplicației care rezolvă acea cerere este susținută de mediator. Acesta primește un obiect care implementeză ”IRequest<T>” și apelează un obiect care implementează ”IRequestHandler<T>” care conține logica necesară pentru a rezolva cererea. Apoi, este returnat rezultatul de tipul ”T”.

**Nivelul Domain**

Acest nivel conține entitățile folosite pentru crearea bazei de date, modelele de transfer a datelor între aplicații, precum și modelele email-urlior care pot fi trimise. În acest nivel se mai găsesc și diverse modele ale configurațiilor găsite în fișierul ”appsettings.json”, ale sistemelor aplicației și a legăturii cu aplicația web.

**Nivelul Interfaces**

În acest nivel se găsesc toate interfețele din aplicație. Acesta a fost creat pentru a facilita implementarea modelului de proiectare ”injecție de dependințe”. Prin folosirea acestor modele, obiectele necesare unei componente sunt create de către framework-ul de injecție a dependințelor. Astfel, componentele nu trebuie să își creeze propriile instanțe.

De asemenea, pentru a decupla legătura strânsă între clase, a fost folosit modelul de proiectare ”inversiunea injecției de depentințe”. Acest model se bazează pe declarearea obiectelor în componente de tipul interfeței obiectului dorit.

Astfel, acest nivel exclude necesitatea de a avea legături între toate nivelurile aplicației și facilitează testarea aplicației prin testele unitare.

**Nivelul Notification**

În cadrul acestui nivel se află serviciul de trimitere a email-urilor. Acesta are rolul de a se conecta la un server care folosește protocolul simplu de transfer de corespondență (SMTP). Tot aici, email-ul care urmează să fie trimis este contruit adăugând emițătorul, destinatarul, subiectul și conținutul.

**Nivelul Data Access**

Acesta este cel mai imbricat nivel. El nu are informații despre nivelurile din exteriorul său și este folosit pentru a accesa baza de date. Aici se găsesc toate migrările efectuate către baza de date, toate repository-urile, definirea contextului bazei de date și unitatea de muncă.

Această unitate de muncă este un model de proiectare care conține o instanță a fiecărui repository și este folosită pentru a asigura integritatea datelor. Astfel, când folosim această unitate de muncă ne asigurăm că folosim același context în cadrul fiecărui repository.

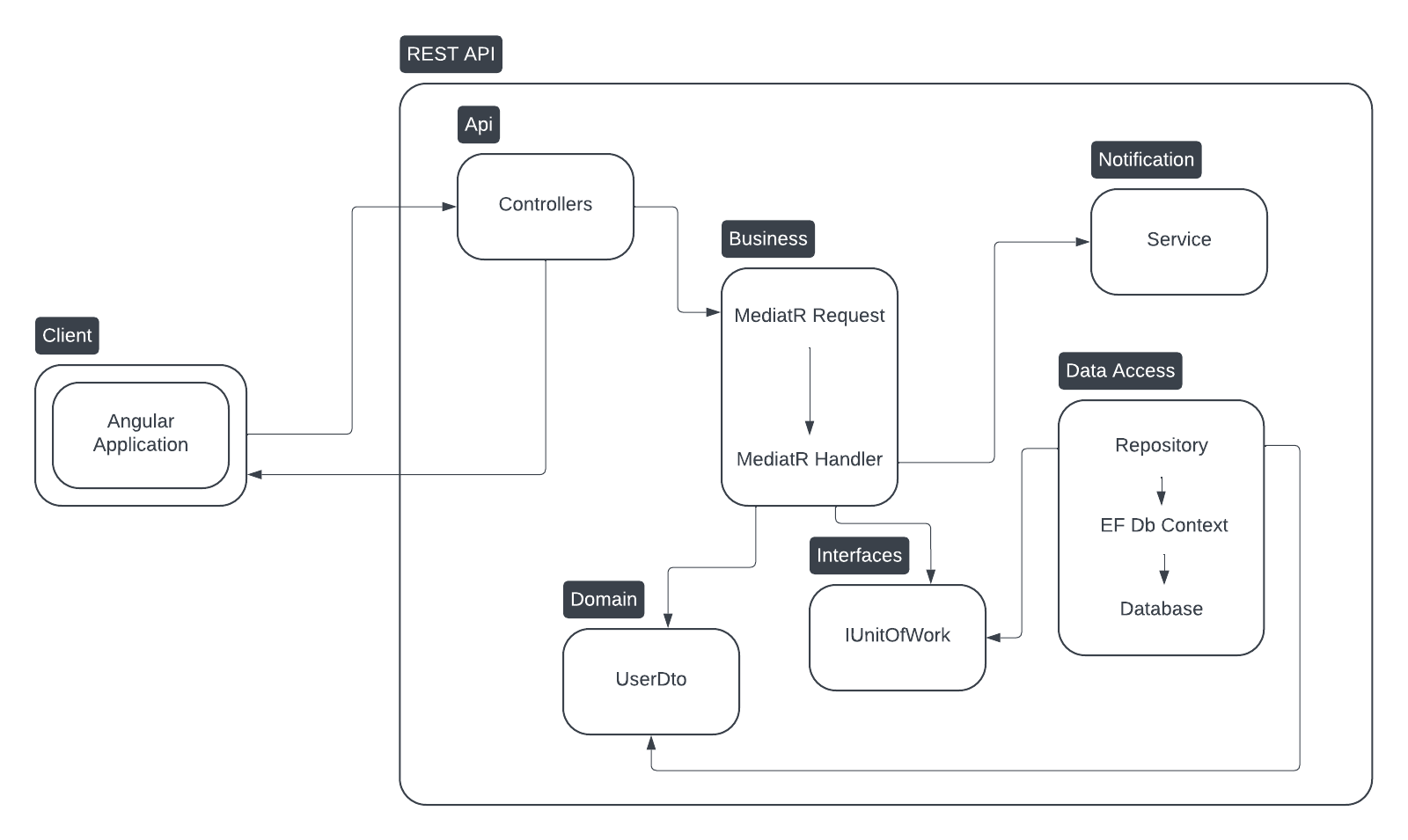


Figura 3.4.3 Arhitectura API-ului

Cea de a doua aplicație, folosită pentru aplicația web este formată din următoarea arhitectură.

Modulul Aplicație este folosit pentru a inițializa aplicația. Aici se importă modulele necesare, se declară compontele și se definesc rutele pentru componente.

Componentele sunt folosite pentru a crea paginile web din aplicație. Acestea conțin propriile schițe ale paginii, stilurile folosite în pagină și propria logică.

Serviciile sunt obiecte care sunt instanțate o singură dată. Acestea conțin logica aplicației și sunt folosite pentru a efectua cererile către API și a recupera datele primite.

Interceptoarele sunt folosite pentru a prinde cererile de tip HTTP înainte de a ieși din aplicație și pentru a le modifica conținutul.

Directiva a fost folosită pentru a adăuga o altă componentă în interiorul unui element de tip HTTP din altă componentă în mod dinamic.

Gărzile sunt folosite pentru a verifica dacă un utilizator este autentificat atunci când accesează o rută.

# 4 Detalii de implementare

## 4.1 Descrierea bazei de date

Baza de date folosită conține în momentul de față un număr de 7 tabele folosite pentru a salva datele de care avem nevoie în buna desfășurare a aplicației, precum și un tabel adițional generat de către Entity Framework Core. Acest tabel stochează informații despre istoricul migrațiilor care au fost aplicate către baza de date, iar fiecare intrare conține un identificator unic și versiunea utilizată pentru migrare.

Tabelul “Users” este folosit pentru a salva utilizatorii care s-au înregistrat în aplicație. Acest tabel conține următoarele câmpuri:

* Id (Text, Cheie primară) – aceasta este generat automat la inserarea utilizatorului în tabel folosind un tip de date numit “Guid” format din 32 de caractere
* Email (Text) – salvează adresa de email a utilizatorului
* FirstName (Text) – stochează prenumele utilizatorului
* LastName (Text) – stochează numele de familie al utilizatorului
* HashedPassword (Text) – salvează valoarea hash generată în urma criptării parolei utilizatorului
* Salt (Text) – este o valoare generată în codul sursă, folosită pentru a îmbunătăți criptarea parolei utilizatorului
* RegistrationTime (Text) – salvează data și ora la care utilizatorul a efectuat înregistrarea
* Username (Text) – stochează numele de utilizator
* Verified (Integer) – este folosit pentru a verifica dacă utilizatorul a accesat link-ul de activare după crearea contului

Tabelul “Timesheets” memorează pontajul efectuat de către utilizator și conține următoarele câmpuri:

* Id (Text, Cheie primară) – este generat automat la inserarea de către utilizator a unui nou pontaj în tabel folosind un tip de date numit “Guid” format din 32 de caractere
* AccountId (Text, Cheie externă) – provine din tabelul “Users” și este folosit pentru a cunoaște utilizatorul care a efectuat pontajul respectiv
* Date (Text) – stochează data pontajului
* StartTime (Text) – salvează timpul de la care începe pontajul în ziua respectivă
* EndTime (Text) – memorează timpul până la care pontajul este realizat în acea zi

Conținutul tabelulului “Projects” cuprinde totalitatea proiectelor create de utilizatori. Acesta cuprinde următoarele câmpuri:

* Id (Text, Cheie primară) – este generat automat la adăugarea unui nou proiect în tabel folosind un tip de date numit “Guid” format din 32 de caractere
* CreatorId (Text, Cheie externă) – provine din tabelul “Users” și este folosit pentru a cunoaște utilizatorul care a creat proiectul respectiv
* Description (Text) – memorează descrierea proiectului
* DueDate (Text) – stochează data și timpul la care se dorește finalizarea proiectului
* Name (Text) – salvează numele proiectului
* StartDate (Text) – este generat automat la inserarea unui nou proiect în tabelă cu data și timpul actual relativ la ora universală coordonată

Tabela “Backlogs” cuprinde totalitatea sarcinilor create care nu au fost atribuite unui sprint și este folosit pentru a avea o legătură între sarcini și proiecte în absența sprint-urilor. În această tabelă se regăsesc următoarele câmpuri:

* Id (Text, Cheie primară) – este generat automat la adăugarea unui nou backlog în tabel folosind un tip de date numit “Guid” format din 32 de caractere
* ProjectId (Text, Cheie externă) – provinde din tabelul “Project” și este folosit pentru a cunoaște proiectul de care aparține backlog-ul respectiv

Tabela “Sprints” conține toate sprint-urile create. Aceasta este formată din următoarele câmpuri:

* Id (Text, Cheie primară) – este generat automat atunci când un nou sprint este adăugat în tabel folosind un tip de date numit “Guid” format din 32 de caractere
* CreatorId (Text, Cheie externă) – provine din tabela “Users” și este utilizat pentru a cunoaște urilizatorul care a creat sprint-ul respectiv
* DueDate (Text) – stochează data și ora la care este plănuit ca un sprint să fie finalizat
* Name (Text) – memorează numele sprint-ului
* ProjectId (Text, Cheie externă) – provine din tabelul “Projects” și este necesar pentru a cunoaște proiectul în cadrul căruia a fost creat sprint-ul
* StartDate (Text) – este generat automat la inserarea unui nou sprint în tabelă cu data și timpul actual relativ la ora universală coordonată
* Active (Integer) – este folosit pentru a monitoriza dacă sprint-ul este activ
* Released (Integer) – este folosit pentru a monitoriza dacă sprint-ul a fost finalizat

Tabela “Tasks” este folosită pentru a memora sarcinile create. Aceasă tabelă conține următoarele câmpuri:

* Id (Text, Cheie primară) – este generat automat la inserarea unei noi sarcini în tabel, folosind tipul de date “Guid”
* AssigneeId (Text, Cheie externă) – provine din tabela “Users” și este folosit pentru a cunoaște utilizatorul care are sarcina respectivă întrebuințată
* BacklogId (Text, Cheie externă) – provine din tabela “Backlogs” și este prezent doar atunci când cheia externă “SprintId” nu este prezentă
* CreatorId (Text, Cheie externă) – provine din tabela “Users” și identifică utilizatorul care a creat sarcina respectivă
* SprintId (Text, Cheie Externă) – este prezent doar atunci când cheia externă “BacklogId” nu este prezentă. Provine din tabela “Sprints” și este folosit pentru a cunoaște sprint-ul în care se dorește a fi rezolvată sarcina
* Description (Text) – conține descrierea sarcinii
* DueDate (Text) – stochează data și timpul la care se dorește finalizarea sarcinii
* Label (Text) – este folosit pentru a ușura gruparea sarcinilor
* Name (Text) – memorează numele sarcinii
* Priority (Integer) – memoreză valoarea relativă a priorității din enumerarea definită în aplicație
* StartDate (Text) – este generat automat la inserarea unei noi sarcini în tabelă cu data și timpul actual relativ la ora universală coordonată
* Status (Integer) – stochează valoarea relativă a statusului din enumerarea definită în aplicație
* StoryPoints (Integer) – memorează punctele date echipa de dezvoltare sarcinii respective
* Type (Integer) – salvează valoarea relativă a tipului de sarcină din enumerarea definită în aplicație

Tabela “Subtasks” conține punctele cheie ale unei sarcini pentru o dezvoltare și o urmărire a progresului mai ușoară. În această tabelă se regăsesc următoarele câmpuri:

* Id (Text, Cheie primară) – este generat automat la inserarea unei noi sub sarcini în tabel, folosind tipul de date “Guid”
* Done (Integer) – marchează dacă sub sarcina a fost completată
* Name – memorează numele sub sarcinii
* TaskId (Text, Cheie primară) – provine din tabelul “Tasks” și este necesar pentru a cunoaște sarcina din care face parte sub sarcina respectivă

Aplicând normalizarea bazei de date am ajuns la configurația de tabele menționată mai sus care cuprinde următoarele relații între tabele.

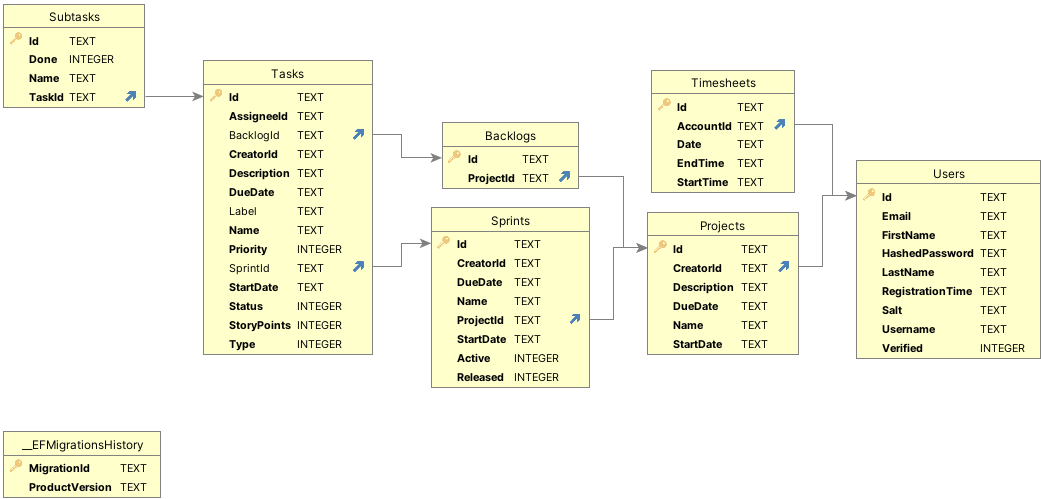


Figura 4.1.1 Relațiile dintre tabele

După cum putem observa în figura 4.1.1, în baza de date proiectată avem relații de one-to-many între majoritatea tabelelor, cu excepția relației de one-to-one dintre tabela “Backlogs” și tabela “Projects”.

### 4.1.2 Implementarea bazei de date

Pentru implementarea bazei de date am ales o abordare code-first, folosind framework-ul “Entity Framework Core” oferit de Microsoft împreună cu SQLite. Pentru a putea începe dezvoltarea a trebuit să configurez inițial proiectul. Acest lucru semnifică instalarea pachetelor specifice pentru Entity Framework Core folosind instrumentul integrat în Visual Studio numit “NuGet” pentru management de pachete. De aici a fost nevoie de instalarea pachetelor:

* Microsoft.EntityFrameworkCore – folosit pentru a crea contextul pentru baza de date
* Microsoft.EntityFrameworkCore.Design – utilizat pentru a facilita gestionarea migrărilor
* Microsoft.EntityFrameworkCore.Sqlite – necesar pentru a putea folosi SQLite pentru baza de date
* Microsoft.EntityFrameworkCore.Tools – pune la dispoziție comenzile necesare pentru modificarea bazei de date și a migrărilor

Pentru a funcționa, aceste pachete trebuie să fie instalate în nivelul de acces la date al aplicației, precum și în nivelul controller pentru a putea fi configurat.

Apoi au fost create modelele entităților care reprezintă tabelele bazei de date. Configurarea relațiilor între acestea este definită prin adăugarea unei proprietăți în fiecare entitate. De exemplu, pentru relația one-to-many dintre proiect și backlog avem o listă de backlog-uri în entitatea proiect și un id de proiect adnotat ca cheie externă din “Project” împreună cu un obiect de tipul proiect în entitatea backlog.

public class Backlog

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public Guid Id { get; set; }

[ForeignKey("Project")]

public Guid ProjectId { get; set; }

public Project Project { get; set; } = null!;

[Required]

public List<SprintTask> Tasks { get; set; } = new List<SprintTask>();

public override bool Equals(object? obj)

{

if (obj == null)

{

return false;

}

if (!(obj is Backlog))

{

return false;

}

return Equals(obj as Backlog);

}

private bool Equals(Backlog obj)

{

return this.Id.CompareTo(obj.Id) == 0 &&

this.ProjectId.CompareTo(obj.ProjectId) == 0 &&

this.Project.Equals(obj.Project) &&

this.Tasks == obj.Tasks;

}

public override int GetHashCode()

{

return HashCode.Combine(Id, ProjectId, Project, Tasks);

}

}

Fiecare entitate, pe lângă proprietățile specifice, conține un override specific acesteia pentru metodele “Equals” și “GetHashCode”. Aceste override-uri ne oferă o siguranță mai mare în compararea a două obiecte de același tip.

După ce entitățile au fost create, este configurat contextul bazei de date pentru fiecare relație dintre tabele, precum și pentru setarea automată a datei și timpului relativ la ora universală coordonată.

În final, a fost generată migrarea inițială și baza de date folosind comenzile “Add-Migration” și “Update-Database” în consola managerului de pachete.

## 4.2 Use Cases

### 4.2.1 Autentificare

Inițial, aplicația redirecționează utilizatorul către pagina de întâmpinare în care poate afla câteva detalii despre aplicație și poate naviga către alte pagini. Pagina de login poate fi accesată prin apăsarea butonului “Join” din bara de unelte din partea de sus a ecranului sau prin butonul “Join now” din cadrul paginii de întâmpinare.

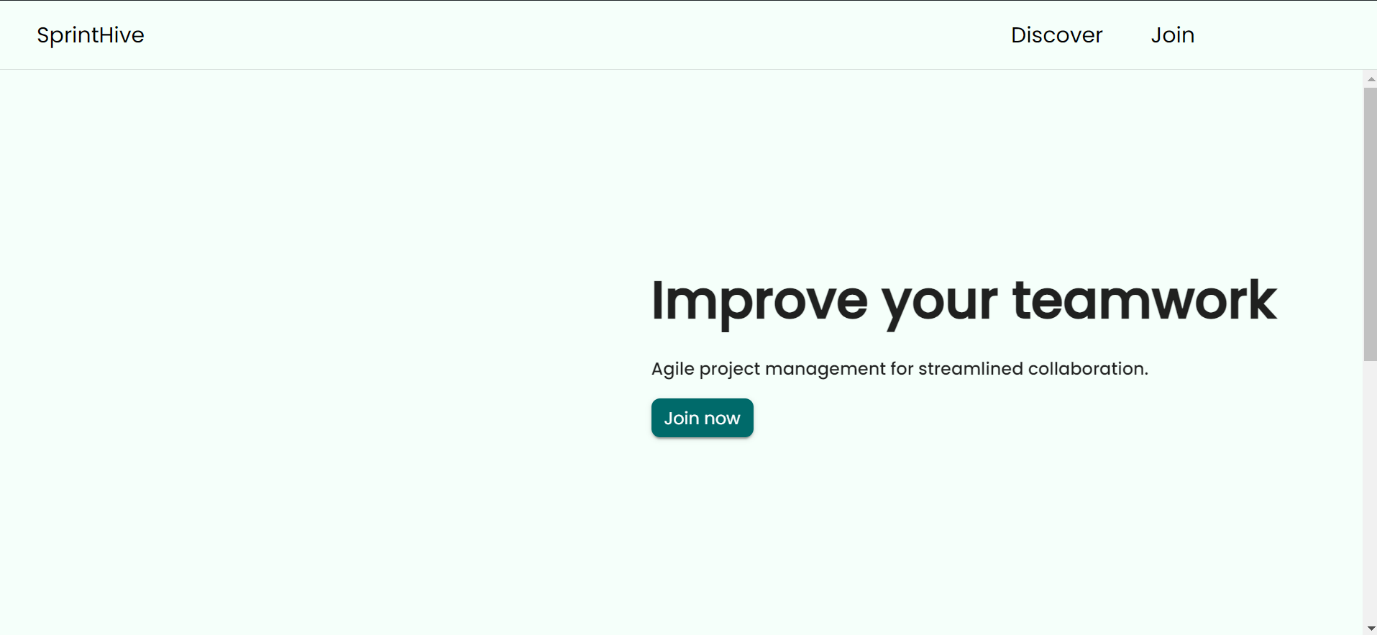


Figura 4.2.1.1 Pagina de landing

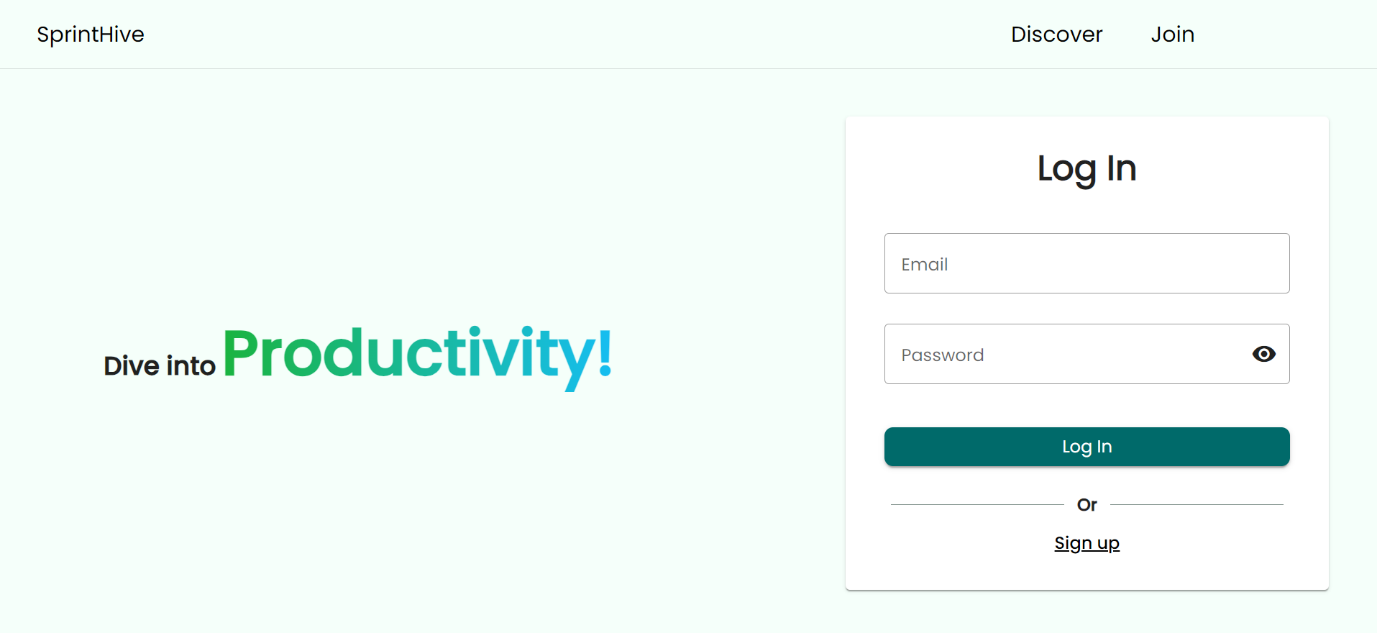


Figura 4.2.1.2 Pagina de log in

Din aceasă pagină, utilizatorul își poate introduce credențialele pentru a se autentifica sau poate naviga către pagina de înregistrare dacă nu deține un cont validat. Dacă utilizatorul a introdus o adresă de email care nu corespunde formei standard sau o parolă care nu îndeplinește condițiile, acesta nu poate încerca să se autentifice și apar următoarele mesaje de eroare.

Atunci când utilizatorul apasă butonul de log in este apelată următoarea metodă.

private login(user: UserAuthentication){

if(this.loginForm.valid){

let userWithEncodedPassword: UserAuthentication = {

email: this.email?.value,

password: btoa(this.password?.value),

};

this.authService.login(userWithEncodedPassword).subscribe(() => {

this.router.navigate(['/dashboard']);

});

}

}

Această metodă verifică dacă datele introduse sunt corecte. Apoi, se declară obiectul specific pentru autentificare și se setează datele. După care, este apelată metoda “login” din serviciul de autentificare. Această metodă este responsabilă cu crearea unei cereri HTTP către endpoint-ul specific din API. Dacă răspunsul primit din API nu este o eroare, utilizatorul este redirecționat către pagina de dashboard și token-ul JWT este salvat în memoria de sesiune.

[HttpPost("authenticate")]

[ServiceFilter(typeof(AuthenticationExceptionFilter))]

public async Task<IActionResult> Authenticate([FromBody] UserAuthenticationDto user)

{

UserAuthenticationCommand request = new UserAuthenticationCommand(user);

string token = await mediator.Send(request);

return Ok(token);

}

Odată ajuns în endpoint, obiectul primit în corpul cererii este validat. Dacă a fost validat cu succes este creată cererea pentru mediator și se așteaptă finalizarea handler-ului specific. Apoi, token-ul JWT primit din mediator este trimis înapoi către aplicația web. Dacă obiectul nu a putut fi validat, este trimisă înapoi la aplicația web o eroare împreună cu descrierea proprietății care nu a putut fi validată.

Mediator face legătura între cerere și handler prin moștenirea unei interfețe de tipul ”IRequest<T>” pentru cerere și o interfață de tipul ”IRequestHandler<Cerere, T>” pentru handler.

public class UserAuthenticationCommand : IRequest<string>

{

public UserAuthenticationDto User { get; }

public UserAuthenticationCommand(UserAuthenticationDto user)

{

User = user ?? throw new ArgumentNullException(nameof(user));

}

}

public class UserAuthenticationHandler : IRequestHandler<UserAuthenticationCommand, string>

{

private readonly IUnitOfWork unitOfWork;

private readonly IPasswordHasher passwordHasher;

private readonly ITokenService tokenService;

public UserAuthenticationHandler(IUnitOfWork unitOfWork,

IPasswordHasher passwordHasher,

ITokenService tokenService)

{

this.unitOfWork = unitOfWork ?? throw new ArgumentNullException(nameof(unitOfWork));

this.passwordHasher = passwordHasher ?? throw new ArgumentNullException(nameof(passwordHasher));

this.tokenService = tokenService ?? throw new ArgumentNullException(nameof(tokenService));

}

public async Task<string> Handle(UserAuthenticationCommand request, CancellationToken cancellationToken)

{

User user = await unitOfWork.Users.FindAsync(u => u.Email == request.User.Email);

if (user is null)

{

throw new UserNotFoundException();

}

if(user.Verified == false)

{

throw new UserNotActivatedException();

}

if (!passwordHasher.VerifyPassword(request.User.Password, user.Salt, user.HashedPassword))

{

throw new InvalidPasswordException();

}

return tokenService.GenerateToken(user);

}

}

În cerere este setat obiectul primit din cererea HTTP, pentru a putea fi folosi în handler. În handler sunt injectate dependințele în constructor, iar apoi se execută logica pentru autentificare. În primul rând se caută utilizatorul în baza de date în funcție de email. Apoi, se verifică dacă acesta există, dacă contul este activat și dacă parola primită coincide cu cea din baza de date.

Pentru fiecare verificare a fost creată o excepție specială pentru a putea trimite înapoi către aplicația web, prin filtrul de servicii, mesaje specifice problemei apărute. Dacă nu au apărut probleme, este generat token-ul JWT specific utilizatorului care dorește să se autentifice și este returnat către endpoint, iar mai apoi către aplicația web.

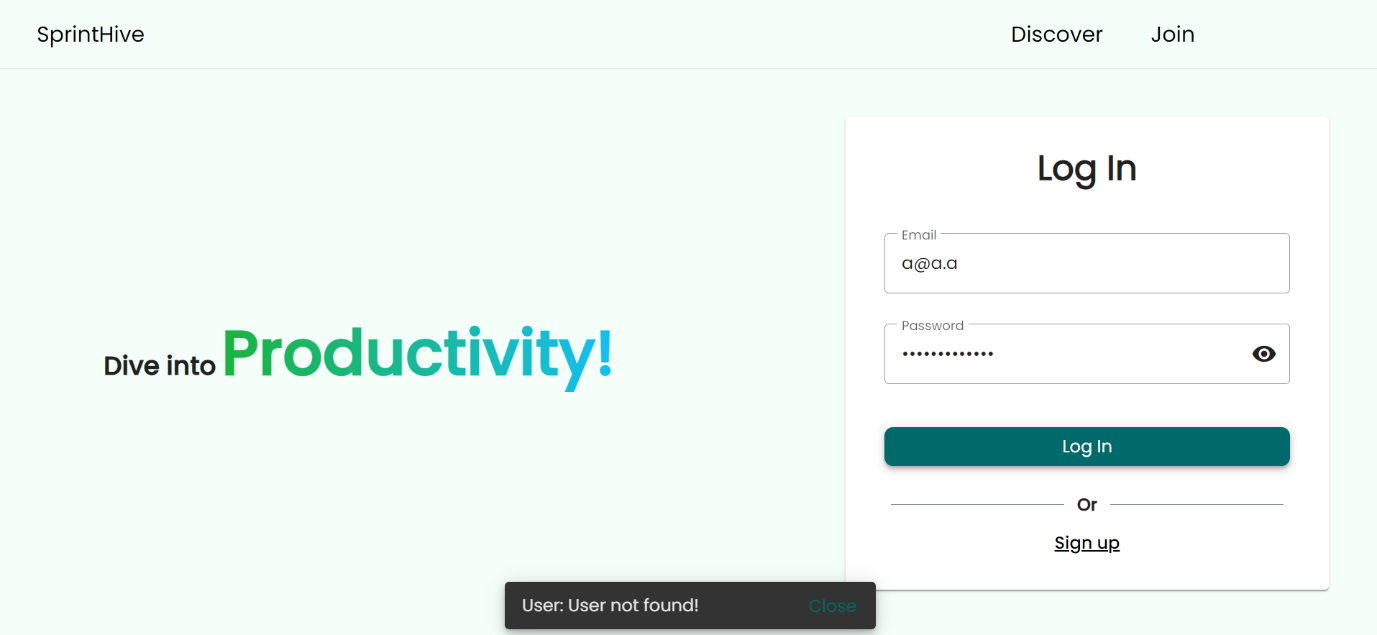


Figura 4.2.1.3 Pagina de login cu mesaj de eroare din API

Totodată, din această pagină, utilizatorul poate naviga înapoi la pagina de întâmpinare prin apăsarea pe titlul aplicației sau la o pagină de detalii.

### 4.2.2 Înregistrare

Dacă utilizatorul nu deține un cont validat sau dorește să creeze un nou cont, acesta poate naviga la pagina de înregistrare venind de la pagina de autentificare.

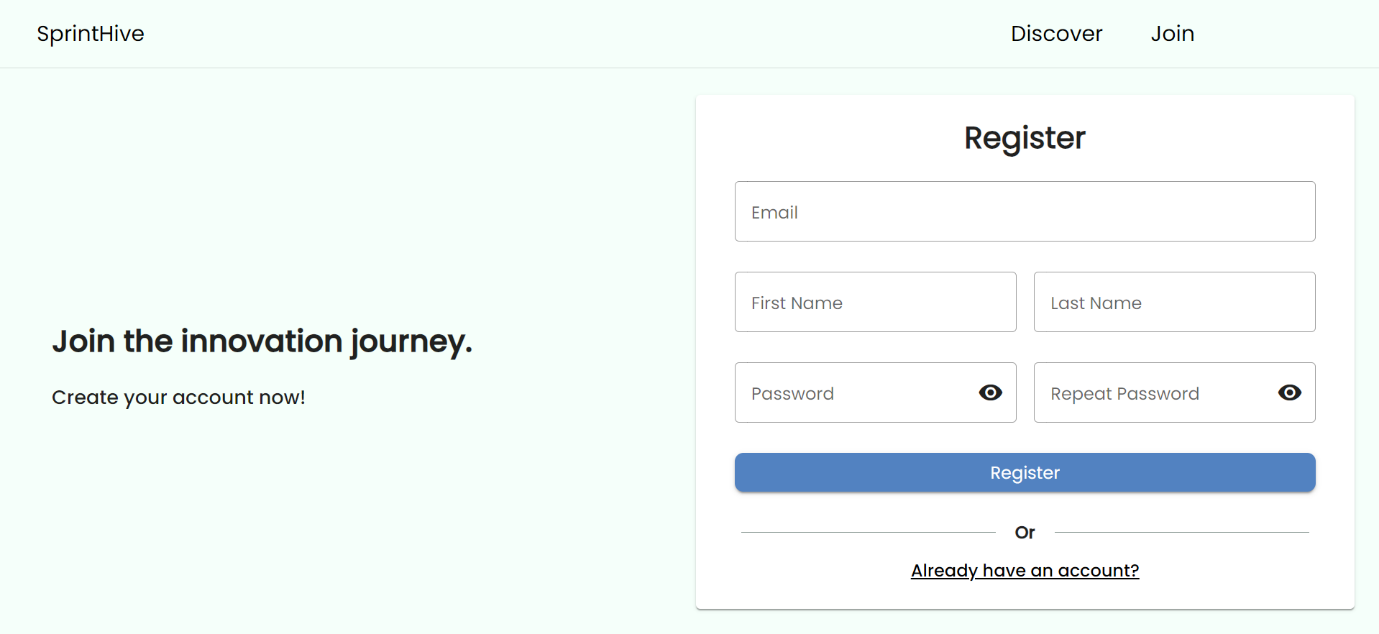


Figura 4.2.2.1 Pagina de înregistrare

Aici, utilizatorul trebuie să își introducă numele, prenumele, adresa de email, parola și să confirme parola ca să se poată înregisrtra. Dacă oricare dintre aceste date introduse nu corespund cu cerințele specificate o să apară un mesaj de eroare în apropierea elementului în care a fost introdusă data. Iar dacă utilizatorul introduce o parolă care nu corespunde cu cerințele îi este afișat mesajul special pentru parolă mai puternică. Dacă cursor-ul este pus peste acest mesaj, sunt afișate cerințele pentru o parolă validă.

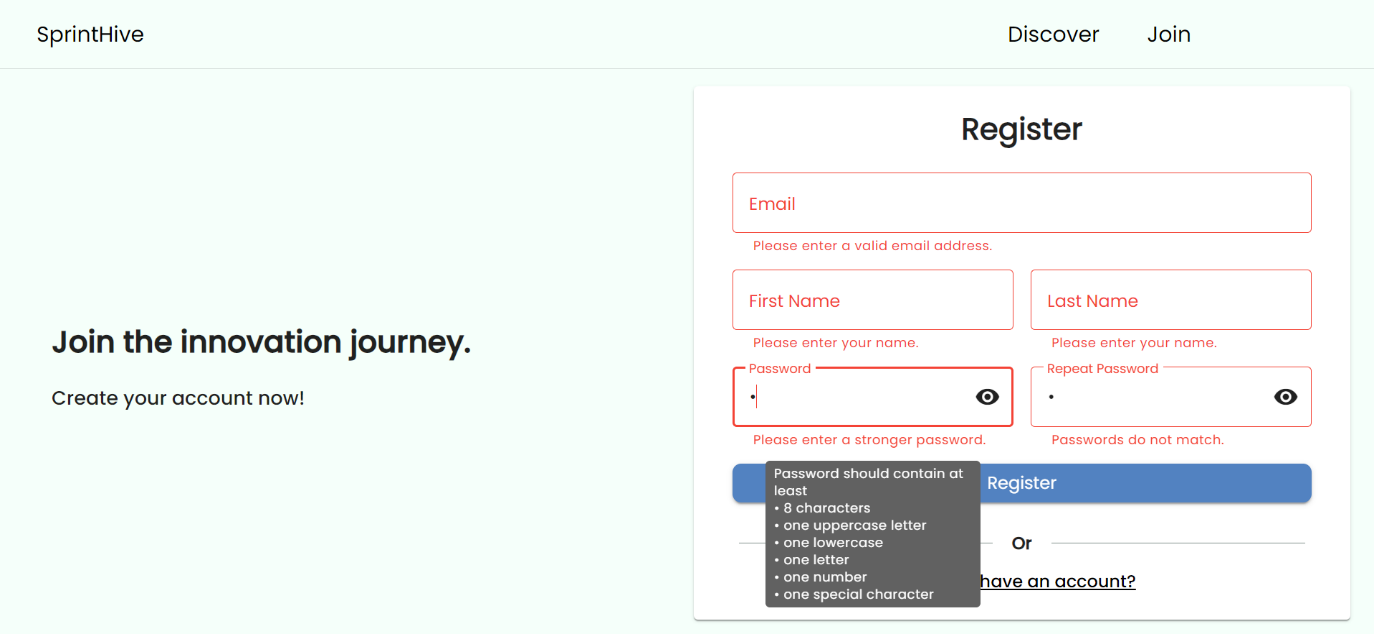


Figura 4.2.2.2 Pagina de înregistrare cu eroare

Dacă utilizatorul a introdus datele corecte și a apăsat butonul de register aplicația trimite o cerere de creare a unui nou cont. În cazul în care noul cont a fost creat cu succes, utilizatorului o să i se afișeze un mesaj de confirmare și o să primească un email în care este rugat să își valideze noul cont.

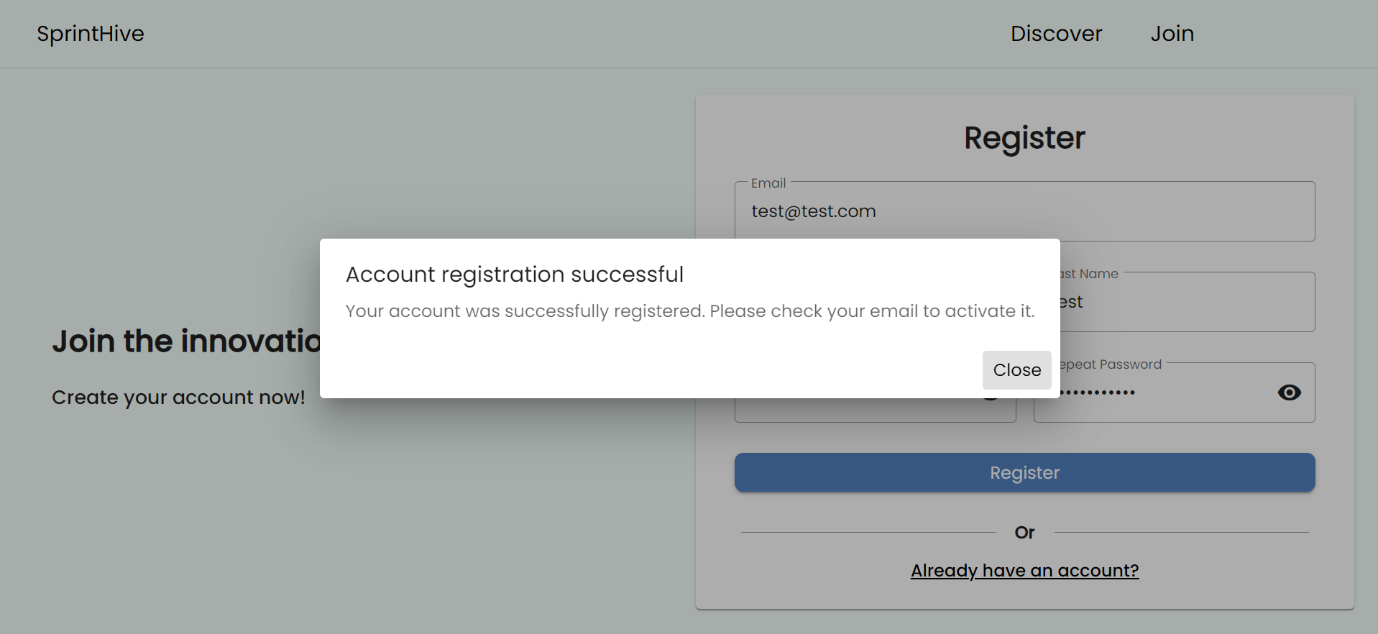


Figura 4.2.2.3 Mesaj de confirmare pentru înregistrare reușită

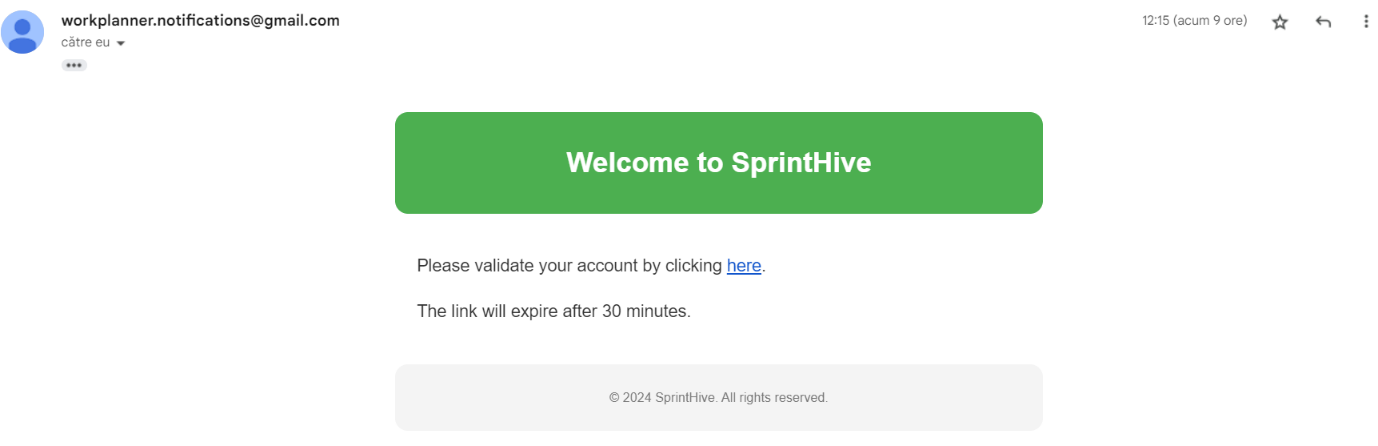


Figura 4.2.2.4 Email pentru validarea noului cont

De asemenea, din această pagină utilizatorul se poate deplasa înapoi la pagina de autentificare, la pagina de informații sau la pagina de întâmpinare.

### 4.2.3 Dashboard

După ce un utilizator s-a autentificat cu succes, acesta este redirecționat către pagina de dashboard. Aici, acesta poate să selecteze proiectul pentru care dorește să vadă informații utile despre progresul în sprint-ul activ, proiect sau comparații între sprint-urile trecute.

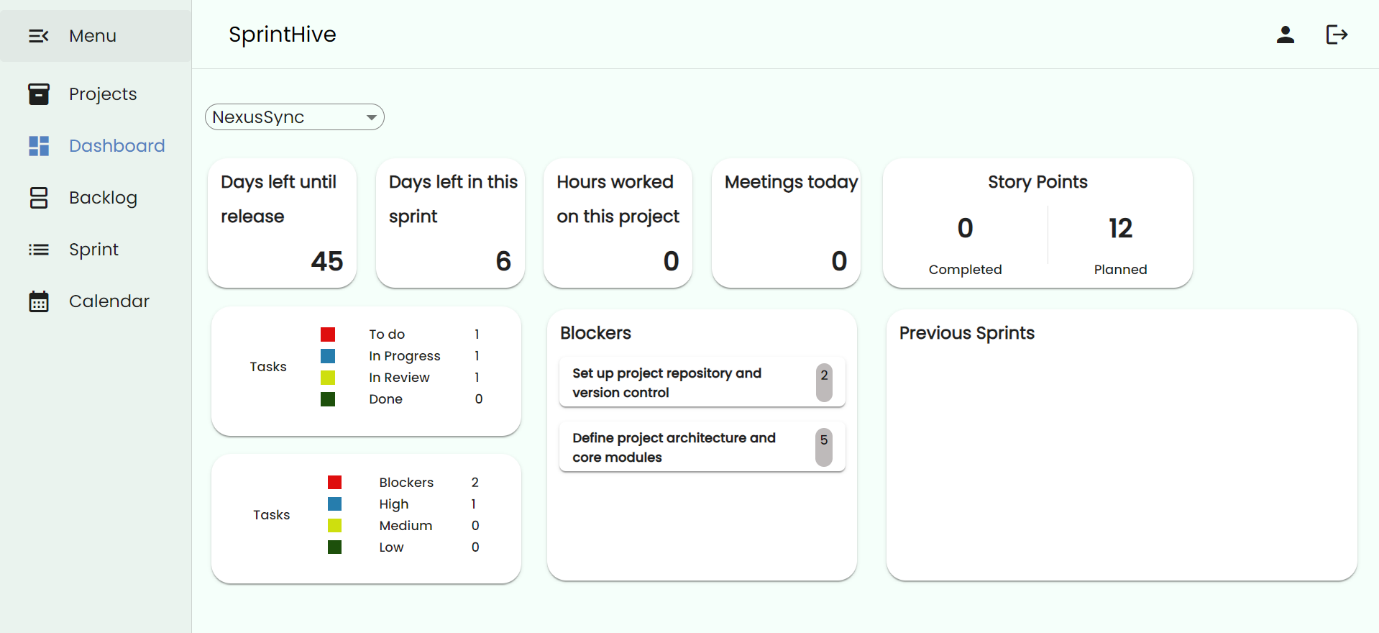


Figura 4.2.3.1 Pagina de dashboard

Dacă nu există proiecte pe care utilizatorul să le poată selecta, acesta poate naviga la pagina de proiecte.

### 4.2.4 Vizualizarea proiectelor și crearea unui nou proiect

În pagina de proiecte utilizatorul poate vedea toate proiectele existente împreună cu detalii despre acestea. De asemena, aici poate filtra proiectele existente pentru a le vedea doar pe cele dorite.

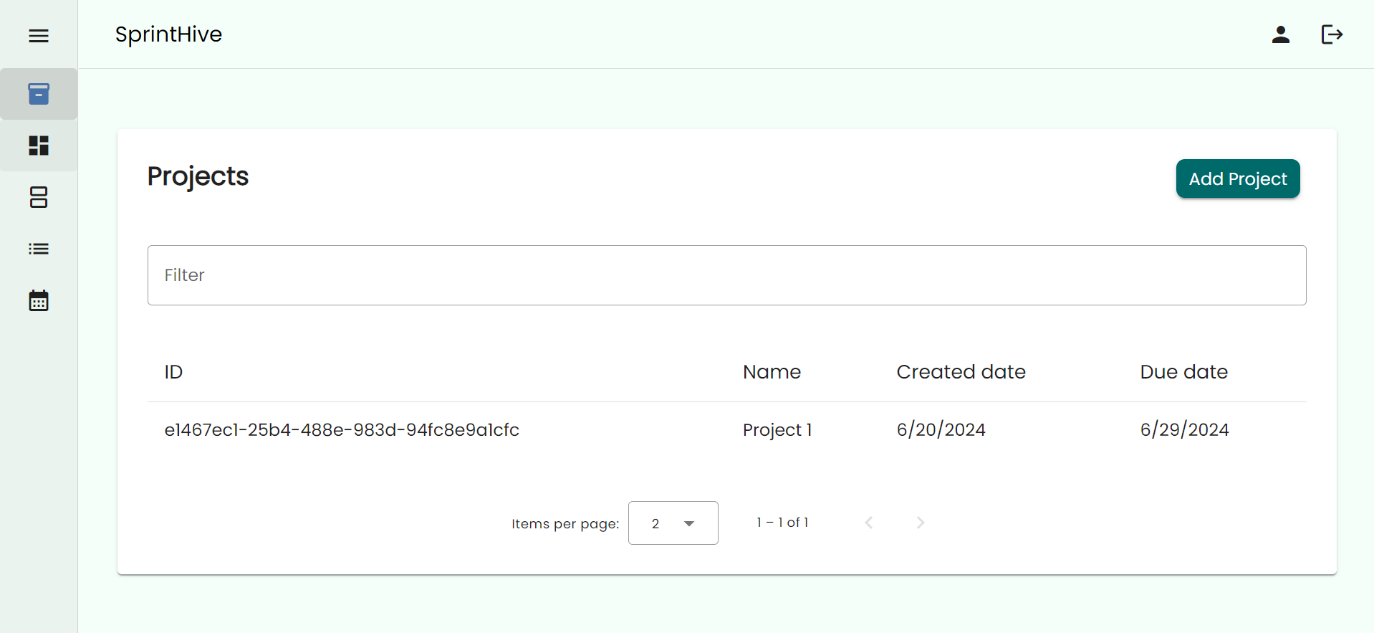


Figura 4.2.4.1 Pagina de vizualizare proiecte

Filtrarea proiectelor este efectuată prin tastarea oricărui caracter în interiorul câmpului “Filter” și este comparată cu toate caracteristicile programului. În continuare, utilizatorul poate adăuga proiecte din această pagină apăsând pe butonul “Add Project”. Acest buton deschide un dialog în care utilizatorul poate introduce datele necesare pentru a adăuga un nou proiect.



Figura 4.2.4.3 Dialog pentru adăugarea unui nou proiect

Pentru a putea crea un nou proiect, utilizatorul trebuie să completeze câmpurile de nume și data finală a proiectului, iar câmpul pentru descriere poate fi ignorat.

### 4.2.5 Vizualizare sprint-uri și crearea unui nou sprint

După ce utilizatorul a adăugat un proiect, în cazul în care nu exista niciunul, acesta poate naviga la pagina de vizualizare a sprint-urilor. Aici o să poată vedea backlog-ul și sprint-urile împreună cu sarcinile corespunzătoare.

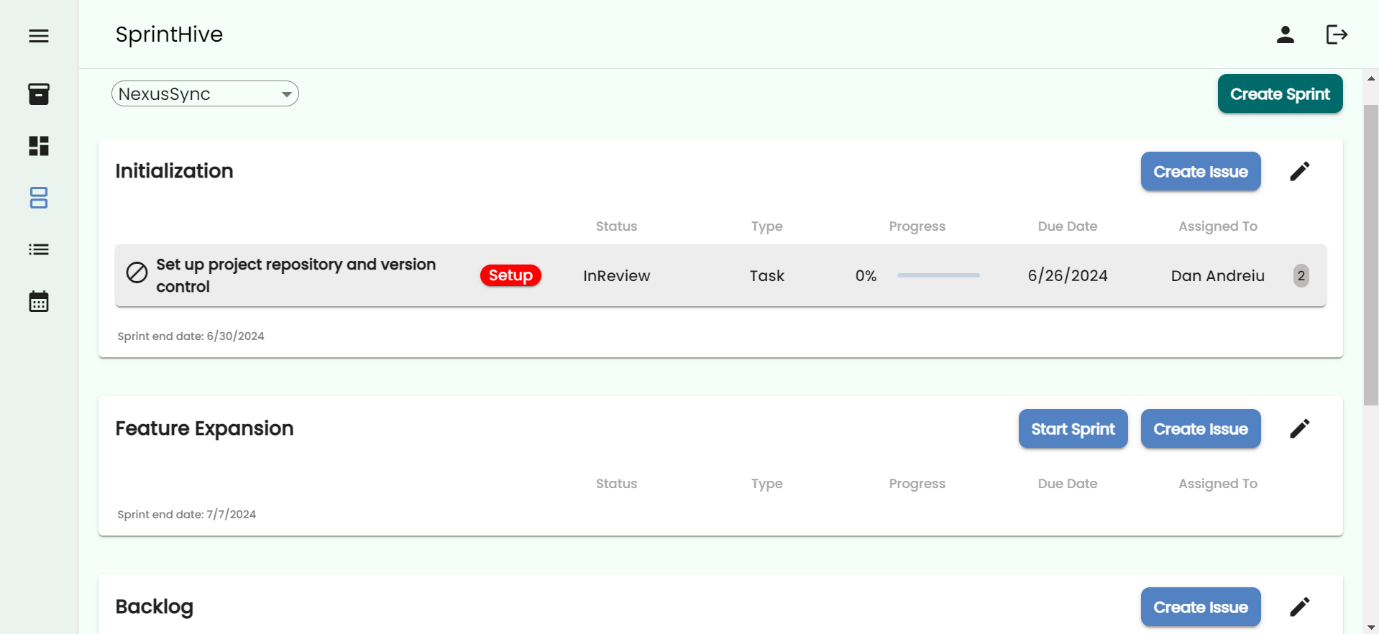


Figura 4.2.5.1 Pagina de vizualizare a sprint-urilor și a backlog-ului

În partea de sus o să fie afișat sprint-ul activ, dacă există unul, iar apoi o să se găsească sprint-urile inactive. Acestea se pot diferenția prin apariția butonului „Start Sprint” la sprint-urile inactive. În cazul în care nu există niciun sprint creat, backlog-ul o să fie tot timpul afișat, iar aici o să se poată adăuga noi sarcini. De asemenea, din această pagina se poate crea un nou sprint prin apăsarea butonului “Create Sprint”. Acest buton deschide un dialog care conține câmpurile necesare pentru a adăuga un nou sprint.

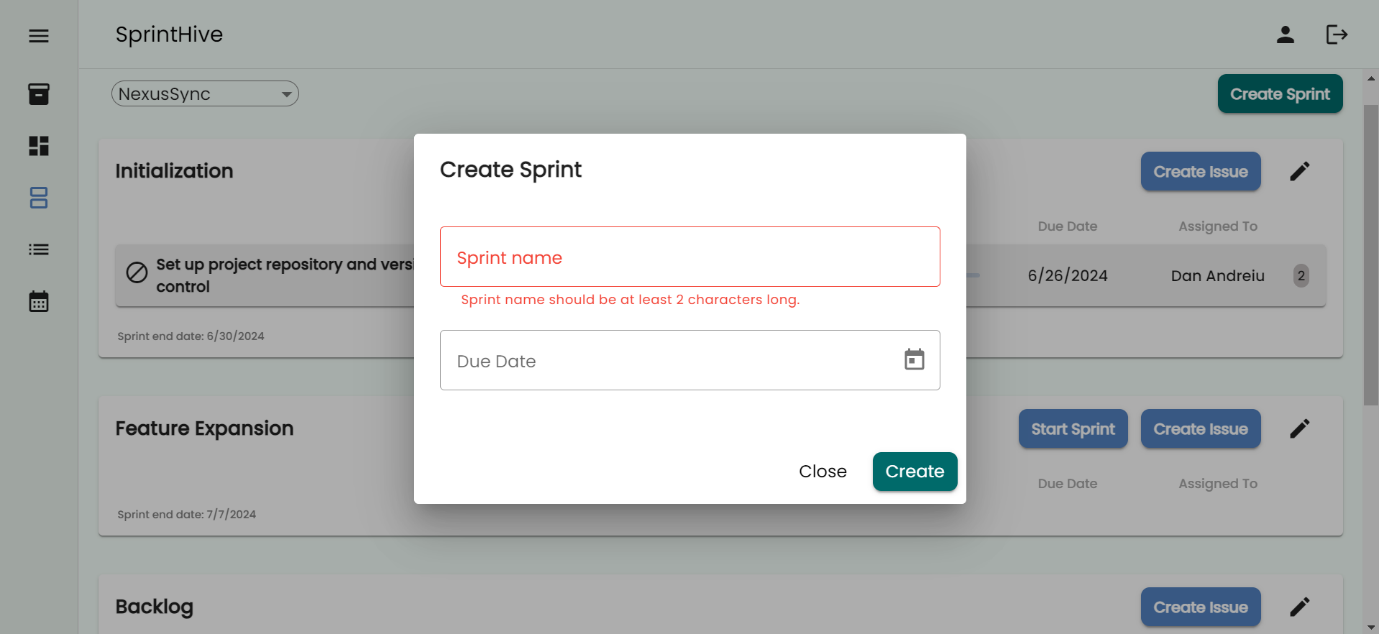


Figura 4.2.5.2 Dialog pentru crearea unui nou sprint

Pentru a putea adăuga un nou sprint, câmpurile prezente în dialog sunt absolut necesare. De asemenea, pentru a muta o sarcină dintr-un sprint în altul sau în backlog, utilizatorul poate selecta sarcina și o poate trage în locul dorit.

Dacă utilizatorul dorește să activeze un sprint, acesta poate apăsa butonul “Start Sprint”, care apelează următoarea metodă.

public startSprint(sprintId: string) {

if(this.activeSprint.id !== ''){

this.openActivateSprintDialog(sprintId);

}

else{

this.sprintService.activate(sprintId).subscribe(() => {

this.ngOnInit();

});

}

}

Această metodă verifică dacă există un sprint activ, iar dacă există, o să fie afișat un dialog de confirmare a activării sprint-ului, deoarece activarea a două sprint-uri în același timp nu este posibilă.

private openActivateSprintDialog(sprintId: string) {

const dialogRef = this.dialog.open(ActivateSprintAlertDialogComponent);

dialogRef.afterClosed().subscribe(result => {

if(result === false || result === undefined){

return;

}

this.sprintService.release(this.activeSprint.id).subscribe(() => {

this.sprintService.activate(sprintId).subscribe(() => {

this.ngOnInit();

});

});

});

}

Atunci când dialogul a fost închis, se verifică dacă utilizatorul a ales să continue cu activarea sprint-ului. Caz în care sprint-ul activ o să fie încheiat, iar acel sprint care se dorea să fie activat o să fie activat dacă nu există probleme în încheierea sprint-ului activ.

### 4.2.6 Crearea unei noi sarcini

Dacă utilizatorul dorește să creeze o nouă sarcină, acesta trebuie să apese pe butonul “Create Issue” din interiorul sprint-ului sau backlog-ului. Astfel, noua sarcină o să apară în interiorul componentei alese. În momentul în care acesta a apăsat pe butonul de creare a sarcinii acesta este redirecționat pe o altă pagină unde poate introduce datele necesare pentru a putea crea sarcina.

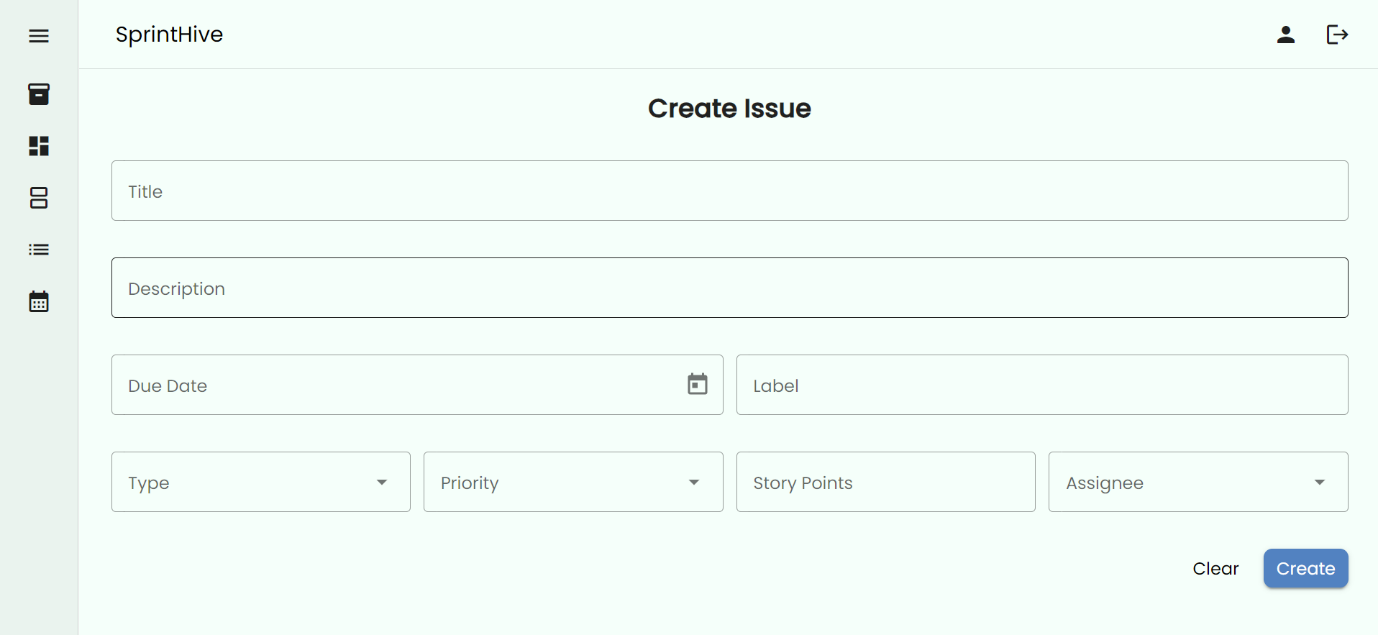


Figura 4.2.6.1 Pagina de creare a unei noi sarcini

În această pagină toate câmpurile, în afară de câmpul pentru label și pentru utilizatorul asignat, sunt necesare și afișează un mesaj de eroare în cazul în care au fost lăsate necompletate. La apăsarea butonului “Create”, dacă toate câmpurile sunt valide, se adaugă noua sarcină în baza de date, iar utilizatorul este redirecționat înapoi la pagina de backlog.

### 4.2.7 Vizualizarea statusului pentru sprintul activ

Dacă utilizatorul dorește să verifice stagiul în care se află sarcinile lui și a colaboratorilor la acest proiect în sprint-ul activ, acesta poate accesa pagina de sprint. În această pagină se află toate sarcinile care doresc să fie completate în cadrul acestui sprint. De asemenea, aici sarcinile sunt grupate în funcție de statusul în care se află pentru o mai bună înțelegere a progresului în sprint.

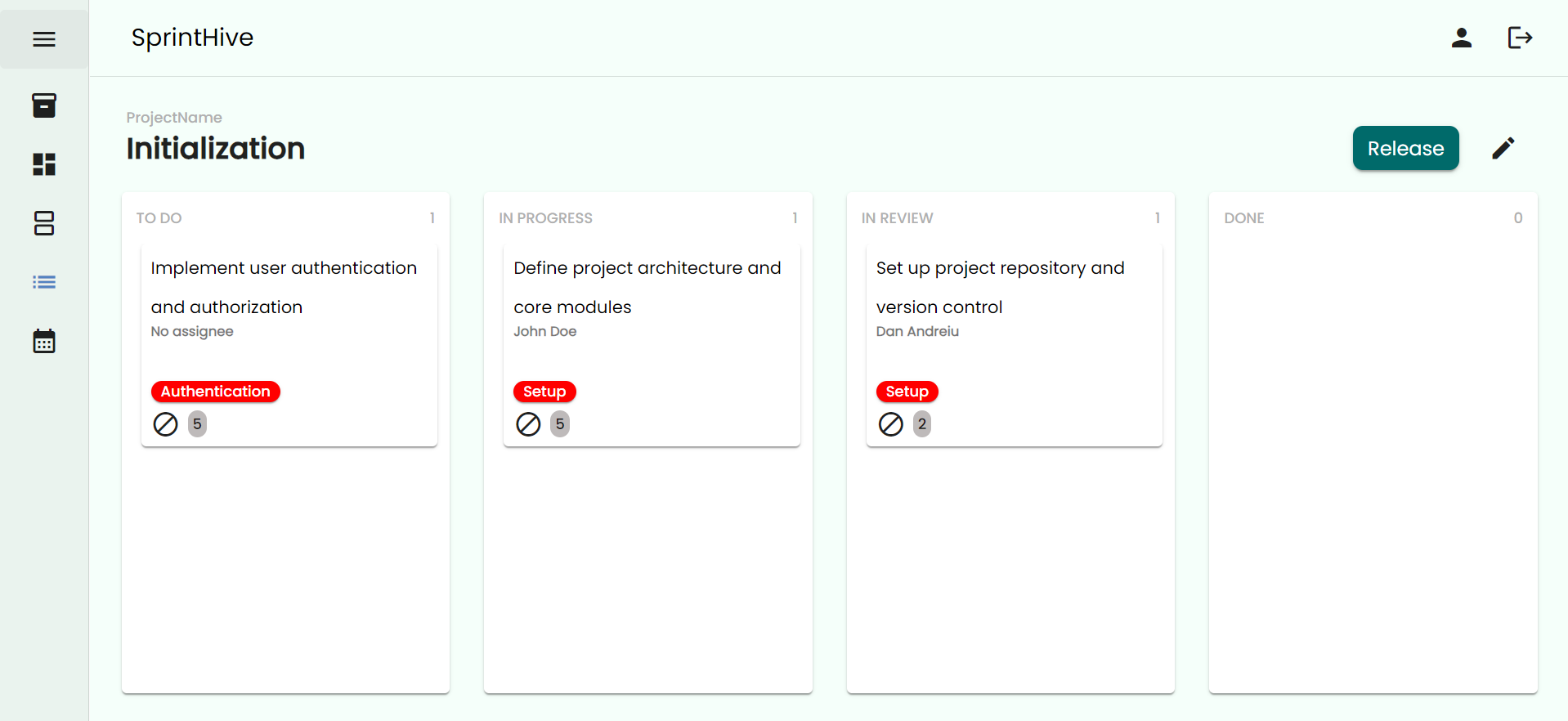


Figura 4.2.7.1 Pagina sprint-ului activ

Din această pagină utilizatorul poate închide sprint-ul curent prin apăsarea butonului “Release”. De asemenea, pentru a modifica status-ul unei sarcini, utilizatorul poate selecta o sarcină și o poate trage în coloana de status dorită, moment în care se execută următorul cod.

public async drop(event: CdkDragDrop<Task[]>) {

if (event.previousContainer === event.container) {

moveItemInArray<Task>(event.container.data, event.previousIndex, event.currentIndex);

} else {

let wasUpdateSuccessful = false;

let movedTask = new Task();

let newStatus: StatusTypes = StatusTypes.ToDo;

switch(event.container.id) {

case 'toDoList':

newStatus = StatusTypes.ToDo;

break;

case 'inProgressList':

newStatus = StatusTypes.InProgress;

break;

case 'inReviewList':

newStatus = StatusTypes.InReview;

break;

case 'doneList':

newStatus = StatusTypes.Done;

break;

default:

console.log('default');

break;

}

movedTask = event.previousContainer.data[event.previousIndex];

const modified$ = this.taskService.updateStatus(movedTask.id, newStatus);

wasUpdateSuccessful = await lastValueFrom(modified$);

if(wasUpdateSuccessful) {

transferArrayItem<Task>(

event.previousContainer.data,

event.container.data,

event.previousIndex,

event.currentIndex,

);

} else {

alert('Task status update failed');

}

}

}

În acest cod se verifică dacă sarcina a fost trasă în același loc, iar dacă nu, se execută mutarea. Mutarea are loc, în primul rând, prin verificarea status-ului în care a fost trasă sarcina. Apoi se apelează API-ul pentru a modifica status-ul în baza de date de unde se primește un raspunse de tip bool. Dacă sarcina a fost modificată în baza de date se poate modifica și în aplicația web. Dacă nu, este afișat un mesaj de alertă și sarcina rămâne în poziția inițială.

### 4.2.8 Pontaj

În pagina de calendar, accesibilă prin intermediul bării de navigare din partea stângă, utilizatorul poate să introducă evenimente în calendar sau să vizualizeze evenimente mai vechi. Aceste evenimente sunt interpretate ca și pontări.

Pentru a adăuga un eveniment utilizatorul trebuie să apese pe calendar relativ la ziua și ora de unde dorește să înceapă evenimentul, iar apoi ținând click-ul apăsat să tragă în jos până ajunge la ora dorită. Acest eveniment poate fi modificat prin apăsarea de două ori în interiorul acestuia, deschizând o fereastră cu detalii pentru acesta prin această medtodă.

private onCalendarEventDoubleClick(event: MouseEvent, eventElement: HTMLElement): void {

this.viewContainerRef = this.dynamicHost.viewContainerRef;

this.viewContainerRef.clear();

const componentRef = this.viewContainerRef.createComponent(CalendarEventDetailsComponent);

this.calendarEventDetailsComponent = componentRef.instance;

componentRef.instance.onClose.subscribe(() => this.onCalendarEventDetailsClose());

componentRef.instance.onDelete.subscribe(() => this.onCalendarEventDelete());

componentRef.instance.onSave.subscribe((event) => this.onCalendarEventModified(event));

this.calendarEventDetailsComponent.initialize(eventElement, this.calendar, this.calendarEvents);

this.renderer.addClass(this.calendar.nativeElement, 'disable-scroll');

event.stopPropagation();

event.preventDefault();

this.removeDocumentListeners();

this.documentClickListener = this.renderer.listen('document', 'click', this.onDocumentClick.bind(this));

this.documentContextMenuListener = this.renderer.listen('document', 'contextmenu', this.onDocumentClick.bind(this));

this.documentMouseDownListener = this.renderer.listen('document', 'mousedown', this.onDocumentClick.bind(this));

}

Această metodă setează referința la container-ul de vizualizare din directiva dynamic host, folosită pentru a afișa componente în cadrul altor elemente din această pagină în mod dinamic. Apoi este creată componenta de detalii a evenimentului și se mapează evenimentele din componenta detalii la metodele aferente din componenta calendar.

După care se inițializează componenta detalii și se elimină posibilitatea de a derula pagina. Apoi sunt anulate acțiunile implicite a evenimentului de mouse și se elimină propragarea acestui eveniment către alte componente.

În final, sunt resetate evenimentele click-urlior din document, după ce a fost adăugată componenta detalii, pentru a putea închide această componentă când realizăm un click-ul în afara acesteia.

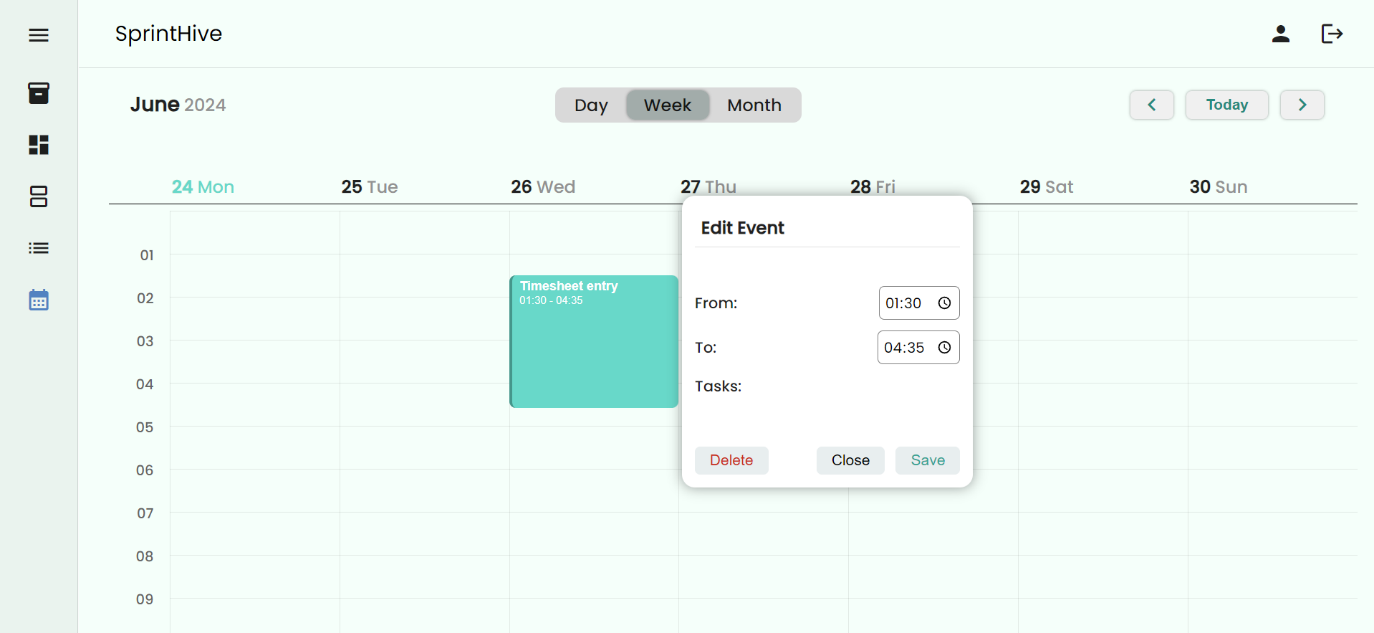


Figura 4.2.8.1 Pagina de calendar cu evenimente

Din această fereastră utilizatorul poate modifica ora la care evenimentul începe și ora la care se oprește. De asemena, din această fereastră utilizatorul poate șterge evenimentul. În această pagină utilizatorul mai dispune și de trei butoane de navigare prin calendar, aflate în partea din dreapta sus a ecranului. Aceste butoane modifică calendarul în funcție de afișajul selectat în partea de sus a paginii, în mijloc.

### 4.2.9 Vizualizarea unei sarcini

Prin apăsarea pe o sarcină, utilizatorul o să fie redirecționat la pagina de vizualizare a sarcinii. Aici îi sunt oferite toate detaliile despre acea sarcină precum și toate sub sarcinile acelei sarcini.

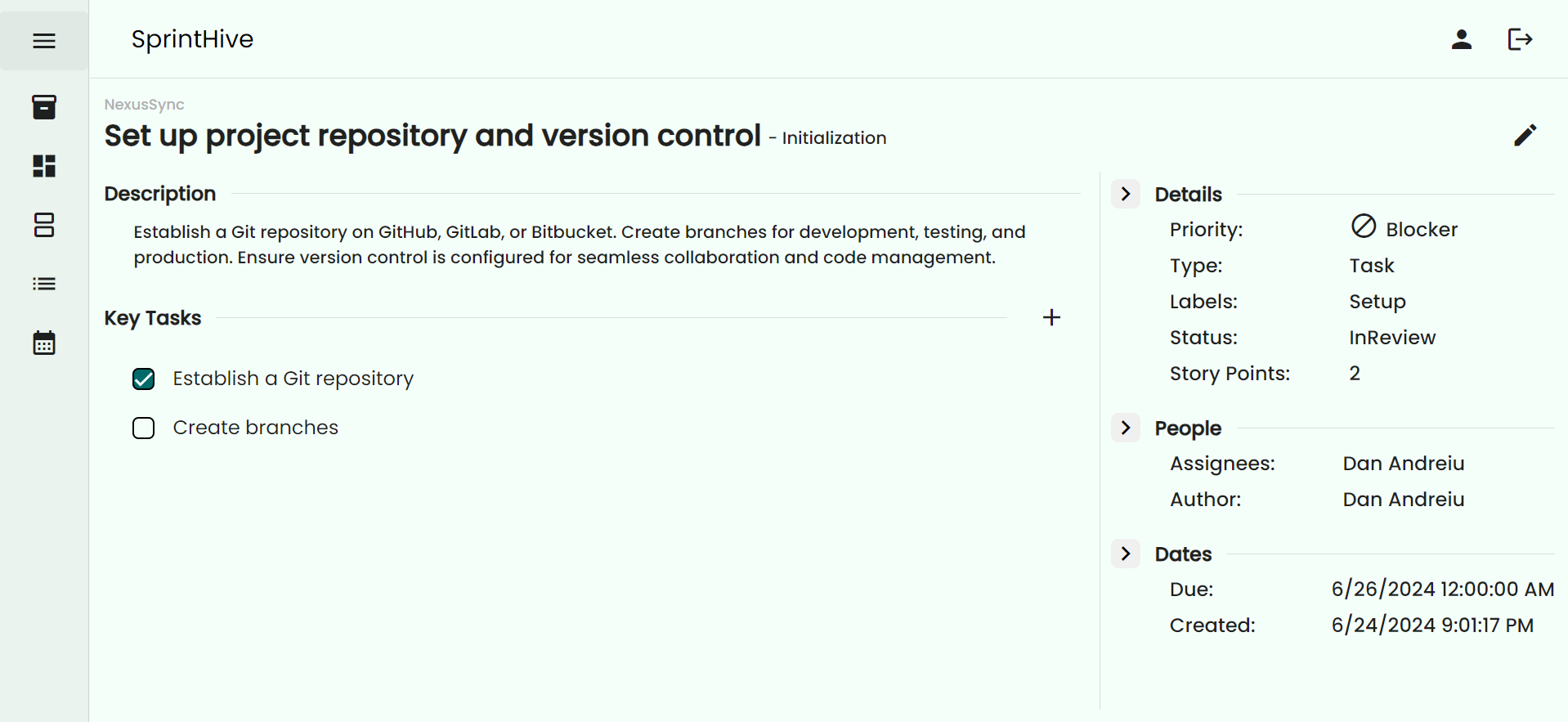


Figura 4.2.9.1 Pagina de vizualizare a sarcinii

În cadrul acestei pagini, utilizatorul poate adăuga noi sub sarcini unei sarcini prin apăsarea butonului “+” din dreptul secțiunii “Key Tasks”. Acest buton deschide un dialog în care este cerut numele sub sarcinii care dorește să fie adăugată, fără de care nu se poate adăuga. După adăugare, sub sarcina este afișată în pagină. Apoi utilizatorul poate bifa căsuța sub sarcinii pentru a o evidenția ca și completată.

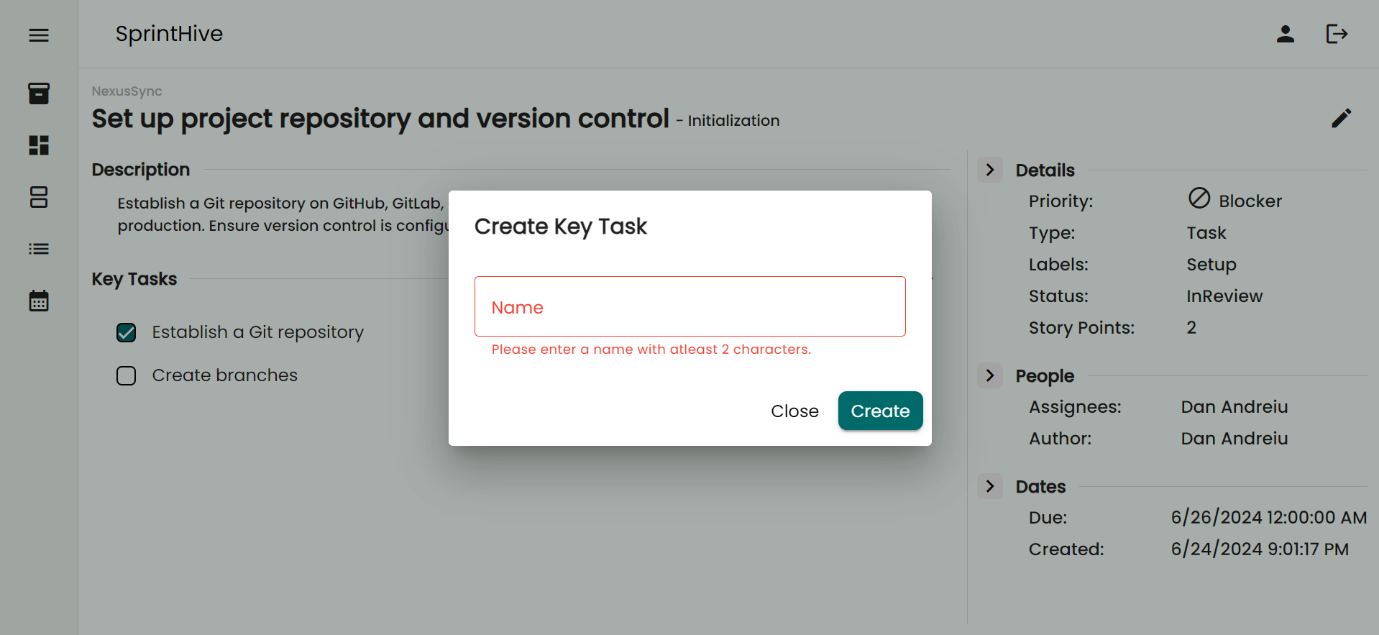


Figura 4.2.9.2 Dialog pentru adăugarea unei noi sub sarcini

### 4.2.10 Vizualizarea informațiilor personale

Dacă utilizatorul dorește să vizualizeze informațiile stocate despre acesta poate naviga către pagina de utilizator. Ca să ajungă pe acea pagină trebuie să apese butonul din bara de unelte cu imaginea unei persoane, situat în partea dreaptă.



Figura 4.2.10.1 Buton pentru navigarea pe pagina utilizatorului

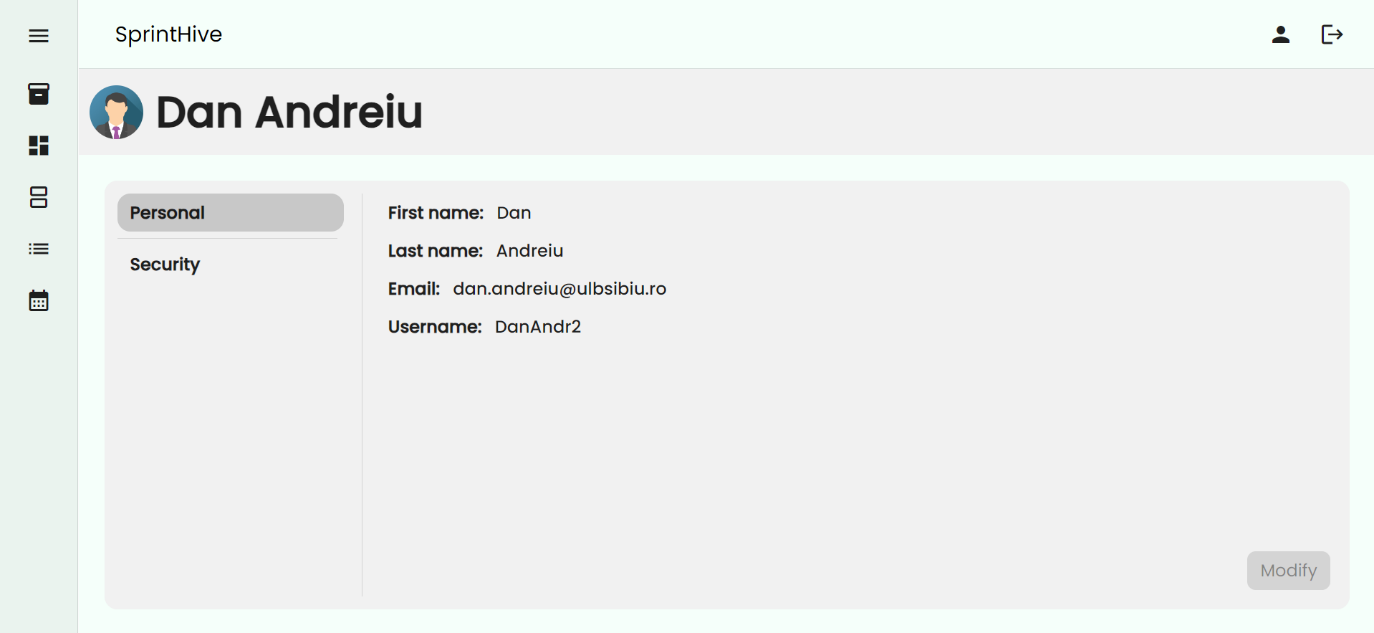


Figura 4.2.10.2 Pagina utilizatorului

În această pagină, inițial sunt afișate datele personale ale utilizatorului, iar acesta poate comuta la afișarea paginii despre securitate.

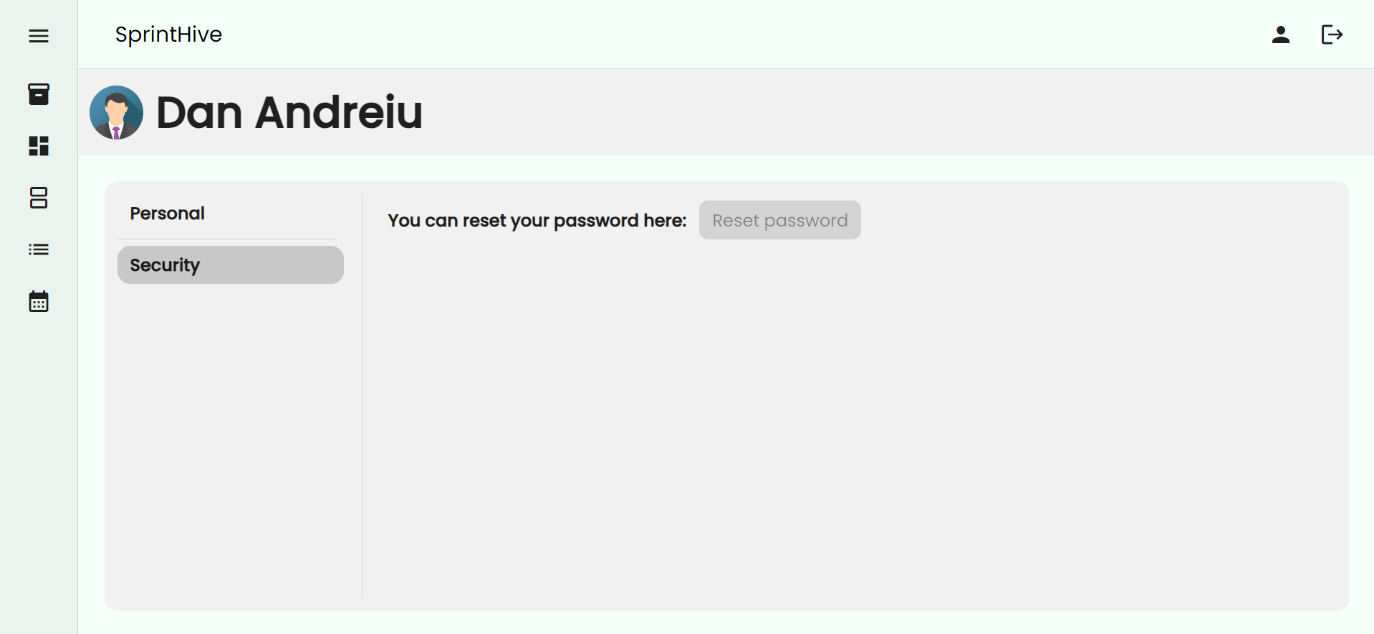


Figura 4.2.10.2 Afișarea paginii de securitate

### 4.2.11 Logout

Dacă utilizatorul autentificat dorește să încheie sesiunea curentă de navigare, acesta poate apăsa butonul din bara de unelte cu imaginea de ieșire din partea dreaptă. După apăsarea acestui buton utilizatorul este redirecționat către pagina de autentificare și accesul la paginile anterioare îi este restricționat.



Figura 4.2.11.1 Buton de logout

## 4.3 Descrierea API-ului

În această secțiune este descris API-ul creat pentru a ușura interacțiunea cu baza de date din pagina web. Acest API pune la dispoziție șase controllere. Pentru a putea autoriza folosirea anumitor endpoint-uri am folosit JSON Web Token. Aceasta a fost realizată prin adnotarea endpoint-ului sau controller-ului care se dorește a nu fi accesat fără autentificare.

De asemenea, endpoint-urile dispun și de un serviciu de filtrare a excepțiilor. Acest serviciu este folosit pentru a captura excepțiile apărute în timpul execuției cererii și pentru a returna un mesaj informativ către pagina web. Codul de status HTTP returnat este 401 Bad Request, împreună cu numele controller-ullui și mesajul excepției în format JSON.

### 4.3.1 User Controller

Este folosit pentru a efectua operații CRUD în tabelul de utilizatori din baza de date. Acesta conține următoarele endpoint-uri.

**Authenticate**

Este folosit pentru a autentifica un utilizator. Acest endpoint nu necesită autorizare pentru a fi folosit.

|  |  |
| --- | --- |
| Request | Response |
| Route: POST /api/User/authenticate  Content-Type: application/json  Accept: \*/\*  Body:  {  "email": "admin@admin.com",  "password": "QWRtaW4xMiE="  } | HTTP Status Code: 200 OK  Content-Type: text/plain  Body: valoarea token-ului generat |

**Register**

Este folosit pentru a înregistra un nou utilizator și nu are nevoie de autentificare pentru a putea fi folosit.

|  |  |
| --- | --- |
| Request | Response |
| Route: POST /api/User/register  Content-Type: application/json  Accept: \*/\*  Body:  {  "firstName": "Martin",  "lastName": "Brundle",  "email": "martin\_brundle@test.com",  "password": "QWRtaW4xMiE="  } | HTTP Status Code: 201 Created |

**Validate**

Este accesat atunci când utilizatorul care s-a înregistrat de curând accesează link-ul din email-ul primit pentru a își valida noul cont. Pentru a putea accesa acest endpoint nu este nevoie de autorizare. După accesarea acestui endpoint, utilizatorul este redirecționat către pagina din aplicația web în care este confirmată validarea contului.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/User/validate?validationToken=ODA3RjNCNTgtQTE1Ri00NEMyLTk0MUUtOEVERUE0MTMxRTMz  Accept: \*/\* | HTTP Status Code: 302 Redirect |

**Get**

Este folosit pentru a pentru a primi informații despre utilizatorul cerut. Acest endpoint are nevoie de autorizare pentru a putea fi folosit. În cazul în care este folosit fără autorizare este returnat codul de status 401.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/User/EE5688E5-0DE6-4653-8122-A1D4FB536DAB  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  {  "firstName": "Dan",  "lastName": "Andreiu",  "email": "dan.andreiu@ulbsibiu.ro",  "username": "DanAndr2"  } |

**GetAll**

Acest endpoint returnează toți utilizatorii existenți. Are nevoie de autorizare pentru a putea fi folsosit. În cazul în care este folosit fără autorizare este returnat codul de status 401.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/User  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "firstName": "Martin",  "lastName": "Brundle",  "email": "martin\_brundle@test.com",  "username": "MartinBrun"  },  {  "firstName": "Dan",  "lastName": "Andreiu",  "email": "dan.andreiu@ulbsibiu.ro",  "username": "DanAndr2"  }  ] |

### 4.3.2 Project Controller

Este folosit pentru a efectua operații CRUD în tabelul de proiecte din baza de date. Pentru a accesa acest controller este nevoie de autorizare. În cazul în care orice endpoint din acest controller este folosit fără autorizare este returnat codul de status 401. Acesta conține următoarele endpoint-uri.

**Create**

Acest endpoint este folosit pentru a crea un nou proiect.

|  |  |
| --- | --- |
| Request | Response |
| Route: POST /api/Project  Content-Type: application/json  Accept: \*/\*  Body:  {  "name": "Project1",  "creatorUsername": "adminadmi",  "description": "test",  "dueDate": "2024-06-20T08:29:30.444Z"  } | HTTP Status Code: 201 Created  Content-Type: application/json  Body:  {  "id": "c8b6cd15-2631-4a31-b7e4-de16adfd5bba",  "name": "Project1",  "description": "test",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "startDate": "2024-06-20T08:31:45",  "dueDate": "2024-06-20T08:29:30.444Z"  } |

**GetAll**

Acest endpoint este folosit pentru a returna toate proiectele existente.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Project  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "e1467ec1-25b4-488e-983d-94fc8e9a1cfc",  "name": "Project 1",  "description": "Primul proiect",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "startDate": "2024-06-20T11:00:45",  "dueDate": "2024-06-29T14:00:00",  "sprints": []  }  ] |

**Get**

Este folosit pentru a returna un proiect specific.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Project/E1467EC1-25B4-488E-983D-94FC8E9A1CFC  Accept: \*/\* | Http Status Code: 200 OK  Content-Type: application/json  Body:  {  "id": "e1467ec1-25b4-488e-983d-94fc8e9a1cfc",  "name": "Project 1",  "description": "Primul proiect",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "startDate": "2024-06-20T11:00:45",  "dueDate": "2024-06-29T14:00:00",  "sprints": []  } |

**GetAllWithAllChildren**

Acest endpoint este folosit pentru a returna toate proiectele existente împreună cu datele legate de acestea prin relațiile definite între tabele.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Project  /getAllChildren  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "e1467ec1-25b4-488e-983d-94fc8e9a1cfc",  "name": "Project 1",  "description": "Primul proiect",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "startDate": "2024-06-20T11:00:45",  "dueDate": "2024-06-29T14:00:00",  "sprints": [  {  "id": "6a80a893-ddc3-496b-b56e-21d51aa2a1fc",  "name": "Sprint 1",  "startDate": "2024-06-20T11:07:55",  "dueDate": "2024-06-21T14:07:00",  "active": false,  "released": true,  "tasks": [  {  "id": "3d154916-6ec8-4fb0-ab15-ed3369d17212",  "name": "Task 1",  "description": "Primul task",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:07:38",  "dueDate": "2024-06-28T14:01:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": [  {  "id": "5fd2a7c8-2226-4170-b5a1-b6ec5c3044c5",  "name": "Subtask1",  "done": true  }  ]  }  ]  },  ]  }  ] |

**GetSprints**

Acest endpoint este folosit pentru a returna toate sprint-urile unui proiect.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Project/E1467EC1-25B4-488E-983D-94FC8E9A1CFC/sprints  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "6a80a893-ddc3-496b-b56e-21d51aa2a1fc",  "name": "Sprint 1",  "startDate": "2024-06-20T11:07:55",  "dueDate": "2024-06-21T14:07:00",  "active": false,  "released": true,  "tasks": []  },  {  "id": "c09ca199-b708-417c-86ec-45c4832db829",  "name": "Sprint 2",  "startDate": "2024-06-23T17:22:35",  "dueDate": "2024-06-27T21:00:00",  "active": false,  "released": true,  "tasks": []  }  ] |

**GetActiveSprint**

Acest endpoint este folosit pentru a returna doar sprint-ul activ al unui proiect.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Project  /E1467EC1-25B4-488E-983D-94FC8E9A1CFC  /activeSprint  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  {  "id": "580115ac-a516-4512-811b-f39e5f33c7b2",  "name": "Sprint 4",  "startDate": "2024-06-23T21:50:39",  "dueDate": "2024-06-25T21:00:00",  "active": true,  "released": false,  "tasks": [  {  "id": "20e7ed49-5916-461a-b1b7-291cf11c03be",  "name": "Task 1",  "description": "Test",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "00000000-0000-0000-0000-000000000000",  "startDate": "2024-06-23T21:51:32",  "dueDate": "2024-06-28T00:00:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 2,  "subtasks": []  }  ]  } |

### 4.3.3 Backlog Controller

Este folosit pentru a efectua operații CRUD în tabelul de backlog din baza de date. Pentru a accesa acest controller este nevoie de autorizare. În cazul în care orice endpoint din acest controller este folosit fără autorizare este returnat codul de status 401. Acesta conține un singur endpoint.

**GetForProject**

Este folosit pentru a returna sarcinile din backlog-ul aferent proiectului al cărui id a fost introdus.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Backlog/  projectId=  E1467EC1-25B4-488E-983D-94FC8E9A1CFC  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "098834c9-da4b-4a52-9d58-69a1fa035d06",  "name": "Task 2",  "description": "",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "00000000-0000-0000-0000-000000000000",  "startDate": "2024-06-20T11:08:18",  "dueDate": "2024-06-26T14:08:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 2,  "subtasks": []  }  ] |

### 4.3.4 Sprint Controller

Este folosit pentru a efectua operații CRUD în tabelul de sprint-uri din baza de date. Pentru a accesa acest controller este nevoie de autorizare. În cazul în care orice endpoint din acest controller este folosit fără autorizare este returnat codul de status 401. Acesta conține următoarele endpoint-uri.

**ReleaseSprint**

Acest endpoint finalizează și dezactivează un sprint.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint/6A80A893-DDC3-496B-B56E-21D51AA2A1FC/release  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

**Create**

Este folosit pentru a crea un nou sprint.

|  |  |
| --- | --- |
| Request | Response |
| Route: POST /api/Sprint  Content-Type: application/json  Accept: \*/\*  Body:  {  "name": "Sprint 6",  "dueDate": "2024-06-21T18:47:05.620Z",  "creatorId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  "projectId": "E1467EC1-25B4-488E-983D-94FC8E9A1CFC"  } | HTTP Status Code: 201 Created  Content-Type: application/json  Body:  {  "id": "1d83c591-00d8-4495-8a40-fc6c20709eab",  "name": "Sprint 6",  "startDate": "2024-06-24T15:11:10",  "dueDate": "2024-06-21T18:47:05.62Z",  "active": false,  "released": false,  "tasks": []  } |

**GetAll**

Acest endpoint returnează toate sprint-urile disponibile.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "6a80a893-ddc3-496b-b56e-21d51aa2a1fc",  "name": "Sprint 1",  "startDate": "2024-06-20T11:07:55",  "dueDate": "2024-06-21T14:07:00",  "active": false,  "released": true,  "tasks": []  },  {  "id": "6fb67d8d-5ac1-451c-95ea-bfff31f87527",  "name": "Sprint 2",  "startDate": "2024-06-22T07:55:07",  "dueDate": "2024-06-21T18:47:05.62",  "active": false,  "released": true,  "tasks": []  },  ] |

**Get**

Acest endpoint returnează un sprint specific.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint/6A80A893-DDC3-496B-B56E-21D51AA2A1FC  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  {  "id": "6a80a893-ddc3-496b-b56e-21d51aa2a1fc",  "name": "Sprint 1",  "startDate": "2024-06-20T11:07:55",  "dueDate": "2024-06-21T14:07:00",  "active": false,  "released": true,  "tasks": []  } |

**GetTasksForSprint**

Acest endpoint returnează toate sarcinile unui sprint.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint  /6A80A893-DDC3-496B-B56E-21D51AA2A1FC  /tasks  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "3d154916-6ec8-4fb0-ab15-ed3369d17212",  "name": "Task 1",  "description": "Primul task",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:07:38",  "dueDate": "2024-06-28T14:01:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": []  }  ] |

**GetTasksByTaskPriorityForSprint**

Este folosit pentru a accesa sarcinile unui sprint în funcție de prioritatea acestora.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint  /6A80A893-DDC3-496B-B56E-21D51AA2A1FC  /tasks /priority/1  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "3d154916-6ec8-4fb0-ab15-ed3369d17212",  "name": "Task 1",  "description": "Primul task",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:07:38",  "dueDate": "2024-06-28T14:01:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": []  }  ] |

**GetTasksByTaskStatusForSprint**

Este folosit pentru a accesa sarcinile unui sprint în funcție de status-ul acestora.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint  /6A80A893-DDC3-496B-B56E-21D51AA2A1FC  /tasks/status/0  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "3d154916-6ec8-4fb0-ab15-ed3369d17212",  "name": "Task 1",  "description": "Primul task",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:07:38",  "dueDate": "2024-06-28T14:01:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": []  }  ] |

**GetActiveSprintForProject**

Acest endpoint returnează sprint-urile active ale unui proiect.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint  /6A80A893-DDC3-496B-B56E-21D51AA2A1FC  /tasks  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  {  "id": "580115ac-a516-4512-811b-f39e5f33c7b2",  "name": "Sprint 4",  "startDate": "2024-06-23T21:50:39",  "dueDate": "2024-06-25T21:00:00",  "active": true,  "released": false,  "tasks": [  {  "id": "20e7ed49-5916-461a-b1b7-291cf11c03be",  "name": "Task 1",  "description": "Test",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "00000000-0000-0000-0000-000000000000",  "startDate": "2024-06-23T21:51:32",  "dueDate": "2024-06-28T00:00:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 2,  "subtasks": []  }  ]  } |

**ActivateSprint**

Acest endpoint activează un sprint.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint/6A80A893-DDC3-496B-B56E-21D51AA2A1FC/activate  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

**GetInactiveSprintsForProject**

Acest endpoint returnează sprint-urile inactive ale unui proiect.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Sprint  /6A80A893-DDC3-496B-B56E-21D51AA2A1FC  /tasks  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "1d83c591-00d8-4495-8a40-fc6c20709eab",  "name": "Sprint 6",  "startDate": "2024-06-24T15:11:10",  "dueDate": "2024-06-21T18:47:05.62",  "active": false,  "released": false,  "tasks": []  },  {  "id": "580115ac-a516-4512-811b-f39e5f33c7b2",  "name": "Sprint 4",  "startDate": "2024-06-23T21:50:39",  "dueDate": "2024-06-25T21:00:00",  "active": false,  "released": false,  "tasks": [  {  "id": "20e7ed49-5916-461a-b1b7-291cf11c03be",  "name": "Task 1",  "description": "Test",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "00000000-0000-0000-0000-000000000000",  "startDate": "2024-06-23T21:51:32",  "dueDate": "2024-06-28T00:00:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 2,  "subtasks": []  }  ]  }  ] |

### 4.3.5 Task Controller

Este folosit pentru a efectua operații CRUD în tabelul de sarcini din baza de date. Pentru a accesa acest controller este nevoie de autorizare. În cazul în care orice endpoint din acest controller este folosit fără autorizare este returnat codul de status 401. Acesta conține următoarele endpoint-uri.

**Create**

Este folosit pentru a crea o nouă sarcină.

|  |  |
| --- | --- |
| Request | Response |
| Route: POST /api/Task  Content-Type: application/json  Accept: \*/\*  Body:  {  "name": "Task",  "description": "test",  "sprintId": "6A80A893-DDC3-496B-B56E-21D51AA2A1FC",  "projectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  "assigneeUsername": "DanAndr",  "dueDate": "2024-06-21T14:07:00",  "priority": 0,  "type": 0,  "storyPoints": 3  } | HTTP Status Code: 201 Created  Content-Type: application/json  Body:  {  "id": "177bd93f-da19-40b8-a7f6-1e19b069e183",  "name": "Task",  "description": "test",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-24T15:30:02",  "dueDate": "2024-06-21T14:07:00",  "label": null,  "priority": 0,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": []  } |

**GetAll**

Acest endpoint returnează toate sarcinile.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Task  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "3d154916-6ec8-4fb0-ab15-ed3369d17212",  "name": "Task 1",  "description": "Primul task",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:07:38",  "dueDate": "2024-06-28T14:01:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": []  },  {  "id": "098834c9-da4b-4a52-9d58-69a1fa035d06",  "name": "Task 2",  "description": "",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:08:18",  "dueDate": "2024-06-26T14:08:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 2,  "subtasks": []  }  ] |

**Get**

Acest endpoint returnează o sarcină specifică împreună cu sub sarcinile acesteia.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Task/ 3D154916-6EC8-4FB0-AB15-ED3369D17212  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  {  "id": "3d154916-6ec8-4fb0-ab15-ed3369d17212",  "name": "Task 1",  "description": "Primul task",  "creatorId": "6ee54cdf-c063-439c-8aa6-8b8458697cd6",  "assigneeId": "c145bfcd-8a6e-405f-aec5-9fbf0080f647",  "startDate": "2024-06-20T11:07:38",  "dueDate": "2024-06-28T14:01:00",  "label": "",  "priority": 1,  "status": 0,  "type": 0,  "storyPoints": 3,  "subtasks": [  {  "id": "fc095890-08af-4da2-a576-cd7916966681",  "name": "Subtask 1",  "done": false  },  {  "id": "ee4aa4cc-f654-44f7-9848-ada680a26ec6",  "name": "Subtask 2",  "done": true  },  ]  } |

**GetSprintName**

Este folosit pentru a afla numele sprint-ului de care aparține o sarcină.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Task/ 3D154916-6EC8-4FB0-AB15-ED3369D17212/sprintName  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: text/plain  Body: "Sprint 1" |

**GetProjectName**

Este folosit pentru a afla numele proiectului-ului de care aparține o sarcină.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Task/ 3D154916-6EC8-4FB0-AB15-ED3369D17212/projectName  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: text/plain  Body: "Project 1" |

**AddSubtask**

Este folosit pentru a adăuga o nouă sub sarcină.

|  |  |
| --- | --- |
| Request | Response |
| Route: PUT /api/Task/ 3D154916-6EC8-4FB0-AB15-ED3369D17212  /newSubtask=Subtask1  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  {  "id": "5fd2a7c8-2226-4170-b5a1-b6ec5c3044c5",  "name": "Subtask1",  "done": false  } |

**UpdateSubtask**

Acest endpoint actualizaează datele unei sub sarcini.

|  |  |
| --- | --- |
| Request | Response |
| Route: PUT /api/Task/ 3D154916-6EC8-4FB0-AB15-ED3369D17212/subtask  Content-Type: application/json  Accept: \*/\*  Body:  {  "id": "5fd2a7c8-2226-4170-b5a1-b6ec5c3044c5",  "name": "Subtask1",  "done": false  } | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

**UpdateStatus**

Acest endpoint actualizaează status-ul unei sarcini.

|  |  |
| --- | --- |
| Request | Response |
| Route: PUT /api/Task/ 3D154916-6EC8-4FB0-AB15-ED3369D17212/status/0  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

**MoveToBacklog**

Acest endpoint transferă o sarcină din sprint în backlog.

|  |  |
| --- | --- |
| Request | Response |
| Route: PUT /api/Task/3D154916-6EC8-4FB0-AB15-ED3369D17212/moveToBacklog  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

**ChangeSprint**

Acest endpoint transferă o sarcină din sprint în alt sprint.

|  |  |
| --- | --- |
| Request | Response |
| Route: PUT /api/Task/3D154916-6EC8-4FB0-AB15-ED3369D17212/6FB67D8D-5AC1-451C-95EA-BFFF31F87527  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

### 4.3.6 Timesheet Controller

Este folosit pentru a efectua operații CRUD în tabelul de pontaj din baza de date. Pentru a accesa acest controller este nevoie de autorizare. În cazul în care orice endpoint din acest controller este folosit fără autorizare este returnat codul de status 401. Acesta conține următoarele endpoint-uri.

**Create**

Este folosit pentru a adăuga un nou pontaj în baza de date. Și returnează id-ul pontajului adăugat.

|  |  |
| --- | --- |
| Request | Response |
| Route: POST /api/Timesheet  Content-Type: application/json  Accept: \*/\*  Body:  {  "date": "2024-06-24",  "startTime": "12:30",  "endTime": "15:00"  } | HTTP Status Code: 201 Created  Content-Type: text/plain  Body: "8b4fe67e-5dbf-4970-95c0-90830c5caf60" |

**GetAllByUser**

Acest endpoint returnează toate pontările unui utilizator.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Timesheet  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "84f5ace8-040e-4050-a492-7f27bd795071",  "date": "8/23/2024",  "startTime": "08:00",  "endTime": "13:00"  }  ] |

**GetAllForUserByDateInterval**

Acest endpoint returnează toate pontările unui utilizator dintr-un anumit interval de zile.

|  |  |
| --- | --- |
| Request | Response |
| Route: GET /api/Timesheet/startDate  =Mon%20Jun%2024%202024  &endDate=  Sun%20Jun%2030%202024  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body:  [  {  "id": "8bcb0d04-7e2d-4db9-a8b9-5287c08be88d",  "date": "6/24/2024",  "startTime": "03:00",  "endTime": "07:00"  }  ] |

**Delete**

Este folosit pentru a șterge o pontare a utilizatorului din baza de date.

|  |  |
| --- | --- |
| Request | Response |
| Route: DELETE /api/Timesheet/8b4fe67e-5dbf-4970-95c0-90830c5caf60  Accept: \*/\* | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

**Update**

Este folosit pentru a modifica o pontare.

|  |  |
| --- | --- |
| Request | Response |
| Route: PUT /api/Timesheet  Content-Type: application/json  Accept: \*/\*  Body:  {  "id": "84f5ace8-040e-4050-a492-7f27bd795071",  "date": "2024-08-23",  "startTime": "08:00",  "endTime": "13:00"  } | HTTP Status Code: 200 OK  Content-Type: application/json  Body: true dacă a fost actualizat sau false în contrar |

# 5 Concluzii și dezvoltări ulterioare

## 5.1 Concluzii

În final, am reușit să implementăm majoritatea funcționalităților planificate inițial, dezvoltând o aplicație în care un utilizator poate dezvolta un proiect folosind metodologiile Agile.

În timpul dezvoltării acestui proiect, am acumulat numeroase cunoștințe și experiențe valoroase, exporând în detaliu anumite framework-uri. Dificultățile întâmpinate în crearea aplicației web au dus la acumularea celor mai multe cunoștințe, în special, asupra poziționării elementelor în pagină și a modului de moștenire a valorilor.

Pe parcursul dezvoltării aplicației, am întâmpinat și câteva dificultăți semnificative în crearea aplicației web care trebuia să fie sensibilă asupra mărimilor dispozitivului pe care este afișată. De asemenea, dezvoltarea unui calendar interactiv cu elemente definite dinamic a fost o provocare surprinzătoare.

## 5.2 Dezvoltări ulterioare

În viitor se poate considera dezvoltarea următoarelor funcționalități și sisteme pentru a îmbunătăți securitatea aplicației, precum și experiența utilizatorilor.

* Sistem de pontaj semi-automat – în funcție de status-ul sarcinilor-urilor la care a fost asignat și pe care le-a finalizat dar și posibilitatea de a modifica acele pontări sau de a insera pontări manual
* Sistem de logging – pentru a monitoriza și înregistra activitățile aplicației, folosind aceste înregistrări pentru îmbunătățirea performanței și pentru a remedia probleme
* Teste unitare – pentru a asigura funcționalitatea corectă a aplicației și pentru a nu adăuga bug-uri în timpul viitoarelor actualizări
* Abilitatea de a folosi aplicația web și pe dispozitivele mobile – pentru a asigura o experiență de utilizare plăcută și intuitivă
* Sistem de ștergere a utilizatorilor care nu au fost verificați – pentru a micșora dimensiunea bazei de date și a scădea timpul de căutare
* Integrarea unui sistem de diferențiere a diferențelor între fișiere
* Implementarea unei palete de culori pentru noapte
* Posibilitatea de a programa meeting-uri și de a le afișa în calendar

Aceste îmbunătățiri vor contribui la creșterea fiabilității și atractivității aplicației, oferind utilizatorilor o experiență mai sigură și mai satisfăcătoare.

# Bibliografie

Conservancy, S. F. (2020, 11 13). Preluat de pe git-scm: https://www.git-scm.com/doc

Figma. (2024). *Figma Learn.* Preluat de pe https://help.figma.com/hc/en-us

Fork. (2020). *Fork - a git client for Mac and Windows.* Preluat de pe https://git-fork.com/blog/

GeeksforGeeks. (2024, 05 23). *Normal Forms in DBMS.* Preluat de pe https://www.geeksforgeeks.org/normal-forms-in-dbms/

GitHub. (2024). *GitHub Docs*. Preluat de pe GitHub: https://docs.github.com/en/get-started

GitHub, I. (2024). *GitHub Tutorial.* Preluat de pe https://docs.github.com/en/get-started/quickstart/hello-world

Martin, R. C. (2012, 08 13). *Clean Coder Blog.* Preluat de pe https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Microsoft. (2024). *ASP.NET.* Preluat de pe https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0

Microsoft. (2024). *C#.* Preluat de pe https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

*SQLite.* (2024). Preluat de pe https://www.sqlite.org/docs.html

*TypeScript.* (2024). Preluat de pe https://www.typescriptlang.org/docs/handbook/variable-declarations.html

Wikipedia. (2024, 06 9). *Angular (web framework).* Preluat de pe https://en.wikipedia.org/wiki/Angular\_(web\_framework)

Wikipedia. (2024, 05 21). *Figma.* Preluat de pe https://en.wikipedia.org/wiki/Figma