

Факултет техничких наука у Чачку
Универзитета у Крагујевцу

Анђела Ђоковић

Развој система за управљање пројектима

Дипломски рад

Факултет техничких наука у Чачку
Универзитета у Крагујевцу



Развој система за управљање пројектима
дипломски рад

Врста студија: Основне академске студије

Назив студијског програма: Информационе технологије

Предмет: Софтверско инжењерство

Студент:

Анђела Ђоковић 33/2020

Руководилац рада:

Проф. др Марија Благојевић, редовни професор

Commented [MB1]: Додати поред "редовни проф. "

Чачак, септембар 2024.

1. РЕЗИМЕ

Овај завршни рад представља развој и имплементацију система за управљање пројектима, креираног коришћењем MERN технологија (MongoDB, Express, React, Node.js). Циљ овог система је олакшавање управљања пројектима, тимовима и задацима у различитим секторима. Систем укључује функционалности као што су креирање, ажурирање, брисање пројеката. Током развоја, коришћене су најбоље праксе за сигурност и перформансе, укључујући JWT аутентификацију и Tailwind CSS за стилизовање. Овај рад пружа детаљан увид у архитектуру, развој и функционалности система, као примере коришћења у различитим окружењима.

Кључне речи:

- Апликација
- MERN технологија
- Корисник
- Пројекат
- Задаци
- Тимови
- Развој софтвера

САДРЖАЈ

1. РЕЗИМЕ	3
2. УВОД	1
3. ОПИС ПРОЈЕКТА	2
3.1. Циљеви и обим пројекта	2
3.2. Захтеви и ограничења	2
4. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ	4
4.1. MongoDB	4
4.2. Express.js и Node.js	4
4.2.1. Архитектонски концепти	5
4.3. React.js	6
4.4. Tailwind CSS	6
5. АРХИТЕКТУРА СИСТЕМА	8
5.1. Преглед архитектуре	8
5.2. Комуникација између компоненти	8
5.3. Обезбеђивање сигурности	9
6. БАЗА ПОДАТАКА	13
6.1. Дизајн базе података	13
6.2. Повезивање са базом података	13
6.3. Модели података	14
6.3.1. User модел	14
6.3.2. Project модел	15
6.3.3. Task модел	16
6.3.4. Team модел	16
6.3.5. Activity модел	17
7. ФУНКЦИОНАЛНОСТИ СИСТЕМА	18
7.1. Пријава и регистрација корисника	18
7.1.1. Login	18
7.1.2. Sign Up	22
7.2. Dashboard	26
7.3. Управљање пројектима	27

7.3.1. Креирање пројекта	27
7.3.2. Брисање пројекта.....	30
7.3.3. Ажурирање пројекта.....	32
7.3.4. Детаљи о пројекту.....	34
7.4. Управљање тимовима	35
7.4.1. Креирање тимова	35
7.4.2. Брисање тимова.....	39
7.4.3. Ажурирање тимова	39
7.4.4. Детаљи о тиму	40
7.5. Управљање задацима	41
7.5.1. Креирање задатака.....	42
7.5.2. Брисање задатака	43
7.5.3. Ажурирање задатака.....	43
7.5.4. Детаљи о задатку.....	44
8. ЗАКЉУЧАК	47
РЕФЕРЕНЦЕ	48

2. УВОД

Управљање пројектима представља кључни аспект пословних процеса у многим секторима. Ефикасна организација тимова и задатака може значајно утицати на успешност пројеката и укупну продуктивност. Са растућим захтевима за брзом и флексибилном испоруком решења, потреба за софистицираним алатима за управљањем пројектима постаје све израженија.

Овај завршни рад истражује управљање пројектима коришћењем MERN технологија, који су међу најпопуларнијим за развој модерних веб апликација. Систем је дизајниран тако да буде применљив у различитим пословним окружењима, омогућавајући корисницима да лако креирају, ажурирају и прате пројекте и задатке, управљају тимовима и корисничким улозима.

Преглед значаја управљања пројектима и примене софтверских решења у различитим секторима објашњава потребу за развојем оваквог система и предности које доноси у смислу побољшања ефикасности организације рада. Разматрају се изазови са којима се сусрећу тимови и менаџери у свакодневном раду и како један добро осмишљен систем може допринети превазилажењу тих изазова.

Такође, важно је напоменути коришћене технологије и њихову улогу у развоју система. MongoDB је одабран као база података због своје флексибилности и могућности руковања великим количинама података. Express и Node.js омогућавају брз и ефикасан развој серверског дела апликације, док је React коришћен за развој интерактивног и кориснички пријатног интерфејса. Tailwind CSS је изабран за стилизовање због своје ефикасности и могућности лаке примене.

Поред техничких аспеката, важно је истаћи да овај систем омогућава бољу координацију и комуникацију међу члановима тимова, што је од суштинског значаја за успех било ког пројекта. Систем пружа алате за праћење напретка, идентификацију потенцијалних проблема и њихово благовремено решавање. Користећи овај приступ, организације могу постићи већу ефикасност и боље управљати својим ресурсима.

Овај рад има за циљ да пружи свеобухватан преглед развоја и примене система за управљање пројектима, истичући његове предности и значај за различите пословне секторе. Кроз детаљну анализу и примере из праксе, приказано је како овај систем може унапредити пословне процесе и допринети успешнијем извршавању пројеката.

3. ОПИС ПРОЈЕКТА

3.1. Циљеви и обим пројекта

Главни циљ овог система је да омогући ефикасно управљање пројектима, тимовима и задацима у различитим пословним окружењима. Систем је креиран тако да кориснику обезбеди алат који ће им омогућити да лакше организују свој рад, прате напредак, управљају ресурсима, и доносе информисане одлуке током трајања пројекта. Ово се постиже путем интуитивног корисничког интерфејса, прилагођеног како почетницима, тако и искусним корисницима, са фокусом на повећање продуктивности и унапређење комуникације унутар тимова.

Специфични циљеви система:

- Омогућити корисницима да креирају нове пројекте, управљају постојећим и прате њихов напредак кроз интерфејс који је лак за употребу.
- Пружити алате за ефикасно управљање тимовима, укључујући додељивање задатака и улога чланова тима.
- Омогућити праћење и ажурирање задатака у оквиру пројеката, укључујући промену статуса, додељивање приоритета и рока завршетка.
- Обезбедити функције за аналитику ради бољег праћења перформанси и напретка пројеката.
- Коришћењем JWT аутентификације, осигурати сигурност приступа систему и заштиту корисничких података.

Опсег: Систем је намењен за употребу у малим и средњим предузећима која се баве пројектно оријентисаним радом, као и у великим организацијама које управљају више пројеката истовремено. Он је такође применљив у различитим индустријама, укључујући ИТ сектор, грађевинарство, маркетинг, образовање, и друге области где је важно ефикасно управљање пројектима.

Домен примене: Систем је дизајниран да подржи различите врсте пројеката, од мањих тимских задатака до великих и сложених пројеката који укључују више тимова и различите фазе развоја. Систем је такође флексибилан и може се прилагодити специфичним потребама корисника, укључујући интеграцију са другим алатима и системима које организације већ користе.

3.2. Захтеви и ограничења

Функционални захтеви описују конкретне функције и способности које систем треба да обезбеди. У контексту система за управљање пројектима функционални захтеви могу укључивати:

- Управљање пројектима
- Управљање тимовима
- Управљање задатака
- Праћење аналитике
- Аутентификација и сигурност

Функционални захтеви овог система укључују различите могућности попут креирања и управљања пројектима, где корисници могу дефинисати циљеве и рокове, додељивати

тимове и задатке. Управљање тимовима омогућава формирање тимова, доделу улога и праћење учинка чланова тима. Праћење задатака је још једна кључна функција која обезбеђује креирање, ажурирање и праћење задатака у оквиру пројеката, са могућношћу постављања приоритета и временских оквира.

Систем мора бити опремљен снажним сигурносним механизмима, као што је JWT аутентификација, која обезбеђује заштиту података и корисника. Функционалности за праћење аналитике доприносе ефикасном праћењу напретка и перформанси пројеката.

Нефункционални захтеви описују конкретне функције и способности које систем треба да обезбеди контексту управљања пројектима, функционални захтеви могу укључивати.

- Перформансе
- Скалабилност
- Безбедност
- Корисничко искуство
- Прилагодљивост

Нефункционални захтеви се односе на перформансе и скалабилност система, како би могао подржати велики број корисника и података. Поред тога, корисничко искуство треба да буде оптимизовано кроз интуитиван и једноставан интерфејс.

Техничка ограничења се односе на факторе који могу утицати на начин на који је систем дизајниран и развијен:

- Зависност од технологија
- Интеграција са постојећим системима
- Хардверски и софтверски захтеви

Техничка ограничења укључују зависност од MERN технологија, што утиче на карактеристике перформанси и функционалности система. Потенцијални изазови у интеграцији са другим системима могу утицати на проширивост система. Такође, хардверски и софтверски захтеви могу утицати на стабилност и перформансе система у зависности од капацитета инфраструктуре коју организација користи.

Организациона решења се односе на факторе унутар организације који могу утицати на успешност имплементације и употребе система:

- Буџет
- Обученост корисника
- Култура организације

Организациона ограничења попут буџета и обучености корисника могу значајно утицати на успех имплементације и коришћења система. Одговарајућа обука и подршка корисницима, као и култура организације која је отворена за нове алате, такође су кључни фактори у постизању оптималних резултата у коришћењу овог система.

4. КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

4.1. MongoDB

MongoDB је документно-оријентисана NoSQL база података која користи флексибилан модел докумената уместо традиционалних релационих табела [1]. За разлику од релационих база података, MongoDB складишти податке у JSON-сличним структурама, званим BSON (Binary JSON), омогућавајући складиштење сложених и угнеждених података на једноставан и директан начин [1]. Овај приступ омогућава високу флексибилност и могућност брзих измена структуре података без потребе за миграцијом [1]. MongoDB је такође познат по својој скалабилности, омогућавајући расподелу података преко више сервера ради бољих перформанси и доступности [1]. MongoDB је такође познат по својој скалабилности, омогућавајући расподелу података преко више сервера ради бољих перформанси и доступности.

Разлози за избор MongoDB:

Флексибилност: MongoDB омогућава развој апликација које се брзо мењају и еволуирају. Његова шема омогућава додавање нових поља без потребе за сложеним миграцијама података.

Скалабилност: Омогућава обраду великих количина података и повећан број корисника без губитка перформанси.

MongoDB Atlas: Као услуга у облаку, MongoDB Atlas обезбеђује аутоматизовану скалабилност, управљање базом, и подршку за глобалну дистрибуцију података, што значи да се база може лако проширити и адаптирати новим захтевима без значајног административног напора.

Безбедност и доступност: Atlas пружа напредне сигурносне мере, укључујући шифровање података, интеграција са сервисима за аутентификацију, и подршку за *disaster recovery* планове.

У оквиру овог пројекта коришћена је MongoDB Atlas платформа за управљање базом података. MongoDB Atlas је *cloud-hosted* решење која пружа потпуну аутоматизацију у управљању MongoDB инстанцама. Atlas омогућава лаку конфигурацију, аутоматско скалирање и глобалну дистрибуцију података, чиме се значајно смањује административни напор потребан за одржавање базе података [1][6].

4.2. Express.js и Node.js

У оквиру пројекта “Развој система за управљање пројектима”, Node.js и Express.js играју кључну улогу у развоју *backend* дела система. Node.js, као платформа омогућава извршавање JavaScript кода на серверу, пружа основу за развој брзих и скалабилних серверских апликација [2]. У ово систему Node.js коришћен за руковање захтевима који долазе од корисника, брду података, и комуникацију са MongoDB базом података.

Express.js, као минималистички фрејмворк за Node.js, омогућио је лако и брзо креирање API сервиса који подржавају све функционалности система[2]. Конкретно, Express је омогућио дефинисање рута за креирање, ажурирање и брисање пројеката, тимова и задатака, као и руковање корисничким аутентификацијама и овлашћењима. Ова структура је

омогућила развој модула који су независни и лако одрживи, што значајно убрзало процес развоја и омогућило лаку интеграцију нових функционалности.

На слици 1 приказан је MERN стек, који укључује MongoDB, Express.js, React.js и Node.js, и представља основну архитектуру која је коришћена за развој наше апликације.



Слика 1. MERN стек

4.2.1. Архитектонски концепти

Током развоја поменутог система коришћени су различити архитектонски концепти како би се обезбедила ефикасност и поузданост система. Један од кључних концепата је **асинхронно програмирање** које омогућава Node.js-у да обрађује захтеве истовремено. Ово значи да систем може да одговори на велики број корисничких захтева без успоравања, што је посебно важно у контексту оваквог система где многи корисници истовремено креирају или ажурирају пројекте и задатке[3].

Express middleware-и су играли важну улогу у имплементацији кључних функционалности као што су аутентификација корисника, валидација података и обрада грешака. Сваки пут када корисник покуша да приступи заштићеним рутама, *middleware* за аутентификацију проверава валидност корисничког JWT токена и одређује да ли корисник има права приступа том ресурсу.

Архитектура система је дизајнирана као **модуларна**, што омогућава да се различите функционалности, попут управљања пројектима, тимовима и задацима, организују у посебне модуле. Овај приступ олакшава интеграцију нових функција у будућности и повећава читљивост и одрживост кода.

Што се тиче безбедности, кориснички подаци су били од примарног значаја. **JWT аутентификација** је коришћена за сигурно управљање корисничким сесијама, обезбеђујући да само овлашћени корисници могу приступити функционалностима попут измене пројеката или приступа задацима тимова. Поред тога, валидација улазних података је примењена како би се заштитило од потенцијално опасних захтева.

Сигурност API-ја је такође била од кључног значаја. Сви API захтеви у систему су дизајнирани са фокусом на безбедност, са додатним слојевима заштите који укључују ограничавање приступа осетљивим подацима само овлашћеним корисницима и употребу шифровања како би се осигурало да су подаци безбедни током преноса[7].

Кроз ове архитектонске концепте и методологије, Express.js и Node.js су омогућили изградњу поузданог и сигурног *backend* дела система за управљање пројектима, који

Commented [MB2]: На сваку слику се треба позвати и у тексту. На пример, на слици 1 приказано је...

ефикасно обрађује корисничке захтеве и обезбеђује високу заштиту података. У овом пројекту, сигурност корисничких података је била од примарног значаја.

4.3. React.js

React.js је коришћен као основна библиотека за развој *frontend*-а, омогућавајући изградњу динамичког и интерактивног корисничког интерфејса који је интуитиван и лак за коришћење[4]. У овом пројекту, је коришћен за креирање различитих интерфејсних компоненти које омогућавају корисницима да управљају пројектима, тимовима и задацима на једноставан начин.

Свака страница у систему за управљање пројектима дизајнирана је као скуп независних и поново употребљивих React компоненти, што смањује комплексност апликације и олакшава њено одржавање и надоградњу. Ове компоненте су организоване тако да омогућавају лако управљање подацима и ажурирање корисничког интерфејса у складу са променама у стању апликације. Кључна предност React-а је употреба *state* објекта за праћење динамичких података, као што су тренутно активни пројекти, листе задатака, корисници и тимови[4]. Основни градивни елементи апликације су управо те React компоненте, које, поред свог специфичног задатка, имају сопствени *state*, стилове и методе. На пример, компонента *AddProjectCard* омогућава корисницима да додају нове пројекте и аутоматски ажурирају листу пројеката након додавања, чиме се постиже интуитиван и ефикасан кориснички интерфејс.

React је идеалан за развој SPA (*Single Page Application*) апликација, јер омогућава динамично ажурирање садржаја без потребе за поновним учитавањем целе странице[4]. Овај приступ омогућава брзу и интерактивну корисничку навигацију, где корисници могу без прекида управљати различитим аспектима пројекта, као што су додавање нових тимова или ажурирање задатака.

React омогућава брзо извршавање апликације кроз виртуелни DOM (*Document Object Model*), што значајно побољшава перформансе апликације, јер минимизира потребу за манипулацијом стварног DOM-а. Овај механизам, заједно са способношћу React-а да моментално реагује на промене у корисничким улазима, побољшава корисничко искуство, што је кључно за апликацију као што је систем за управљање пројектима која обрађује велики број захтева[4].

Захваљујући својој компонентној структури, React омогућава развој апликација које су лако прошириве и одрживе. Свака компонента је самостална, што олакшава развијање и тестирање нових функционалности. У овом систему, React је омогућио корисницима брз, интерактиван и интуитиван начин за управљање пројектима, тимовима и задацима, док је развојном тиму омогућио ефикасно одржавање и проширивање апликације.

4.4. Tailwind CSS

Tailwind CSS коришћен је за стилизовање апликације и креирање интуитивног и визуелног атрактивног корисничког интерфејса. Tailwind CSS је утицао на ефикасност развоја интерфејса, омогућавајући брзо и флексибилно примењивање стила директно у HTML, без потребе за писањем додатних CSS класа и стилова.

Једна од главних предности Tailwind-а је његова утилитарна природа, која омогућава креирање сложених и прилагођених дизајна без конфликта у именовању класа или

стилизицији[5]. Ово је значајно смањило време развоја и допринело конзистентности и одрживости стила кроз читаву апликацију. У систему за управљање пројектима, Tailwind CSS је примењен за креирање чистог и функционалног интерфејса, који омогућава корисницима да брзо и лако управљају пројектима, тимовима и задацима.

5. АРХИТЕКТУРА СИСТЕМА

5.1. Преглед архитектуре

Архитектура систем за управљање пројектима заснована је на моделу клијент-сервер и користи MERN стек технологију. Систем је организован у три основна слоја: кориснички интерфејс (frontend), сервер (backend) и база података.

Frontend: Кориснички интерфејс је реализован као *Single Page Application* (SPA), омогућавајући корисницима брзо и интерактивно управљање пројектима, тимовима и задацима. Компонентна структура React-а обезбеђује лако управљање подацима и динамичко ажурирање интерфејса.

Backend: Сервер је одговоран за обраду корисничких захтева, интеграцију са базом података и управљање сесијама и аутентификацијом. Овај слој користи REST API за ефикасну комуникацију са *frontend*-ом и базом података.

База података: Подаци о пројектима, тимовима, задацима и корисницима чувају се у MongoDB-у, који обезбеђује флексибилност и скалабилност. Коришћење MongoDB Atlas-а додатно побољшава доступност и управљање подацима.

Основни модули и њихове међусобне интеракције:

- Кориснички модул
- Пројектни модул
- Тимски модул
- Задатак модул
- Модул активности

Сваки од ових модула игра кључну улогу у функционисању система за управљање пројектима. Кориснички модул је одговоран за безбедност и контролу приступа, управљајући регистрацијом корисника и њиховим улогама. Пројектни модул обухвата све функције везане за управљање пројектима, омогућавајући корисницима да креирају и ажурирају пројекте, као и да додељују тимове и задатке. Тимски модул пружа алате за креирање и организацију тимова, чиме омогућава сарадњу између чланова тима и њихово повезивање са одговарајућим пројектима. Модул задатака омогућава ефикасно праћење задатака у оквиру пројеката, укључујући праћење статуса и напретка, што је од суштинске важности за успешну реализацију пројеката. На крају, модул активности прати активности корисника.

Ови модули сарађују кроз API-е, обезбеђујући несметану размену података и функционалну интеграцију свих делова система, чиме се постиже ефикасност и конзистентност у раду.

5.2. Комуникација између компоненти

Комуникација између *frontend*-а и *backend*-а у овом пројекту од кључног је значаја за правилно функционисање апликације. Ова комуникација се одвија у реалном времену, где *frontend* компонента, развијена у React.js, шаље HTTP захтеве серверу за приступ, модификацију или брисање података, а сервер, покренут на Node.js и Express.js, одговара са траженим информацијама или потврдом успешне акције.

REST (Representational State Transfer) је архитектонски стил који дефинише начин на који веб сервиси комуницирају путем HTTP протокола. RESTful API је интерфејс који омогућава комуникацију између клијентске и серверске стране, користећи основне HTTP методе као што су 'GET', 'POST', 'PUT', и 'DELETE' за различите операције над ресурсима. У овом пројекту, сваки ресурс, попут пројеката, тимова или задатака, је представљен путем URL-а, а подаци се размењују у JSON формату. RESTful API је дизајниран да буде једноставан, скалабилан и лак за имплементацију, чиме омогућава брзу и ефикасну комуникацију између различитих компоненти система.

У систему за управљање пројектима, RESTful API омогућава размену података између *frontend* и *backend* компоненти. На пример, када корисник жели да дода нови пројекат, React.js компонента шаље 'POST' захтев на руту '/api/projects/add-project' са подацима о пројекту. Сервер, који користи Node.js и Express.js, обрађује овај захтев, чува податке у MongoDB бази података и враћа JSON одговор са детаљима новог пројекта.

Такође, за преглед постојећих пројеката, React компонента шаље 'GET' захтев на '/api/projects/get-projects', а сервер враћа листу пројеката у JSON формату. Овај приступ омогућава структурирану и ефикасну комуникацију између *frontend* и *backend* компоненти, што значајно унапређује брзину и перформансе апликације, омогућавајући корисницима да управљају пројектима без потребе за ручним управљањем подацима.

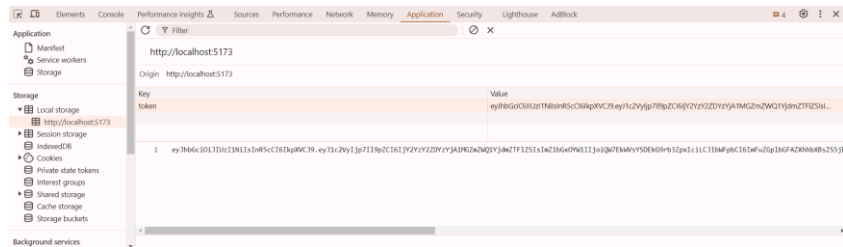
Детаљни опис рада апликације и пример кода биће дати у наставку рада, где ће бити представљени одговарајући делови кода за имплементацију ових функција.

5.3. Обезбеђивање сигурности

У оквиру система за управљање пројектима, један од најважнијих аспеката је обезбеђивање сигурности корисничких података и заштита од неовлашћеног приступа. Због тога је имплементација адекватног аутентификационог система кључна за очување интегритета и приватности података. У овом систему користи се *JSON Web Token (JWT)* технологија за аутентификацију, која омогућава сигуран приступ ресурсима и функционалностима апликације.

JWT је популаран метод за аутентификацију који се користи у модерним веб апликацијама. JWT токени су JSON објекти који се криптографски потписују и који садрже информације о кориснику, као и временску ознаку важења токена. Главна предност овог приступа је што омогућава *stateless* аутентификацију, где сервер не мора да чува сесије или стање корисника, што значајно смањује оптерећење на серверу и побољшава перформансе[7].

Када се корисник успешно пријави у систем, сервер генерише JWT токен који се затим шаље клијенту. Клијент чува овај токен у локалној меморији или колачићима, како је приказано на слици 2. Токен се додаје уз сваки наредни захтев који се упућује серверу. На тај начин, сервер може да верификује аутентичност корисника без потребе за поновним пријављивањем, што побољшава сигурност и корисничко искуство [7].



Слика 2. Local Storage

У овом систему, клијентски код користи *axios* за слање HTTP захтева ка серверу[8]. Токен, који се генерише након успешне аутентификације, чува се на клијентској страни у *localStorage*. Тај токен се потом додаје у заглавље сваког захтева као *Bearer* токен, чиме се осигурава да сервер увек добија доказ о аутентичности корисника пре обраде захтева. Поред тога, ако сервер врати статус 401, што указује на неовлашћен приступ, систем ће аутоматски обрисати токен и преусмерити корисника на страницу за пријаву.

Конфигурације на страни клијента:

```
import axios from 'axios';
```

```
const axiosInstance = axios.create({
  baseURL: 'http://localhost:8000/api',
  headers: {
    'Content-Type': 'application/json'
  }
});

axiosInstance.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers['Authorization'] = `Bearer ${token}`;
  }
  return config;
}, (error) => {
  return Promise.reject(error);
});
```

```

axiosInstance.interceptors.response.use(
  response => response,
  error => {
    if (error.response && error.response.status === 401) {
      localStorage.clear();
      window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);

```

```
}  
);
```

```
export default axiosInstance;
```

Са серверске стране, сваки захтев који стигне мора бити проверен. Ово се ради помоћу *middleware* функције која користи JWT библиотеку да декодира токен и провери његову исправност[6]. У случају невалидног или истеклог токена, приступ се аутоматски одбија, што спречава било какву злоупотребу.

Middleware функција за аутентификацију:

```
const jwt = require("jsonwebtoken");  
const User = require("../models/user.model");  
  
const authenticateToken = (req, res, next) => {  
  const authHeader = req.headers["authorization"];  
  const token = authHeader && authHeader.split(" ")[1];  
  
  if (!token) {  
    return res.status(401).json({ message: "Access denied. No token provided." });  
  }  
  
  jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, decoded) => {  
    if (err) {  
      return res.status(403).json({ message: "Invalid token." });  
    }  
    req.user = decoded.user;  
    next();  
  });  
};  
  
module.exports = { authenticateToken };
```

Ова функција прво проверава да ли је токен присутан у захтеву. Ако није, враћа се одговор са статусом 401 и поруком да је приступ одбијен. Ако токен постоји, верификује се његова исправност користећи тајни кључ (ACCESS_TOKEN_SECRET), који је чуван на серверу. У случају успеха, декодирани подаци о кориснику се чувају у *req.user* и прослеђују се наредним *middleware*-има или контролерима.

Уз коришћење JWT-а, додатне мере заштите су кључне за осигурање комплетне сигурности апликације. Токени би требало да имају ограничен период важења како би се смањио ризик од злоупотребе у случају да токен буде компромитован. Да би корисници могли да остану пријављени без потребе за поновним уносом података, користе се рефреш токени који омогућавају генерисање нових приступних токена.

Сигурно чување токена је такође важно; они би требало да се чувају на сигуран начин, било у *HttpOnly* колачићима или у сигурној меморији, како би се спречили XSS напади и крађа токена. Секретни кључеви који се користе за потписивање токена морају бити

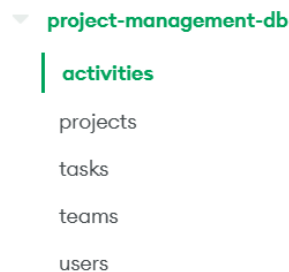
безбедно чувани и криптографски хеширани, чиме се спречава њихов неовлашћен приступ. У комбинацији са наведеним мерама, JWT аутентификација омогућава флексибилан, сигуран и ефикасан приступ у управљању корисничким приступом у систему, штитећи како корисничке податке, тако и ресурсе целокупне апликације.

6. БАЗА ПОДАТАКА

6.1. Дизајн базе података

Дизајн базе података заснован је на MongoDB, који представља нерелациони систем за управљање базама података заснован на документима. За овај пројекат изабран је MongoDB због његове флексибилности и могућности брзог управљања великом количином податка у JSON сличном формату. Ова база података омогућава ефикасно складиштење података у колекцијама, где свака колекција представља један ентитет.

Главне колекције у бази података, као што је приказано на слици 3, су Users, Projects, Tasks, Teams, и Activities. Свака од ових колекција има дефинисан модел који одређује структуру података за сваки документ унутар колекције. Овакав начин организације омогућава лаку манипулацију и модификацију података, као и једноставно проширивање система новим функционалностима.



Слика 3. Приказ колекција у бази података

6.2. Повезивање са базом података

Пројекат користи MongoDB као систем за управљање базама података. Повезивање са MongoDB базом података се врши уз помоћ *mongoose* библиотеке.

Пример конфигурације сервера и повезивања са MongoDB базом података:

```
require("dotenv").config();
const config = require("./config.json");
const mongoose = require("mongoose");
const express = require("express");
const cors = require("cors");

const app = express();
const PORT = process.env.PORT || 8000;

app.use(express.json());
app.use(cors({ origin: "*" }));

mongoose.connect(config.connectionString, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
```

```

.catch(err => console.error('Failed to connect to MongoDB:', err));

const userRoutes = require("./routes/user.routes");
const taskRoutes = require("./routes/task.routes");
const teamRoutes = require("./routes/team.routes");
const projectRoutes = require("./routes/project.routes");
const dashboardRoutes = require("./routes/dashboard.routes");
const searchRoutes = require("./routes/search.routes");
const fileRoutes = require("./routes/file.routes");
app.use("/api", searchRoutes);
app.use("/api/users", userRoutes);
app.use("/api/tasks", taskRoutes);
app.use("/api/teams", teamRoutes);
app.use("/api/projects", projectRoutes);
app.use("/api", dashboardRoutes);
app.use('/api', fileRoutes);

app.get("/", (req, res) => {
  res.json({ data: "hello" });
});

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

module.exports = app;

```

Конфигурација и читавање окружења: *dotenv* библиотека се користи за читавање конфигурацијских података из *.env* фајла. *config.json* садржи *connectionString* за повезивање на MongoDB базу података.

Повезивање са MongoDB: *mongoose.connect* метод се користи за повезивање на MongoDB базу података са опцијама за нови парсер и уједињену топологију.

Постављање руте: Руте су дефинисане за управљање различитим захтевима, као што су корисници, задаци, тимови, пројекти, и управљање датотекама.

Ова конфигурација осигурава да апликација може да комуницира са MongoDB базом података и обавља све потребне операције унутар система.

6.3. Модели података

6.3.1. User модел

Колекција *Users* чува све битне информације о корисницима система. Документи у овој колекцији укључују основне податке као што су име, емаил, лозинка и улога корисника. Поред тога, корисници могу бити повезани са пројектима и тимовима у којима учествују, као и са задатцима које извршавају. Ова колекција омогућава управљање корисничким информацијама и њиховим улогама у систему.

Шема за *User* модел:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const userSchema = new Schema({
  fullName: { type: String },
  email: { type: String },
  password: { type: String },
  createdAt: { type: Date, default: Date.now },
  role: { type: String, default: 'User' },
});

module.exports = mongoose.model("User", userSchema);
```

6.3.2. Project модел

Колекција *Projects* чува све важне информације о пројектима у систему. Документи у овој колекцији описују сваки пројекат и укључују податке као што су назив пројекта, опис, датуми почетка и завршетка, статус пројекта, приоритет, као и кориснике и тимове који су задужени за пројекат. Поред тога, пројекти могу имати повезане задатке и додате датотеке. Ова организација омогућава преглед и управљање свим аспектима пројекта на једном месту.

Шема за *Project* модел:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const projectSchema = new Schema({
  name: { type: String, required: true },
  description: { type: String, required: true },
  startDate: { type: Date, required: true },
  dueDate: { type: Date, required: true },
  status: { type: String, enum: ['Not Started', 'In Progress', 'Completed', 'On Hold'], default: 'Not Started' },
  priority: { type: String, enum: ['High', 'Medium', 'Low'], default: 'Medium' },
  createdBy: { type: Schema.Types.ObjectId, ref: 'User' },
  assignedTo: [{ type: Schema.Types.ObjectId, ref: 'User' }],
  teams: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Team' }],
  tasks: [{ type: Schema.Types.ObjectId, ref: 'Task' }],
  files: [{
    fileName: String,
    filePath: String
  }]
});

const Project = mongoose.model('Project', projectSchema);
module.exports = Project;
```

6.3.3. Task модел

Колекција *Tasks* чува све податке о задатцима који су део пројеката. Сваки задатак укључује наслов, опис, статус (на пример, "To Do", "In Progress", "Completed"), датум завршетка, кориснике којима је задатак додељен, и корисника који је креирао задатак. Такође, задаци могу имати коментаре који омогућавају бољу комуникацију и праћење напретка. Ова колекција омогућава детаљно праћење свих задатака у оквиру пројеката.

Шема за *Task* модел:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const commentSchema = new Schema({
  user: { type: Schema.Types.ObjectId, ref: 'User', required: true },
  comment: { type: String, required: true },
  timestamp: { type: Date, default: Date.now }
});

const taskSchema = new Schema({
  title: { type: String, required: true },
  description: { type: String },
  status: { type: String, enum: ['To Do', 'In Progress', 'Completed'], default: 'To Do' },
  createdAt: { type: Date, default: Date.now },
  assignedTo: [{ type: Schema.Types.ObjectId, ref: 'User' }],
  dueDate: { type: Date },
  createdBy: { type: Schema.Types.ObjectId, ref: 'User' },
  priority: { type: String, enum: ['High', 'Medium', 'Low'], default: 'Low' },
  project: { type: Schema.Types.ObjectId, ref: 'Project', required: true },
  comments: [commentSchema]
});

const Task = mongoose.model('Task', taskSchema);
module.exports = Task;
```

6.3.4. Team модел

Колекција *Teams* чува све битне податке о тимовима који су укључени у пројекте. Тимови су повезани са пројектима и састоје се од корисника који заједно раде на извршењу задатака. Сваки документ у овој колекцији садржи информације као што су назив тима, опис тима, листу чланова који су део тима, као и податак о кориснику који је креирао тим. Оваква организација омогућава ефикасно управљање тимовима и њиховим улогама у оквиру различитих пројеката.

Шема за *Team* модел:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
```

```
const teamSchema = new Schema({
  name: { type: String, required: true },
  description: { type: String },
  members: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  createdBy: { type: Schema.Types.ObjectId, ref: 'User' }
});
```

```
const Team = mongoose.model('Team', teamSchema);
```

```
module.exports = Team;
```

6.3.5. Activity модел

Колекција *Activities* бележи све релевантне активности које се односе на пројекте и задатке у систему. Сваки документ у овој колекцији садржи поруку која описује активност, временски печат када је активност извршена, тип активности, акцију која је предузета, као и референце на корисника, задатак и пројекат са којима је активност повезана. Овај модел омогућава детаљно праћење акција и пружа историју промена у систему, чиме се обезбеђује транспарентност и боља контрола над радом на пројектима.

Шема за *Activity* модел:

```
const mongoose = require('mongoose');

const activitySchema = new mongoose.Schema({
  message: { type: String, required: true },
  timestamp: { type: Date, default: Date.now },
  type: { type: String, required: true },
  action: { type: String, required: true },
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  taskId: { type: mongoose.Schema.Types.ObjectId, ref: 'Task' },
  projectId: { type: mongoose.Schema.Types.ObjectId, ref: 'Project' },
});
```

```
module.exports = mongoose.model('Activity', activitySchema);
```

Ова организација базе података пружа снажну основу за управљање свим аспектима пројектног система, укључујући кориснике, пројекте, задатке, тимове и активности. MongoDB омогућава лако управљање и манипулацију подацима, што је кључно за флексибилност и раст система.

7. ФУНКЦИОНАЛНОСТИ СИСТЕМА

У овом делу рада биће детаљно описане и представљене кључне функционалности система. Овај преглед ће обухватити како су различите компоненте имплементиране, уз пратеће примере кода и илустрације које ће помоћи у бољем разумевању функционалности. Циљ је да се пружи свеобухватан увид у способности система, укључујући начин на који се његове функционалности интегришу и како доприносе испуњавању захтева корисника. Детаљно ће бити приказано како сваки део система доприноси целокупној ефикасности и корисничком искуству, наглашавајући кључне аспекте имплементације и њихов утицај на рад система.

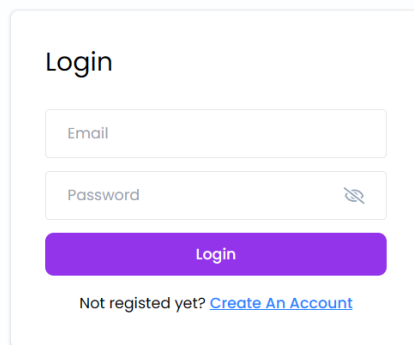
7.1. Пријава и регистрација корисника

Да би корисник приступио систему, потребно је да прође кроз процес регистрације или пријаве. Ако корисник нема налог, мораће да се региструје у систему. Регистрација укључује попуњавање образаца са основним подацима као што су име, е-маил адреса, лозинка и улога који ће бити коришћени за креирање новог корисничког налога.

Ако корисник већ има налог, треба да се пријави помоћу својих постојећих акреденцијала. Процес пријаве подразумева унос е-маил адресе и лозинке у одговарајуће форме, након чега ће систем аутоматски верификовати информације и омогућити приступ личном налогу.

Ова два процеса су основа за безбедан приступ и управљање корисничким подацима у систему, осигуравајући да само овлашћени корисници могу приступити специфичним функционалностима и информацијама.

7.1.1. Login



Слика 4. Приказ Login странице

На слици 4 приказана је компонента *Login* која омогућава корисницима да се пријаве у систем. Када корисник попуни обрасце за е-маил и лозинку кликом на *Login* дугме функција *handleLogin* обавља валидацију унетих података и шаље захтев серверу. Ако су подаци исправни и сервер потврди пријаву, приступни токен се чува у *localStorage*, а корисник се преусмерава на контролни панел (*/dashboard*). У случају грешке, одговарајућа порука се приказује кориснику.

Commented [MB3]: Исећи около ову слику тако да остане само логин форма. Увећај тада, изгледаће лепше и читљивије. Важи за све сличне слике.

```

const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);

  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();

    if (!validateEmail(email)) {
      setError("Please enter a valid email address.");
      return;
    }

    if (!password) {
      setError("Please enter the password.");
      return;
    }

    setError("");

    try {
      const response = await axiosInstance.post("/users/login", {
        email: email,
        password: password,
      });

      if (response.data && response.data.accessToken) {
        localStorage.setItem("token", response.data.accessToken);
        navigate("/dashboard");
      }
    } catch (error) {
      if (error.response && error.response.data && error.response.data.message) {
        setError(error.response.data.message);
      } else {
        setError("An unexpected error occurred. Please try again.");
      }
    }
  };

  return (
    <
    <Navbar />
  )

```



```

    <div className="flex items-center justify-center mt-28">
      <div className="w-full max-w-sm p-4 bg-white border border-gray-200 rounded-lg shadow sm:p-6 md:p-8 dark:bg-gray-800 dark:border-gray-700">
        <form onSubmit={handleLogin}>
          <h4 className="text-2xl mb-7">Login</h4>
          <input
            type="text"
            placeholder="Email"
            className="input-box"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
          <PasswordInput
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
          {error && <p className="text-red-500 text-xs pb-1">{error}</p>}
          <button type="submit" className="btn-primary">Login</button>
          <p className="text-sm text-center mt-4">
            Not registered yet? <Link to="/signUp" className="font-medium text-primary underline">Create An
Account</Link>
          </p>
        </form>
      </div>
    </div>
  );
};

```

`export default Login;`

Ова функција обрађује захтеве за пријаву корисника на серверској страни. Проверавање корисничких акредитива и генерација приступног токена је део овог процеса.

```

const User = require("../models/user.model");
const jwt = require("jsonwebtoken");

exports.login = async (req, res) => {
  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(400).json({ error: true, message: "All fields are required" });
  }

  try {
    const userInfo = await User.findOne({ email });

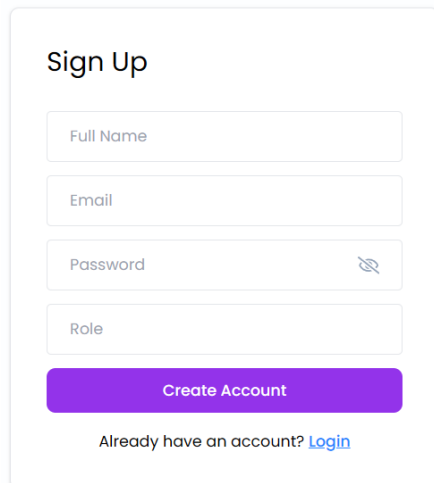
```

```
if (!userInfo) {
  return res.status(400).json({ message: "User not found" });
}

if (userInfo.password === password) {
  const user = { user: userInfo };
  const userAccessToken = jwt.sign(user, process.env.ACCESS_TOKEN_SECRET, {
    expiresIn: "36000m"
  });

  return res.json({
    error: false,
    message: "Login Successful",
    email,
    accessToken: userAccessToken
  });
} else {
  return res.status(400).json({
    error: true,
    message: "Invalid Credentials"
  });
}
} catch (error) {
  return res.status(500).json({ error: true, message: "Internal Server Error" });
}
};
};
```

7.1.2. Sign Up



The image shows a 'Sign Up' form. It has four input fields: 'Full Name', 'Email', 'Password' (with a toggle icon), and 'Role'. Below these fields is a purple 'Create Account' button. At the bottom, there is a link that says 'Already have an account? Login'.

Слика 5. Приказ Sign Up странице

На слици 5 приказана је компонента *SignUp* која омогућава новим корисницима да се региструју у систем. Када корисник попуни обрасце са пуним именом, е-маил адресом, лозинком и улогом, и кликне на дугме *Create Account*, функција *handleSignUp* врши валидацију унетих података и шаље захтев серверу. Ако недостају подаци или су неважећи, кориснику се приказује одговарајућа порука о грешци. Ако су сви подаци исправни и регистрација је успешна, приступни токен се чува у *localStorage*, а корисник се преусмерава на страницу за пријаву (*/login*) страницу како би се улоговао. Ако се појави било каква грешка током процеса, корисник добија поруку о грешци.

```
const SignUp = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [role, setRole] = useState("");
  const [error, setError] = useState(null);
  const [successMessage, setSuccessMessage] = useState("");

  const navigate = useNavigate();

  const handleSignUp = async (e) => {
    e.preventDefault();

    if (!name) {
      setError("Please enter your name");
      return;
    }
  }
}
```

```

}

if (!validateEmail(email)) {
  setError("Please enter a valid email address.");
  return;
}

if (!password) {
  setError("Please enter the password");
  return;
}

if (!role) {
  setError("Please enter your role");
  return;
}

setError("");
setSuccessMessage("");

try {
  const response = await axiosInstance.post("/users/create-account", {
    fullName: name,
    email: email,
    password: password,
    role: role,
  });

  if (response.data && response.data.error) {
    setError(response.data.message);
    return;
  }

  if (response.data && response.data.accessToken) {
    localStorage.setItem("token", response.data.accessToken);
    setSuccessMessage("Account created successfully! Redirecting to login...");
    setTimeout(() => {
      navigate("/login");
    }, 2000);
  }
} catch (error) {
  if (error.response && error.response.data && error.response.data.message) {
    setError(error.response.data.message);
  } else {
    setError("An unexpected error occurred. Please try again.");
  }
}

```

```

    }
  }
};

return (
  <>
    <Navbar />
    <div className="flex items-center justify-center mt-28">
      <div className="w-full max-w-sm p-4 bg-white border border-gray-200 rounded-lg shadow sm:p-6 md:p-8 dark:bg-gray-800 dark:border-gray-700">
        <form onSubmit={handleSignUp}>
          <h4 className="text-2xl mb-7">Sign Up</h4>
          <input
            type="text"
            placeholder="Full Name"
            className="input-box"
            value={name}
            onChange={(e) => setName(e.target.value)}
          />
          <input
            type="text"
            placeholder="Email"
            className="input-box"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
          <PasswordInput
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
          <input
            type="text"
            placeholder="Role"
            className="input-box"
            value={role}
            onChange={(e) => setRole(e.target.value)}
          />
          {error && <p className="text-red-500 text-xs pb-1">{error}</p>}
          {successMessage && <p className="text-green-500 text-xs pb-1">{successMessage}</p>}
          <button type="submit" className="btn-primary">Create Account</button>
        </form>
        <p className="text-sm text-center mt-4">
          Already have an account? { " " }
          <Link to="/login" className="font-medium text-primary underline">
            Login

```

```

    </Link>
  </p>
</div>
</div>
</>
);
};

```

```
export default SignUp;
```

Функција *createAccount* управља процесом регистрације корисника. Прво, проверава да ли су сви обавезни подаци (потпуно име, е-маил адреса, лозинка и улога) доступни у захтеву. Ако недостаје било који податак или ако корисник са истом е-маил адресом већ постоји у бази, враћа се одговарајућа грешка. Ако су сви подаци валидни, функција креира новог корисника у бази података и генерише приступни токен за аутентификацију. Токен се враћа у одговору, што омогућава кориснику да приступи систему. У случају било каквих техничких грешака током обраде, враћа се порука о унутрашњој грешци.

```

const User = require("../models/user.model");
const jwt = require("jsonwebtoken");

exports.createAccount = async (req, res) => {
  const { fullName, email, password, role } = req.body;

  if (!fullName || !email || !password || !role) {
    return res.status(400).json({ error: true, message: "All fields are required" });
  }

  try {
    const isUser = await User.findOne({ email });
    if (isUser) {
      return res.status(400).json({ error: true, message: "User already exists" });
    }

    const user = new User({ fullName, email, password, role });
    await user.save();

    const accessToken = jwt.sign({ userId: user._id, role: user.role },
process.env.ACCESS_TOKEN_SECRET, {
  expiresIn: "36000m"
});

    return res.status(201).json({
      error: false,
      user,
      accessToken,
    });
  } catch (error) {
    return res.status(500).json({ error: true, message: "Internal server error" });
  }
};

```

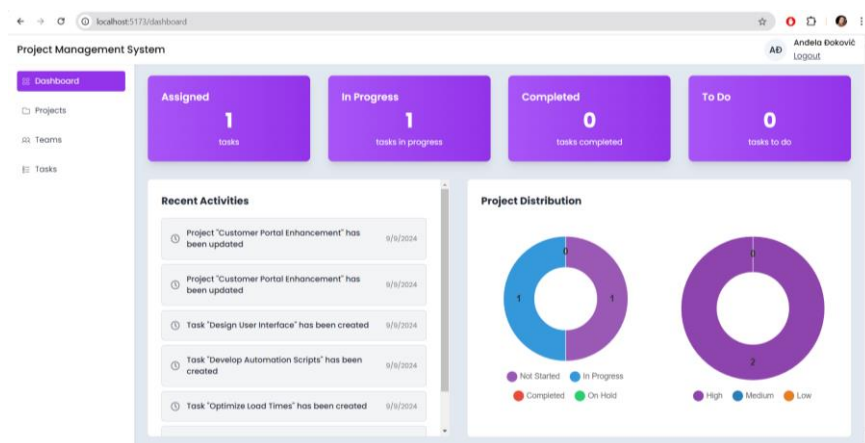
```

        message: "Registration Successful"
    });
} catch (error) {
    return res.status(500).json({ error: true, message: "Internal Server Error" });
}
};
};

```

7.2. Dashboard

Страница *Dashboard* приказује кључне информације о активностима и задацима корисника, омогућавајући им брз увид у статистике и недавне промене. Јасна организација података на овој страници, као што је приказано на слици 6, помаже корисницима да ефикасно управљају својим пројектима и задацима.



Слика 6. Dashboard

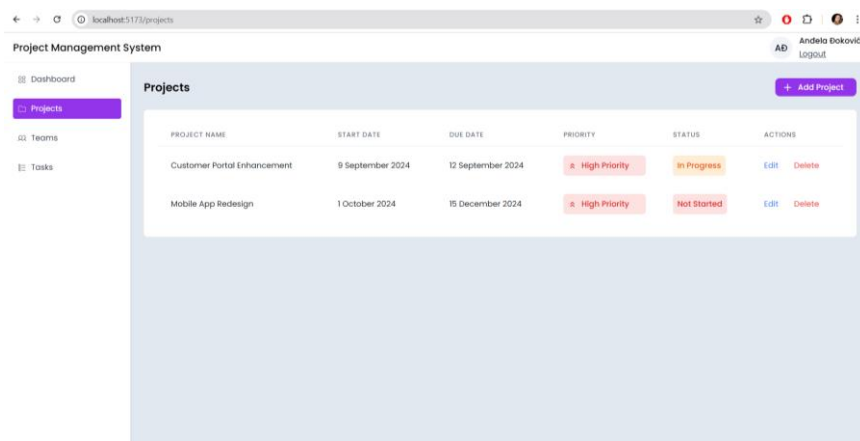
У горњем делу странице налази се *Navbar* и *Sidebar* који корисницима омогућавају лаку навигацију кроз апликацију. Испод њих, налази се низ *Dashboard Card* компоненти које визуализују важне статистичке податке. Ове картице укључују број додељених задатака, задатака у току, завршених задатака и задатака који тек треба да се обаве.

Испод статистичких картица, страница приказује *Recent Activity Card* који представља недавне активности, укључујући ажурирања и промене, помажући корисницима да буду информисани о последњим догађајима. Поред тога, налази се *Project Distribution Chart* који корисницима пружа визуелни преглед расподеле пројеката по статусу и приоритету. Овај графикон користи *Doughnut* дијаграме да прикаже колико пројеката припада сваком статусу и приоритету, помажући у бољем разумевању стања пројеката у систему.

Овај уређајен и функционалан интерфејс омогућава корисницима да брзо прегледају своје задатке, активности и пројекте, чиме се побољшава продуктивност и организација.

7.3. Управљање пројектима

Овај део апликације омогућава управљање пројектима на начин који је фокусиран на конкретне потребе корисника. Корисницима је омогућено да прегледају пројекте на којима су укључени, што им пружа преглед свих пројеката са којима су повезани. Поред тога, апликација омогућава додавање нових пројеката, као и њихово брисање и измену. Ова функционалност, приказана на слици 7, омогућава флексибилно управљање пројектима и прилагођавање задатака у складу са корисничким захтевима.



Слика 7. Приказ странице за управљање пројектима

7.3.1. Креирање пројекта

Креирање нових пројеката у апликацију омогућава корисницима да креирају пројекте уносећи основне информације као што су име, опис, датум почетка, рок, статус и приоритет. Корисници могу додати кориснике и тимове који ће бити додељени пројекту.

Процес додавања новог пројекта започиње кликом на дугме *Add Project*, што отвара форму за унос података, приказана на слици 8. Унутар ове форме корисницима су доступна сва потребна поља за унос кључних информација о пројекту. Након попуњавања поља, као што су име пројекта, опис, приоритет и статус, корисници могу изабрати кориснике и тимове који ће бити укључени у пројекат.

По завршетку попуњавања форме, корисници могу да сачувају нови пројекат кликом на дугме *Create Project*. Апликација ће обрадити податке и додати пројекат у систем. Ако се пројекат успешно дода, форма ће се затворити, освежавајући табелу пројеката са новим податком.

Add Project

Name

Description

Start Date Due Date

dd/mm/yyyy --:-- dd/mm/yyyy --:--

Priority Status

Medium Not Started

Assign Users

Search users...

Assign Teams

Search teams...

Create Project

Слика 8. Креирање новог пројекта

Контролер за креирање пројеката управља захтевима за додавање нових пројеката у базу података. Када се изврши захтев за креирање пројекта, контролер прво извлачи потребне податке из тела захтева, укључујући име, опис, датум почетка, рок, статус, приоритет, кориснике и тимове који ће бити додељени пројекту. Поред тога, контролер чува информацију о томе ко је креирао пројекат, што се добија из података о кориснику који је извршио захтев.

Контролер затим проверава да ли су сви обавезни подаци присутни. Ако неки од обавезних података недостаје, враћа се одговор са статусом 400 и поруком о потреби свих поља. Ако су сви подаци присутни, контролер ствара нови објекат пројекта, који се чува у бази података.

Након успешно креираног пројекта, контролер бележи активност у систему. Овај запис активности укључује поруку о креирању пројекта, врсту активности, акцију, идентификациони број корисника и пројекта.

На крају, контролер враћа одговор са статусом 201, потврђујући да је пројекат успешно креиран, заједно са подацима о новом пројекту. У случају грешке током креирања пројекта, враћа се одговор са статусом 500 и поруком о серверској грешци.

Обрада на серверској страни:

```
exports.addProject = async (req, res) => {
  const { name, description, startDate, dueDate, status, priority, assignedTo, teams } = req.body;
```

```

const createdBy = req.user._id;

if (!name || !description || !startDate || !dueDate || !status || !priority || !assignedTo || !teams) {
  return res.status(400).json({ message: 'All fields are required' });
}

try {
  const project = new Project({ name, description, startDate, dueDate, status, priority, assignedTo, teams,
    createdBy });
  await project.save();

  // Logging the activity
  await Activity.create({
    message: `Project "${project.name}" has been created`,
    type: 'project',
    action: 'created',
    userId: req.user._id,
    projectId: project._id
  });

  res.status(201).json({ message: 'Project created successfully', project });
} catch (error) {
  console.error('Error creating project:', error);
  res.status(500).json({ message: 'Server error' });
}
};

```

Код за обраду форме у клијентској апликацији управља подацима које корисник уноси и шаље захтев серверу за креирање или ажурирање пројекта. Прикупља податке из *formData*, укључујући идентификаторе корисника и тимова, и конвертује их у одговарајући формат. Проверава да ли су сви потребни подаци присутни, и ако недостаје неки податак, приказује грешку у конзоли. Према томе да ли се ради о креирању или ажурирању пројекта, шаље POST или PUT захтев серверу користећи *axiosInstance*. У случају успешног захтева, позива *onProjectAdded* и затвара форму; у случају грешке, приказује поруку о грешци. Клијентски код сарађује са серверским контролером који обрађује захтев, управља подацима и бележи активности, и враћа одговор који клијентски код користи за ажурирање интерфејса.

Обрада на клијентској страни:

```

const handleSubmit = async (e) => {
  e.preventDefault();

  const assignedToUserIds = formData.assignedTo.map(user => user._id);
  const assignedToTeamIds = formData.teams.map(team => team._id);

  const projectDataToSend = {

```

```

    ...formData,
    assignedTo: assignedToUserIds,
    teams: assignedToTeamIds,
  };

  const { name, description, startDate, dueDate, status, priority } = projectDataToSend;

  if (!name || !description || !status || !assignedToUserIds.length || !dueDate || !priority || !startDate) {
    console.error('All fields are required');
    return;
  }

  try {
    let response;
    if (project) {
      response = await axiosInstance.put(`/projects/update-project/${project._id}`, projectDataToSend);
    } else {
      response = await axiosInstance.post('/projects/add-project', projectDataToSend);
    }
    if (onProjectAdded) onProjectAdded();
    onClose();
  } catch (error) {
    console.error('Error saving project:', error.response?.data || error.message);
  }
};

```

Дугме за креирање пројекта:

```

<button type="submit" className="btn-primary w-32 flex items-center justify-center space-x-2 h-8 px-3">
  {project ? "Save Changes" : "Create Project"}
</button>

```

7.3.2. Брисање пројеката

Брисање пројеката врши се кликом на акцију *Delete* поред пројекта који желите да уклоните. Само корисник који је креирао пројекат има дозволу да га обрише. Ако корисник који није креирао пројекат покуша да изврши акцију брисања, систем ће приказати поруку о грешци и спречити брисање. Након успешног брисања, табела пројеката се аутоматски освежава како би одражавала промене.

Контролер за брисање пројеката одговоран је за уклањање пројеката из базе података на основу идентификатора пројекта. Када корисник иницира захтев за брисање, контролер најпре проверава да ли пројекат са датим идентификатором постоји. Уколико пројекат није пронађен, враћа се порука о томе да пројекат не постоји.

Након тога, контролер проверава да ли је корисник који покушава да обрише пројекат и његов креатор. Ако није, систем враћа поруку о забрани приступа и спречава брисање. Ако је корисник овлашћен, контролер наставља са брисањем пројекта из базе података, као и свих задатака који су повезани са тим пројектом.

Такође, контролер бележи активност у систему како би се пратила акција брисања, укључујући информације о пројекту и кориснику који је извршио брисање. На крају, враћа се порука о успешном брисању пројекта. У случају било каквих проблема током процеса, контролер ће врати грешку са одговарајућим објашњењем.

Обрада на серверској страни:

```
exports.deleteProject = async (req, res) => {
  const projectId = req.params.id;
  const userId = req.user._id;

  try {
    const project = await Project.findById(projectId);

    if (!project) {
      return res.status(404).json({ message: 'Project not found' });
    }

    if (project.createdBy.toString() !== userId.toString()) {
      return res.status(403).json({ message: 'You are not authorized to delete this project' });
    }

    await Project.findByIdAndDelete(projectId);
    await Task.deleteMany({ project: projectId });
    await Activity.create({
      message: `Project "${project.name}" has been deleted`,
      type: 'project',
      action: 'deleted',
      userId: req.user._id,
      projectId: project._id
    });

    res.status(200).json({ message: 'Project deleted successfully' });
  } catch (error) {
    console.error("Error deleting project.", error);
    res.status(500).json({ message: 'An error occurred while deleting the project', error: error.message });
  }
};
```

Функција *handleDelete* на клијентској страни управља брисањем пројекта. Када корисник иницира брисање, функција прво спречава да се догађај клика пренесе на родитељске елементе користећи *e.stopPropagation()*. Након тога, функција шаље DELETE захтев серверу на адресу `http://localhost:8000/api/projects/delete-project/${project._id}` користећи *axiosInstance*. Ако је брисање успешно, функција позива *onDelete* са идентификатором пројекта како би се ажурирао интерфејс. У случају грешке, ако сервер врати статус 403, функција приказује поруку о забрани приступа. За друге врсте грешака, приказује општу поруку о неочекиваној грешци.

Обрада на клијентској страни:

```
const handleDelete = async (e) => {
  e.stopPropagation();
  try {
    await axiosInstance.delete(`http://localhost:8000/api/projects/delete-project/${project._id}`);
    if (onDelete) {
      onDelete(project._id);
    }
  } catch (error) {
    if (error.response && error.response.status === 403) {
      window.alert(error.response.data.message);
    } else {
      window.alert("An unexpected error occurred. Please try again.");
    }
  }
};
```

Дугме за брисање:

```
<td className="px-6 py-4 whitespace-nowrap text-sm text-gray-900">
  <div className="flex gap-5">
    <button className="text-blue-500 hover:text-blue-700 mr-2" onClick={handleEdit}>
      Edit
    </button>
    <button className="text-red-500 hover:text-red-700" onClick={handleDelete}>
      Delete
    </button>
  </div>
</td>
```

7.3.3. Ажурирање пројекта

Ажурирање пројекта врши се кликом на акцију *Edit*, што отвара форму са већ попуњеним подацима о одабраном пројекту, приказана на слици 9. У овој форми, корисник може прегледати и изменити све информације везане за пројекат, укључујући име, опис, датуми, статус, приоритет, додељене кориснике и тимове.

Након што корисник изврши све потребне измене, клик на *Save Changes* покреће процес ажурирања. Подаци из форме се шаљу на сервер, где се врши ажурирање у бази података са новим информацијама. Тиме се осигурава да су све промене коректно примењене и да се пројекат ажурира са новим подацима.

Слика 9. Ажурирање података о пројекту

Контролер за ажурирање пројекта омогућава корисницима да измене постојеће пројекте у бази података. Процес почиње примањем захтева који садржи идентификатор пројекта и нове вредности за све потребне податке, као што су име, опис, датум почетка, рок, статус, приоритет, додељени корисници, тимови и задаци. Ако недостаје неки од обавезних података, контролер враћа одговор са статусом 400 и поруком о потребним пољима.

Уколико су сви подаци присутни, контролер врши ажурирање пројекта у бази података користећи идентификатор пројекта и нове вредности. Након успешног ажурирања, контролер бележи ову акцију у записима активности, указујући на то да је пројекат ажуриран. Ако се пројекат не пронађе или ако дође до грешке током ажурирања, контролер враћа одговарајуће поруке о грешци и статусне кодове.

Обрада података на серверу:

```
exports.updateProject = async (req, res) => {
  const projectId = req.params.projectId;
  const { name, description, startDate, dueDate, status, priority, assignedTo, teams, tasks } = req.body;

  if (!name || !description || !startDate || !dueDate) {
    return res.status(400).json({ message: 'Name, description, startDate, and dueDate fields are required' });
  }

  try {
```

```

const updatedProject = await Project.findByIdAndUpdate(
  projectId,
  {
    name,
    description,
    startDate,
    dueDate,
    status,
    priority,
    assignedTo,
    teams,
    tasks
  },
  { new: true }
);

if (!updatedProject) {
  return res.status(404).json({ message: 'Project not found' });
}

await Activity.create({
  message: `Project "${updatedProject.name}" has been updated`,
  type: 'project',
  action: 'updated',
  userId: req.user._id,
  projectId: updatedProject._id
});

res.status(200).json(updatedProject);
} catch (error) {
  console.error("Error updating project:", error);
  res.status(500).json({ message: 'Server error' });
}
};

```

7.3.4. Детаљи о пројекту

Кликом на пројекат отвара се страница са детаљним информацијама о пројекту, која пружа свеобухватан увид у све аспекте пројекта. На овој страници могу се видети сви тимови који су додељени пројекту, као и корисници који су ангажовани на пројекту. Поред тога, приказују се и задаци који су повезани са пројектом, што омогућава корисницима да лако прегледају статус сваког задатка.

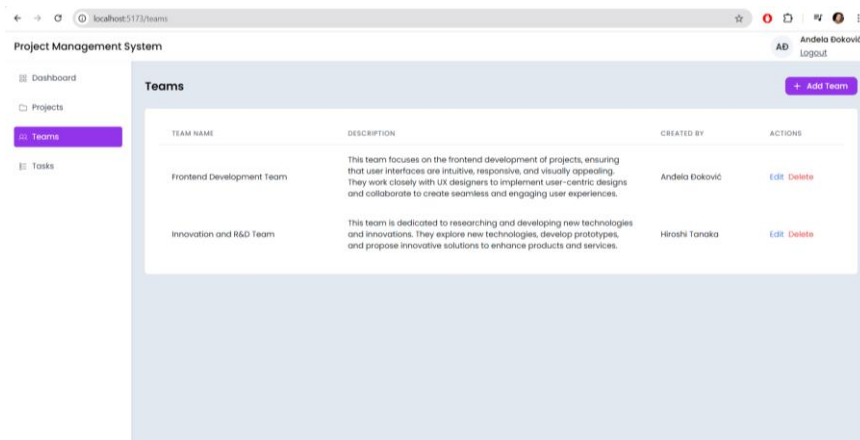
Страница такође омогућава управљање фајловима који су везани за пројекат. Корисници могу додавати нове фајлове или преузимати постојеће фајлове који су повезани са пројектом, чиме се осигурава да су сви важни документи и ресурси лако доступни. Ова функционалност доприноси бољој организацији и праћењу свих аспеката пројекта.

У оквиру ове стране, корисници такође имају могућност извршавања различитих измена на пројекту. Ово укључује уклањање или додавање корисника, додавање или уклањање задатака, као и додавање или уклањање тимова. Ове опције омогућавају лако управљање пројектом и прилагођавање према потребама тима и пројекта.

7.4. Управљање тимовима

Овај део апликације омогућава корисницима да ефикасно управљају тимовима повезаним са пројектима, пружајући им преглед свих тимова са којима раде. Корисници имају могућност да додају нове тимове, као и да измене или обришу постојеће, што омогућава лакше организовање тимских ресурса. Страница за управљање тимовима, приказана на слици 10, интегрише све ове функционалности на једноставан и интуитиван начин.

Корисници имају могућност да креирају нове тимове, дефинишу њихове чланове и додељују тимове одређеним пројектима. Такође, могу управљати постојећим тимовима, укључујући модификацију информација о тимовима и уклањање тимова када више нису потребни. Ове функције омогућавају корисницима да прилагоде тимове према потребама пројекта и осигурају да сви ресурси буду ефикасно распоређени.



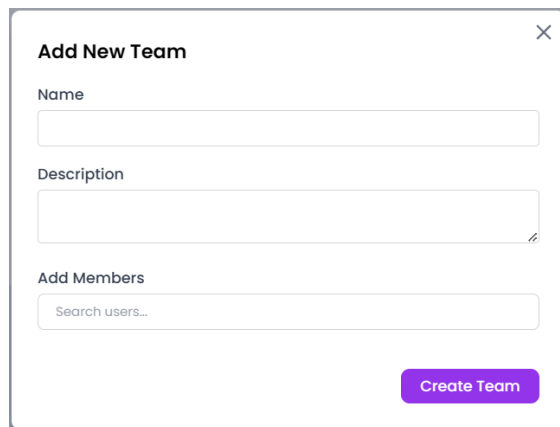
Слика 10. Приказ странице за управљање тимовима

7.4.1. Креирање тимова

Креирање нових тимова у апликацији омогућава корисницима да формирају тимове уношењем основних информација као што су име тима, опис и чланови. Корисници могу додати чланове тима путем алата за претрагу корисника, који омогућава једноставно проналажење и избор корисника који ће бити део тима. Ова функција олакшава организацију и управљање људским ресурсима у оквиру различитих пројеката.

Процес започиње кликом на дугме *Add Team*, што отвара форму за унос података о тиму. Форма, приказана на слици 11, садржи поља за име тима, опис и опцију за додавање чланова. Након што корисници попуне сва поља и додају потребне чланове, тим се креира кликом

на дугме *Create Team*. По успешном додавању, списак тимова се освежава како би приказао нови тим.



Слика 11. Креирање новог тима

Приликом додавања корисника тимовима, *UserSearchBar* компонента игра важну улогу у олакшавању претраге и селекције корисника. Корисници могу претраживати све доступне кориснике у систему и додати их тимовима користећи интуитиван интерфејс за претрагу. Компонента аутоматски шаље захтев серверу са унетим критеријумом претраге и враћа листу одговарајућих корисника, омогућавајући кориснику да изабере једног или више корисника.

Компонента користи стања као што су *searchQuery* за праћење унетог текста претраге, *userList* за приказ листе пронађених корисника и *selectedUsers* за управљање већ изабраним корисницима. Након селекције корисника, њихова имена се приказују у листи, а корисници могу бити уклоњени из тима кликом на икону за брисање.

```
const UserSearchBar = ({ assignedUsers = [], onAssignUsers = () => { }, showSelectedUsers = true }) => {
  const [searchQuery, setSearchQuery] = useState("");
  const [userList, setUserList] = useState([]);
  const [showDropdown, setShowDropdown] = useState(false);
  const [selectedUsers, setSelectedUsers] = useState(assignedUsers || []);
  const [isFocused, setIsFocused] = useState(false);

  useEffect(() => {
    setSelectedUsers(assignedUsers);
  }, [assignedUsers]);

  useEffect(() => {
    if (searchQuery.length > 0) {
      const fetchUsers = async () => {
        try {
```

```

    const response = await axiosInstance.get(`/users/search?search=${searchQuery}`);
    const filteredUsers = response.data.filter(user =>
      !selectedUsers.some(u => u._id === user._id)
    );
    setUserList(filteredUsers);
    setShowDropdown(true);
  } catch (error) {
    console.error('Error fetching users:', error);
  }
};
fetchUsers();
} else {
  setUserList([]);
  setShowDropdown(false);
}
}, [searchQuery, selectedUsers]);

const handleSearchChange = (e) => {
  setSearchQuery(e.target.value);
};

const handleUserSelect = (user) => {
  if (!selectedUsers.find(u => u._id === user._id)) {
    const updatedUsers = [...selectedUsers, user];
    setSelectedUsers(updatedUsers);
    onAssignUsers(updatedUsers);
  }
  setSearchQuery("");
  setShowDropdown(false);
};

const handleRemoveUser = (userId) => {
  const updatedUsers = selectedUsers.filter(user => user._id !== userId);
  setSelectedUsers(updatedUsers);
  onAssignUsers(updatedUsers);
};

return (
  <div className="relative">
    <input
      type="text"
      value={searchQuery}
      onChange={handleSearchChange}
      onFocus={() => setIsFocused(true)}
    />
  </div>
);

```

```

      onBlur={() => setIsFocused(false)}
      className={`form-input-box w-full rounded-md px-4 py-2 transition-colors duration-300
$ {isFocused || searchQuery ? 'border-purple-500' : 'border-gray-300'}`}
      placeholder="Search users..."
    />
    <FaMagnifyingGlass className="absolute inset-y-0 right-0 flex items-center px-3 text-gray-500" />
    {showDropdown && (
      <ul className="absolute z-10 mt-2 text-sm w-full bg-white border rounded-lg shadow-lg">
        {userList.length > 0 ? (
          userList.map(user => (
            <li
              key={user._id}
              onClick={() => handleUserSelect(user)}
              className="p-2 hover:bg-purple-100 cursor-pointer"
            >
              {user.fullName} ({user.email})
            </li>
          ))
        ) : (
          <li className="p-2 text-gray-500">No results found</li>
        )}
      </ul>
    )}
    {showSelectedUsers && selectedUsers.length > 0 && (
      <div className="mt-2 flex flex-wrap gap-2">
        {selectedUsers.map(user => (
          <div key={user._id} className="flex items-center bg-purple-100 border border-purple-300
text-purple-800 p-2 px-4 rounded-md shadow-md">
            <span className="text-xs truncate">{user.fullName}</span>
            <button
              type="button"
              onClick={() => handleRemoveUser(user._id)}
              className="ml-2 text-red-500 hover:text-red-700"
            >
              <IoMdClose size={13} />
            </button>
          </div>
        ))}
      </div>
    )}
  </div>
);
};

export default UserSearchBar;

```

Контролер *searchUsers* у серверској апликацији обрађује захтеве за претрагу корисника. Када корисник унесе критеријум претраге у *UserSearchBar* компоненту, овај контролер прима захтев са одговарајућим параметром и претражује базу података користећи *RegExp* како би пронашао кориснике чије име садржи задати текст. У случају да је претрага успешна, враћа листу корисника, док у супротном враћа одговарајућу поруку о грешци. Овај контролер је кључан за динамичку претрагу и селекцију корисника у систему.

Обрада података на серверу:

```
exports.searchUsers = async (req, res) => {
  try {
    const query = req.query.search;
    if (!query) {
      console.log("Search query is missing");
      return res.status(400).json({ message: "Search query is required" });
    }
    const users = await User.find({ fullName: new RegExp(query, 'i') });
    res.json(users);
  } catch (error) {
    console.error("Error searching users:", error);
    res.status(500).json({ message: error.message });
  }
};
```

Ова компонента значајно доприноси ефикасности система, омогућавајући корисницима да брзо пронађу и додају одговарајуће чланове тимова на пројекте.

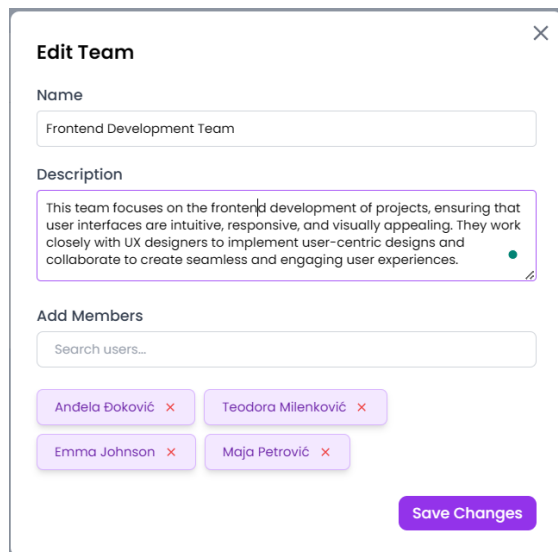
7.4.2. Брисање тимова

Брисање тимова се врши на исти начин као и брисање пројеката, кликом на акцију *Delete* поред тима који желите да уклоните. Само корисник који је креирао тим има дозволу да га обрише. Уколико корисник који није креирао тим покуша да изврши акцију брисања, систем ће приказати поруку о грешци и спречити брисање. Као и код брисања пројеката, након успешног брисања, табела тимова се аутоматски освежава како би одражавала промене.

7.4.3. Ажурирање тимова

Ажурирање тимова се врши на сличан начин као и ажурирање пројеката. Кликом на акцију *Edit* поред тима који желите да измените, отвара се форма са већ попуњеним подацима о одабраном тиму. У овој форми, приказаној на слици 12, корисник може прегледати и изменити све информације везане за тим, укључујући име тима, опис и додељене кориснике.

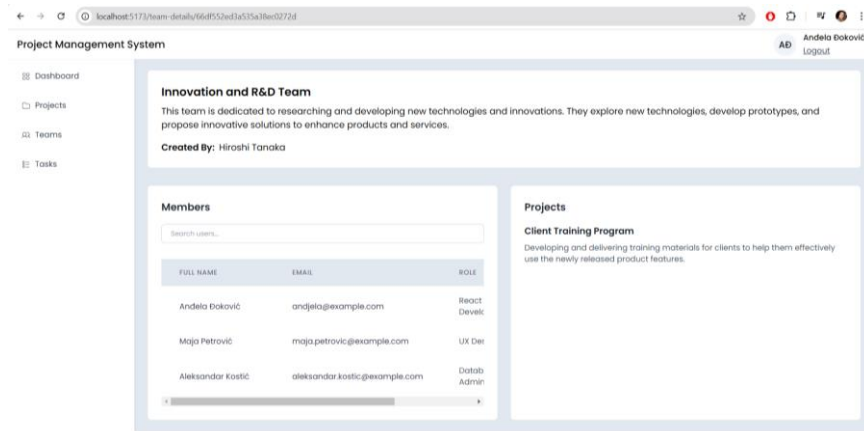
Након што су све измене унесене, клик на *Save Changes* покреће процес ажурирања података на серверу. Ажурирање у бази података осигурава да су све промене успешно примењене и да се тим ажурира са новим информацијама.

The image shows a modal window titled "Edit Team" with a close button (X) in the top right corner. Inside the modal, there are three main sections: 1. "Name" with a text input field containing "Frontend Development Team". 2. "Description" with a text area containing the text: "This team focuses on the frontend development of projects, ensuring that user interfaces are intuitive, responsive, and visually appealing. They work closely with UX designers to implement user-centric designs and collaborate to create seamless and engaging user experiences." 3. "Add Members" with a search input field labeled "Search users...". Below the search field, there are four member cards, each with a name and a red 'X' icon for removal: "Andela Đoković", "Teodora Milenković", "Emma Johnson", and "Maja Petrović". At the bottom right of the modal is a purple button labeled "Save Changes".

Слика 12. Ажурирање података о тиму

7.4.4. Детаљи о тиму

Кликом на тим отвара се страница са детаљним информацијама о тиму, приказана на слици 13. На овој страници корисници могу видети све чланове тима, њихове улоге и пројекте на којима тим ради. Пружа се преглед свих важних аспеката везаних за тим, укључујући чланове и њихове улоге у тиму, као и на којим пројектима је тим ангажован. Овај преглед омогућава боље разумевање задатака и обавеза тима, чиме се побољшава управљање и координација.

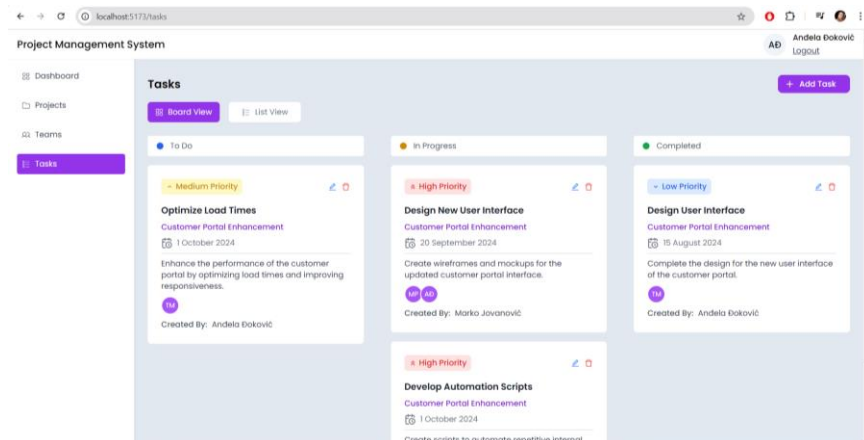


Слика 13. Приказ детаљнијих информација за тим

Осим прегледа, страница такође омогућава управљање тимом. Корисници могу додавати нове чланове тима или уклањати постојеће, чиме се обезбеђује флексибилност у организацији тима. Детаљнији подаци о члановима тима, као што су њихови контакт подаци и улоге, такође су доступни, што доприноси бољој организацији и сарадњи у оквиру тима.

7.5. Управљање задацима

На страници приказаној на слици 14, корисницима је омогућено да управљају задацима који припадају различитим пројектима. Ова страница пружа преглед свих задатака, било да су у току или завршени. Корисници могу лако додавати нове задатке, као и брисати или измењивати постојеће, што омогућава ефикасно управљање и праћење напретка у пројекту.



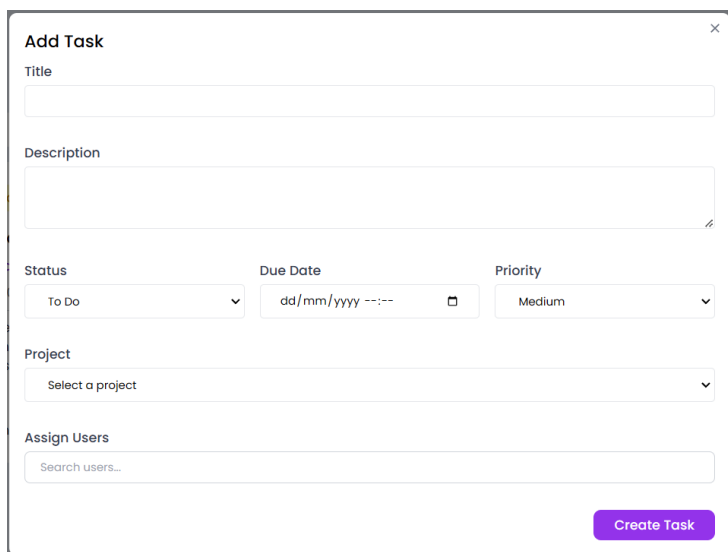
Слика 14. Приказ странице за управљање задацима

Корисници могу да креирају нове задатке уносећи важне информације као што су назив задатка, опис, статус, рок, приоритет и пројекат коме припада. Такође, могу управљати постојећим задацима, укључујући измену информација о задатку, као што су промене у опису, статусу или приоритету, као и уклањање задатака који више нису потребни. Ове функције омогућавају ефикасно праћење напретка задатака и осигуравају да сви аспекти пројекта буду адекватно управљани и завршени у предвиђеном року.

7.5.1. Креирање задатака

Креирање нових задатака у систему омогућава корисницима да унесу све потребне информације о задатку, укључујући назив, опис, статус, рок, приоритет, пројекат и додељене кориснике. Корисници могу приступити форми, приказаној на слици 15, за додавање новог задатка кликом на дугме *Add Task*, што отвара форму у којој могу унети све релевантне податке.

Форма за додавање задатка укључује поља за унос назива задатка, опис, статус (*To Do*, *In Progress*, *Completed*), датум и време рока, као и приоритет (високи, средњи или ниски). Корисници такође могу одабрати пројекат којем задатак припада и назначити кориснике којима ће задатак бити додељен. По завршетку уноса свих потребних података и подношењу форме, подаци се шаљу серверу и нови задатак се креира у бази података.



Слика 15. Креирање новог задатка

У овом контексту, важан део форме за додавање задатка је поље форме та претрагу пројеката. Ово поље омогућава корисницима да претражу пројекте којима су додељени. Уместо да користе општи списак свих пројеката у систему, корисници могу да прегледају и изаберу само оне пројекте којима су већ додељени, чиме се обезбеђује боља управљивост и фокусираност. Овакво ограничење у претрази помаже у избегавању конфузије и

омогућава корисницима да лакше пронађу релевантне пројекте за конкретне задатке. По завршетку уноса свих потребних података и подношењу форме, подаци се шаљу серверу и нови задатак се креира у бази података.

Поред креирања задатака, корисници могу прегледати постојеће задатке у два различита приказа: *Board View* и *Table View*. Ови прикази пружају флексибилност у начину прегледа и управљања задацима, омогућавајући корисницима да одаберу најбољи начин за визуализацију и управљање својим задацима у зависности од својих потреба.

7.5.2. Брисање задатака

Брисање задатака се врши на исти начин као и брисање тимова и пројеката. Кликом на акцију *Delete* поред задатка који желите да уклоните, задатак ће бити обрисан. Само корисник који је креирао задатак има дозволу да га обрише. Уколико корисник који није креирао задатак покуша да изврши акцију брисања, систем ће приказати поруку о грешци и спречити брисање. Након успешног брисања, табела задатака се аутоматски освежава како би одражавала промене.

7.5.3. Ажурирање задатака

Ажурирање задатака се врши на сличан начин као и ажурирање тимова и пројеката. Кликом на акцију *Edit* поред задатка који желите да измените, отвара се форма, приказана на слици 16, са већ попуњеним подацима о одабраном задатку. У овој форми, корисник може прегледати и изменити све информације везане за задатак, укључујући назив задатка, опис, статус, рок, приоритет, пројекат и додељене кориснике.

Након што су све измене унесене, клик на *Save Changes* покреће процес ажурирања података на серверу. Ажурирање у бази података осигурава да су све промене успешно примењене и да се задатак ажурира са новим информацијама.

Слика 16. Ажурирање података о задатку

7.5.4. Детаљи о задатку

На страници детаља о задатку, корисници могу добити свеобухватан увид у све информације везане за одређени задатак. Ова страница приказује све важне детаље о задатку, укључујући назив, опис, статус, рок, приоритет, пројекат и додељене кориснике.

Корисници такође имају могућност да додају коментаре на задатак. Коментари се могу унети у одговарајуће поље, а нови коментари се додају кликом на дугме *Add Comment*. Сви остављени коментари ће бити приказани са информацијама о кориснику који је коментар оставио, као и времену и датуму када је коментар додат. Ова функционалност омогућава корисницима да размењују информације и дискусије о напредовању задатка, чиме се побољшава сарадња и управљање задатком.

```
<div className="bg-white p-6 rounded-md mb-6">
  <h3 className="text-lg font-semibold text-gray-800 mb-4">Comments</h3>
  <div className="mb-4 flex items-start gap-2">
    <textarea
      className="form-input-box flex-1"
      rows="1"
      placeholder="Add a comment..."
      value={newComment}
      onChange={(e) => setNewComment(e.target.value)}
    ></textarea>
    <button
```

```

      className="btn-primary w-36 text-xs p-2"
      onClick={handleAddComment}
    >
      Add Comment
    </button>
  </div>
  <div className="space-y-6">
    {comments.length > 0 ? comments.map((comment) => (
      <div key={comment._id} className="p-4 bg-slate-200 border border-gray-200
rounded-md flex items-start space-x-4">
        <div className="flex-1">
          <div className="flex justify-between items-center mb-1">
            <p className="text-sm font-semibold text-gray-
800">{comment.user.fullName}</p>
            <span className="text-xs text-gray-
500">{formatDateTime(comment.timestamp)}</span>
          </div>
          <p className="text-sm text-gray-700">{comment.comment}</p>
        </div>
      </div>
    )) : (
      <p className="text-gray-500">No comments yet.</p>
    )}
  </div>
</div>

```

Контролер за додавање коментара управља процесом додавања нових коментара на задатке. Када корисник унесе коментар и пошаље захтев, контролер пронађе задатак по *taskId*. Ако задатак не постоји, враћа се статус 404 и порука „*Task not found*“. Ако задатак постоји, контролер креира нови коментар и додаје га у задатак, чувајући промене у бази података. Након тога, задатак се поново учита са ажурираним коментарима и корисничким информацијама. Контролер такође бележи активност додавања коментара, чиме се осигурава праћење важних догађаја у систему.

```

exports.addComment = async (req, res) => {
  const { taskId } = req.params;
  const { comment } = req.body;

  try {
    const task = await Task.findById(taskId);
    if (!task) return res.status(404).json({ message: 'Task not found' });

    const newComment = {
      user: req.user._id,
      comment: comment,
    };
  }

```

```
task.comments.push(newComment);
await task.save();

const populatedTask = await Task.findById(taskId).populate('comments.user', 'fullName');

await Activity.create({
  message: `A new comment has been added to task "${task.title}"`,
  type: 'task',
  action: 'comment_added',
  userId: req.user._id,
  taskId: task._id,
  projectId: task.project
});

res.status(201).json(populatedTask.comments[populatedTask.comments.length - 1]);
} catch (error) {
  console.error('Error adding comment:', error);
  res.status(500).json({ message: 'Server error', error });
}
};
```

8. ЗАКЉУЧАК

У овом раду представљен је развој и имплементација система за управљање пројектима уз коришћење савремених веб технологија. Главни циљ је био да се креира флексибилан и сигуран систем који омогућава ефикасно управљање пројектима, задацима и тимовима. Кроз коришћење MongoDB базе података, омогућено је складиштење података у JSON формату и брза манипулација над њима, што је било од суштинске важности за функционисање апликације.

Безбедност система је осигурана применом JWT аутентификације, што обезбеђује да су подаци корисника заштићени и да је приступ систему омогућен само овлашћеним корисницима. Кориснички интерфејс је дизајниран у React-у, са фокусом на једноставност и функционалност, што омогућава лаку навигацију кроз апликацију и ефикасно управљање задацима и тимовима.

Иако је систем постигао све иницијално постављене циљеве, постоји простор за даљи развој и унапређење. Потенцијална побољшања укључују проширење функционалности и прилагођавање система новим захтевима корисника. Ово оставља отворене могућности за будућа унапређења која би могла допринети још већој ефикасности и корисничком задовољству.

У целини, овај систем представља солидну основу за даљи развој и прилагођавање специфичним потребама корисника и организација. Уз константна унапређења и нове функционалности, ова апликација има потенцијал да постане кључни алат за управљање пројектима у различитим индустријама.

Commented [MB4]: Закључак нека почне на новој страни. Заправ свако ново поглавље увек треба да почиње на новој страни.

РЕФЕРЕНЦЕ

- [1] *Express - Node.js web application framework*. Преузето са Express.js: <https://expressjs.com/> (последњи приступ: 19. августа 2024)
- [2] *MongoDB Documentation*. Преузето са MongoDB: <https://www.mongodb.com/docs/> (последњи приступ: 19. августа 2024)
- [3] *Node.js Documentation u Asynchronous Programming*. Преузето са Node.js : <https://nodejs.org/docs/> (последњи приступ: 19. августа 2024)
- [4] *React - A JavaScript library for building user interfaces*. Преузето са React : <https://reactjs.org/docs/getting-started.html> (последњи приступ: 19. септембар 2024)
- [5] *Tailwind CSS - Documentation*. Преузето са Tailwind CSS: <https://tailwindcss.com/docs> (последњи приступ: 19. септембар 2024)
- [6] *Node.js middleware for handling multipart/form-data*. Преузето са Multer: <https://github.com/expressjs/multer> (последњи приступ: 8. септембар 2024)
- [7] *JSON Web Tokens*. Преузето са JWT.IO.: <https://jwt.io/> (последњи приступ: 8. септембар 2024)
- [8] *Promise based HTTP client for the browser and node.js*. Преузето са Axios: <https://axios-http.com/> (последњи приступ: 8. септембар 2024)

Commented [MB5]: Референце нека почну на новој страни. Испред обриши 9. Додај бар још 3-4 референце. Тамо где си написала линк додај датум последњег приступа.