

**ВИСОКА ТЕХНИЧКА ШКОЛА
СТРУКОВНИХ СТУДИЈА У НОВОМ САДУ
СТУДИЈСКИ ПРОГРАМ: ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ**

**Примена интернет технологија у
пројектовању full-stack веб апликације за
вођење евиденције малих радионица
ЗАВРШНИ РАД**

**Кандидат:
Владимир Хромиш ИТ103/17**

Нови Сад, 2024.

**ВИСОКА ТЕХНИЧКА ШКОЛА
СТРУКОВНИХ СТУДИЈА У НОВОМ САДУ
СТУДИЈСКИ ПРОГРАМ: ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ**

**Примена интернет технологија у
пројектовању full-stack веб апликације за
вођење евиденције малих радионица
ЗАВРШНИ РАД**

Кандидат:
Владимир Хромиш ИТ103/17

Ментор:
мр Нинослава Тихи, пред.

Нови Сад, 2024.

**ВИСОКА ТЕХНИЧКА ШКОЛА
СТРУКОВНИХ СТУДИЈА У НОВОМ САДУ
СТУДИЈСКИ ПРОГРАМ: ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ**

Назив теме завршног рада:

Примена интернет технологија у пројектовању full-stack веб апликације за вођење евиденције малих радионица

Текст задатка:

- Описати интернет технологије коришћене за израду full-stack веб апликације
- Креирати fron-tend веб апликације коришћењем React компоненти, TypeScript-а за типизацију и Ant Design-а за визуелне елементе
- Креирати backend апликације помоћу Express.js-а за комуникацију са frontend-ом и MySQL базом
- Креирати MySQL базу података и применити Axios.js-а за слање HTTP захтева ка backend-у и извршавање CRUD (Create, Read, Update, Delete) операција над подацима у MySQL бази
- Дискутовати добијене резултате

Комисија:

1. _____, председник
2. _____, ментор
3. _____, члан

Датум: _____
Оцена: _____

Нови Сад, 2024

САДРЖАЈ:

1.	УВОД.....	1
2.	ЦИЉ РАДА.....	2
3.	ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА	3
3.1	React.js	3
3.1.1	React.js	3
3.1.2	Кључни Концепти	3
3.1.3	Предности коришћења React-a.....	5
3.1.4	React у контексту овог пројекта.....	6
3.2	Node.js.....	7
3.2.1	Увод	7
3.2.2	Кључне Карактеристике	7
3.2.3	Предности коришћења Node.js-a.....	8
3.2.4	Node.js у контексту овог пројекта.....	8
3.3	Express.js.....	9
3.3.1	Увод	9
3.3.2	Кључне Карактеристике	9
3.3.3	Предности коришћења Express.js-a.....	9
3.3.4	Express.js у контексту овог пројекта.....	10
3.4	MySQL.....	10
3.4.1	Увод	10
3.4.2	Кључне Карактеристике	10
3.4.3	Предности коришћења MySQL-a.....	11
3.4.4	MySQL у контексту овог пројекта.....	12
3.5	Ant Design.....	12
3.5.1	Увод	12
3.5.2	Кључне Карактеристике	12
3.5.3	Предности коришћења Ant Design-a.....	13
3.5.4	Ant Design у контексту овог пројекта.....	14
3.6	Axios	14
3.6.1	Увод	14
3.6.2	Кључне карактеристике	14
3.6.3	Axios у контексту овог пројекта	14
3.7	TypeScript	14
3.7.1	Увод	14
3.7.2	Кључне карактеристике	15

3.7.3	Предности коришћења TypeScript-а	16
3.7.4	TypeScript у контексту овог пројекта	16
4.	ИЗРАДА АПЛИКАЦИЈЕ.....	17
4.1	Структура апликације	17
4.1.1	Преглед главног директоријума.....	17
4.1.2	Преглед “backend” директоријума.....	18
4.1.3	Преглед “frontend” директоријума:.....	19
4.2	Frontend развој.....	20
4.2.1	Почетна страница	20
4.2.1	Форма у три корака	23
4.2.2	Први корак унос или одабир контакта:	24
4.2.3	Други корак Унос детаља о послу:	25
4.2.4	Трећи корак Унос података о наплати (за завршене послове):.....	26
4.2.5	Унос новог производа	27
4.2.6	Табеле и Модални прозори.....	28
4.2.7	Интеракција са backend-ом користећи axios:	30
4.2.8	TypeScript за побољшање квалитета кода.....	33
4.3	Backend развој.....	34
4.3.1	Обрада HTTP захтева	34
4.3.2	Express.js API руте и њихове функционалности	34
4.3.3	MySQL база података.....	36
5.	ЗАКЉУЧАК.....	37
6.	ЛИТЕРАТУРА	38

1. УВОД

У данашњем дигиталном добу, мала предузећа и радионице суочавају се са све већом потребом за ефикасним алатима за управљање пословањем. Дигитализација је променила начин на који послујемо. Мала предузећа и радионице све више препознају предности дигиталних алата за повећање продуктивности и учинковитости. За разлику од традиционалних метода, дигитална решења омогућују бржи приступ подацима, бољу организацију пословних процеса и лакше доношење информираних одлука.

Овај завршни рад има за циљ да представи развој једне такве апликације, креиране специфично за потребе малих радионица. Циљ је био да се развије интуитивно и свеобухватно web решење које ће омогућити лакше праћење свих аспеката пословања, од евиденције о извршеним пословима и стању залиха, до управљања финансијама.

Избор баш ове теме за завршни рад био је мотивисан личним искуством и жељом да се реши проблем с којим се суочавају многа мала предузећа. Као блиски пријатељ с власником једне такве мале радионице, сведочио сам изазовима с којима се сусретао приликом вођења евиденције о пословању користећи традиционалне методе, попут Excel табела. Честе погрешке при уносу података, губитак информација и недостатак прегледности били су само неки од проблема с којима се сусретао. Управо због тога сам одлучио развити web апликацију која ће омогућити сигурније, учинковитије и прегледније управљање тим делом пословања.

2. ЦИЉ РАДА

Циљ овог рада је развити web апликацију која ће омогућити малим радионицама ефикасно управљање пословањем, смањити ризик од губитка података и олакшати доношење пословних одлука. Апликација ће се фокусирати на праћење послова, робе и финансија, те ће бити дизајнирана тако да буде интуитивна и једноставна за коришћење, чак и за оне који немају претходно искуство с рачунарским програмима.

Главни циљ овог рада је креирати web апликацију која ће заменити застареле методе вођења евиденције у малим радионицама, попут Excel таблица. Апликација ће пружити сигурну и централизовану платформу за чување и управљање подацима, те ће омогућити генерисање различитих извештаја који ће помоћи у доношењу информисаних пословних одлука.

Такође, циљ је да се прикаже како се апликација за чију изградњу су коришћене различите технологије React.js, Node.js, Express.js, MySQL интегрише у функционалну целину.

За реализацију овог пројекта, одлучио сам се за следећи технолошки stack:

React.js: Као модеран Javascript Framework за изградњу корисничког интерфејса, React.js је омогућио брзу и ефикасну развојну продуктивност. Његова компонента базирана архитектура олакшала је организацију кода и одржавање апликације.

Node.js: Као runtime окружење за JavaScript, је омогућио изградњу server-side дела апликације. Његова асинхрона архитектура је идеална за развој real-time апликација, каква је ова.

Express.js: Као минималистички web framework за Node.js, Express.js је послужио као основа за изградњу Backend дела. Његова флексибилност и велика заједница су га учиниле одличним избором за овај пројекат.

MySQL: Као релациона база података, MySQL је коришћен за чување података о пословима, роби и финансијама. Његова робусност и зрелост су га учиниле поузданом опцијом за ову врсту апликације.

У наставку рада биће детаљније описани сви аспекти развоја апликације, од дизајна базе података, преко имплементације пословне логике, до изградње корисничког интерфејса. Такође, биће приказани резултати тестирања и евалуације развијене апликације.

3. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА

У овом поглављу ћемо се дубље позабавити техничким аспектима изградње овог веб сајта. Објаснићемо како смо користили React.js за креирање интерфејса, Node.js као окружење за извршавање server-side кода, Express.js за креирање API и Ant Design-а за бржу и ефикаснију изградњу компоненти.

Опширније: Осим наведених технологија, у изради смо користили и Axios за лакше управљање HTTP захтевима те TypeScript за додатну типизацију и сигурност кода.

3.1 React.js

3.1.1 *React.js*

Често називан само React, је популарна JavaScript библиотека отвореног кода коју је развио Facebook за изградњу корисничких интерфејса. У овом поглављу ћемо детаљно истражити React, његове кључне концепте, предности и како се уклапа у ширу слику развоја веб апликација.

React је одабран због своје декларативне природе, што значи да описујемо како желимо да наш интерфејс изгледа, а React се брине о томе да га ажурира ефикасно када се подаци промене. Ово поједностављује развој комплексних интерфејса и чини код лакшим за разумевање и одржавање.

3.1.2 *Кључни Концепти*

3.1.2.1 Компоненте

У срцу React-а лежи концепт компоненти. Компоненте су самосталне, јединице кода које би требале бити за виšekратну употребу и оне представљају део корисничког интерфејса. Можемо их замислити као градивне блокове од којих се састоји наша апликација. Свака компонента дефинише како одређени део интерфејса треба да изгледа и како треба да се понаша.

Компоненте могу бити једноставне, попут дугмета или текстуалног поља, или сложене, попут форме за пријаву или целог приказа листе производа. Оно што је важно је да свака компонента има јасно дефинисану одговорност и да се може користити више пута унутар апликације.

3.1.2.2 JSX

JSX (JavaScript XML) је синтаксна екстензија за JavaScript која нам омогућава да пишемо HTML-сличан код унутар JavaScript-а. Иако на први поглед може изгледати необично, JSX нам заправо олакшава дефинисање структуре компоненти и чини код читљивијим.

Уместо да користимо компликоване JavaScript функције за креирање елемената, са JSX-ом можемо директно да пишемо HTML тагове унутар наших компоненти. Ово чини код интуитивнијим, посебно за оне који су већ упознати са HTML-ом.

3.1.2.3 Виртуални DOM

Један од кључних фактора који доприноси перформансама React апликација је концепт виртуалног DOM-а. Уместо да директно манипулише стварним DOM-ом (Document Object Model) претраживача, React користи виртуелну репрезентацију DOM-а.

Када се подаци у апликацији промене, React прво ажурира виртуални DOM, а затим пореди нову верзију виртуалног DOM-а са претходном. На основу те разлике, React израчунава најефикаснији начин да ажурира стварни DOM, мењајући само оне елементе који су се заиста променили. Ово смањује број директних манипулација стварним DOM-ом, што доводи до бржег и ефикаснијег ажурирања интерфејса.

3.1.2.4 HOOKS

HOOKS су релативно нова функционалност у React-у која је уведена у верзији 16.8. Они нам омогућавају да користимо стање и друге React функционалности унутар функционалних компоненти. Пре увођења HOOKS-а, стање је било могуће користити само унутар класа компоненти.

HOOKS су заправо функције које почињу са префиксом use (нпр. useState, useEffect, useContext). Сваки hook има специфичну намену и омогућава нам да извршавамо одређене акције у различитим фазама животног циклуса компоненте.

3.1.3 Предности коришћења React-a

3.1.3.1 Декларативност

Једна од највећих предности React-a је његова декларативна природа. Уместо да се фокусирамо на императивне кораке како да ажурирамо интерфејс, ми једноставно описујемо како желимо да интерфејс изгледа у датом тренутку. React се брине о томе да интерфејс буде синхронизован са подацима, чак и када се подаци промене.

Ово чини код читљивијим и лакшим за разумевање, посебно када радимо на комплексним интерфејсима. Такође, олакшава тестирање и отклањање грешака, јер можемо да се фокусирамо на проверу да ли интерфејс исправно приказује податке, а не на то како се подаци ажурирају.

3.1.3.2 Компонентна архитектура

React подстиче компонентну архитектуру, где се апликација дели на мање, самосталне компоненте. Ово има неколико предности:

Вишекратност кода: Компоненте се могу користити више пута унутар апликације, што смањује количину кода који треба да напишемо и олакшава одржавање.

Модуларност: Свака компонента има јасно дефинисану одговорност, што чини код организованијим и лакшим за разумевање.

Тестирање: Компоненте се могу тестирати независно једна од друге, што олакшава проналажење и исправљање грешака.

3.1.3.3 Перформансе

React користи виртуални DOM да оптимизује ажурирања интерфејса. Уместо да свака промена података доведе до потпуног поновног рендеровања целог интерфејса, React ажурира само оне делове који су се заиста променили. Ово значајно побољшава перформансе апликације, посебно када радимо са великим количинама података или комплексним интерфејсима.

3.1.3.4 Велика заједница и екосистем

React има велику и активну заједницу, што значи да постоји обиље ресурса, библиотека и алата који нам могу помоћи у развоју. Такође, постоји много online туторијала, курсева и блог постова који нам могу помоћи да научимо React и да решимо проблеме на које наиђемо.

3.1.3.5 Флексибилност

React је веома флексибилан и може се користити за изградњу различитих врста апликација:

Web апликације: React је првобитно дизајниран за изградњу web апликација, и то је и даље његова примарна намена.

Мобилне апликације: Помоћу React Native-а, можемо користити React за изградњу мобилних апликација за iOS и Андроид платформе.

Десктоп апликације: Помоћу Enozent-а, можемо користити React за изградњу десктоп апликација за Windows, MacOS и Linux оперативне системе.

3.1.4 React у контексту овог пројекта

У овом пројекту, React ће бити коришћен за изградњу frontend-а, односно дела апликације који корисник види и са којим интерагује. React ће нам омогућити да креирамо динамичан и респонзиван кориснички интерфејс, који ће ефикасно приказивати податке из базе података и омогућити кориснику да извршава различите акције.

Комбинацијом React-а са другим технологијама као што су Node.js, Express.js и MySQL, моћи ћемо да изградимо комплетну web апликацију која ће имати и frontend и backend компоненту.

У наредним поглављима ћемо детаљније истражити како се React користи у пракси, кроз конкретне примере и имплементацију функционалности апликације.

3.2 Node.js

3.2.1 Увод

Node.js је open-source JavaScript runtime који омогућава извршавање JavaScript кода на серверској страни. Ово поглавље ће се детаљно бавити Node.js-ом, његовим карактеристикама, предностима и начином на који се уклапа у развој web апликација, посебно у контексту backend развоја.

Node.js је изабран због своје способности да ефикасно обрађује велики број истовремених захтева, што га чини идеалним за изградњу скалабилних и перформантних backend система. Такође, коришћење JavaScript-а и на frontend-у и на backend-у има за предност да је по природи компатибилнији од алтернатива

3.2.2 Кључне Карактеристике

3.2.2.1 Non-blocking I/O model

Node.js користи non-blocking I/O модел, што значи да може да обрађује више захтева истовремено без блокирања. Када Node.js наиђе на операцију која захтева чекање (нпр. читање из фајла или упит бази података), он неће блокирати цео процес. Уместо тога, он ће наставити да обрађује друге захтеве, а када се операција заврши, Node.js ће се вратити на њу и наставити са извршавањем.

Овај модел омогућава Node.js-у да ефикасно обрађује велики број конкурентних захтева, чак и на серверима са ограниченим ресурсима.

3.2.2.2 Event-driven архитектура

Node.js је заснован на event-driven архитектури. То значи да се извршавање кода покреће догађајима, као што су завршетак I/O операције, пријем HTTP захтева или истицање тајмера.

Када се догоди неки догађај, Node.js ће позвати одговарајућу callback функцију која ће обрадити тај догађај. Овај модел омогућава Node.js-у да буде веома ефикасан у обради великог броја догађаја, јер не мора да троши ресурсе на непотребно чекање.

3.2.2.3 NPM (Node Package Manager)

NPM је систем за управљање пакетима за Node.js. Он омогућава једноставну инсталацију, ажурирање и уклањање Node.js модула (пакета). NPM садржи огроман број модула који се могу користити за различите намене, од web framework-а и база података до алата за тестирање и deployment.

Коришћење NPM-а значајно олакшава развој Node.js апликација, јер нам омогућава да брзо и једноставно додајемо нове функционалности у апликацију.

3.2.3 Предности коришћења Node.js-a

3.2.3.1 Високе перформансе и скалабилност

Захваљујући non-blocking I/O моделу и event-driven архитектури, Node.js је веома ефикасан у обради великог броја конкурентних захтева. Ово га чини идеалним за изградњу скалабилних backend система који могу да поднесу велики број корисника.

JavaScript на обе стране

Коришћење JavaScript-a и на frontend-у и на backend-у има неколико предности:

Лакша размена кода и знања: Исти језик се користи на обе стране, што олакшава комуникацију и сарадњу између frontend и backend тимова.

- **Бржи развој:** Програмери могу да користе своје JavaScript вештине и на frontend-у и на backend-у, што убрзава процес развоја.
- **Мање контекстуалних промена:** Програмери не морају да прелазе између различитих језика и окружења, што им омогућава да остану фокусирани на решавање проблема.

3.2.3.2 Велика заједница и екосистем

Node.js има велику и активну заједницу, што значи да постоји обиље ресурса, библиотека и алата који нам могу помоћи у развоју. NPM садржи огроман број модула који се могу користити за различите намене, што нам омогућава да брзо и једноставно додајемо нове функционалности нашој апликацији.

3.2.3.3 Микро сервисна архитектура

Node.js је погодан за изградњу микро сервисних архитектура, где се апликација дели на мање, независне сервисе. Ово има неколико предности:

- **Скалабилност:** Једна од предности микро сервисне архитектуре је могућност да се сваки појединачни сервис прилагођава променама у оптерећењу независно од осталих сервиса. Ово значи да уколико један део апликације доживи повећану употребу, можемо повећати ресурсе само за тај део, без потребе да мењамо цео систем.
- **Флексибилност:** Сервиси се могу развијати и ажурирати независно један од другог, што убрзава процес развоја и омогућава лакше експериментисање са новим технологијама.
- **Отпорност на грешке:** Ако један сервис закаже, то неће оборити целу апликацију.

3.2.4 Node.js у контексту овог пројекта

У овом пројекту, Node.js ће бити коришћен за изградњу backend-a, односно дела апликације који се бави обрадом података, комуникацијом са базом података и пружањем API endpoint-a за frontend. Node.js ће нам омогућити да изградимо ефикасан и скалабилан backend који ће моћи да подржи велики број корисника и захтева.

Комбинацијом Node.js-a са Express.js framework-ом и MySQL базом података, моћи ћемо да изградимо комплетан backend систем који ће омогућити frontend-у да приступа подацима и извршава различите акције.

У наредним поглављима ћемо детаљније истражити како се Node.js користи у пракси, кроз конкретне примере и имплементацију backend функционалности апликације.

3.3 Express.js

3.3.1 Увод

Express.js, или само Express, је минималистички и флексибилан web framework за Node.js који олакшава изградњу web апликација и API-ја. У овом поглављу детаљније ћемо истражити Express.js, његове кључне карактеристике и предности, као и његову улогу у изградњи backend-а наше апликације.

Express је постао популаран избор за Node.js програмере због своје једноставности, лакоће коришћења и способности да се брзо изграде робусне web апликације. Он пружа основну структуру и алате који су неопходни за рутирање захтева, управљање middleware-ом и интеракцију са базама података.

3.3.2 Кључне Карактеристике

3.3.2.1 Рутирање

Рутирање је процес усмеравања HTTP захтева на одговарајуће функције за обраду. Express пружа једноставан и интуитиван начин за дефинисање рута у апликацији. Можемо дефинисати руте за различите HTTP методе (GET, POST, PUT, DELETE, итд.) и различите URL путање.

3.3.2.2 Middleware

Middleware су функције које се извршавају током обраде HTTP захтева. Оне могу да приступе објекту захтева (request), објекту одговора (response) и следећем middleware-у у циклусу захтева-одговора. Middleware се користе за различите задатке, као што су:

- Парсирање тела захтева (body parsing)
- Аутентификација и ауторизација
- Руковање грешкама (error handling)
- Бележење (logging)
- И још много тога по потреби

Express омогућава лако додавање и уклањање middleware-а, што нам даје велику флексибилност у прилагођавању обраде захтева према нашим потребама.

3.3.2.3 Template Engines

Template engine-и омогућавају генерисање динамичког HTML садржаја на серверу. Express подржава различите template engine-е, као што су Pug, EJS и Handlebars. Ово нам омогућава да одвојимо логику апликације од презентације, што чини код читљивијим и лакшим за одржавање.

3.3.3 Предности коришћења Express.js-а

3.3.3.1 Једноставност и лакоћа коришћења

Express је дизајниран да буде минималистички и лаган. Он пружа само основне алате који су неопходни за изградњу web апликација, што га чини лаким за учење и коришћење. Нема непотребне

комплексности или магије, што нам омогућава да се фокусирамо на решавање проблема и изградњу функционалности.

3.3.3.2 Флексибилност

Express је веома флексибилан и може се прилагодити различитим потребама и стиливима развоја. Можемо да бирамо које middleware-е желимо да користимо, који template engine желимо да користимо, па чак и како желимо да структурирамо нашу апликацију.

3.3.3.3 Велика заједница и екосистем

Express има велику и активну заједницу, што значи да постоји обиље ресурса, туторијала и модула који нам могу помоћи у развоју. Такође, постоји много компанија које користе Express у продукцији, што значи да је то проверен и поуздан framework.

3.3.4 *Express.js у контексту овог пројекта*

У овом пројекту, Express ће бити кључна компонента backend-а. Он ће нам омогућити да дефинишемо API endpoint-е које ће frontend користити за комуникацију са backend-ом. Такође, користећемо Express middleware за обраду захтева, аутентификацију, руковање грешкама и друге задатке.

Express ће нам омогућити да брзо и ефикасно изградимо backend који ће бити скалабилан и лако одржив.

У наредним поглављима ћемо се бавити конкретном имплементацијом Express.js-а у нашем пројекту, укључујући дефинисање рута, коришћење middleware-а и повезивање са базом података.

3.4 MySQL

3.4.1 Увод

MySQL је један од најпопуларнијих система за управљање релационим базама података отвореног кода (енгл. open-source relational database management system). У овом поглављу, детаљније ћемо се упознати са MySQL-ом, његовим кључним карактеристикама, предностима и начином на који ће бити интегрисан у наш пројекат како би обезбедио поуздано складиштење и управљање подацима.

MySQL је изабран због своје поузданости, перформанси, широке подршке и лакоће коришћења. Његова способност да ефикасно складишти и управља великим количинама структурираних података чини га идеалним избором за различите врсте web апликација.

3.4.2 Кључне Карактеристике

3.4.2.1 Релациони модел података

MySQL се заснива на релационом моделу података, што значи да се подаци организују у табеле које су повезане међусобно преко релација (веза). Овај модел омогућава ефикасно складиштење и управљање структурираним подацима, као и лако извршавање упита за претраживање, филтрирање и агрегацију података.

3.4.2.2 SQL (Structured Query Language)

MySQL користи SQL за интеракцију са базом података. SQL је стандардизован језик за управљање релационим базама података који омогућава креирање, модификовање и брисање табела, као и извршавање сложених упита над подацима.

3.4.2.3 ACID усаглашеност

MySQL је ACID усаглашен, што значи да гарантује atomicity, consistency, isolation и durability трансакција. Ово осигурава интегритет података чак и у случају грешака или отказа система.

3.4.2.4 Скалабилност

MySQL се може скалирати хоризонтално и вертикално како би подржао растуће потребе апликације. Хоризонтално скалирање подразумева додавање нових сервера бази података, док вертикално скалирање подразумева надоградњу хардвера постојећег сервера.

3.4.2.5 Безбедност

MySQL пружа различите механизме за осигуравање безбедности података, као што су контрола приступа, енкрипција и репликација података.

3.4.3 Предности коришћења MySQL-а

3.4.3.1 Поузданост и стабилност

MySQL је проверен и поуздан систем за управљање базама података који се користи у многим великим и критичним апликацијама. Његова ACID усаглашеност осигурава интегритет података чак и у случају грешака или отказа система.

3.4.3.2 Високе перформансе

MySQL је оптимизован за брзо извршавање упита и ефикасно коришћење ресурса система, што га чини погодним за апликације које захтевају високе перформансе.

3.4.3.3 Лакоћа коришћења

MySQL је релативно лак за учење и коришћење, посебно за оне који су већ упознати са SQL-ом. Постоји велики број алата и ресурса који олакшавају рад са MySQL-ом, као што су графички интерфејси, административни алати и online туторијали.

3.4.3.4 Велика заједница и екосистем

MySQL има велику и активну заједницу, што значи да постоји обиље подршке, документације и алата који нам могу помоћи у развоју.

3.4.3.5 Цена

MySQL је систем отвореног кода (open-source), што значи да је бесплатан за коришћење. Ово га чини атрактивним избором за пројекте са ограниченим буџетом.

3.4.4 MySQL у контексту овог пројекта

У овом пројекту, MySQL ће бити коришћен за складиштење и управљање подацима апликације. Користиће се за креирање табела, дефинисање релација између њих и извршавање упита за претраживање, филтрирање и ажурирање података.

Node.js backend ће комуницирати са MySQL базом података користећи одговарајући драјвер или ORM (Object-Relational Mapper) како би олакшао интеракцију са базом.

MySQL ће нам омогућити да изградимо робустан и скалабилан систем за складиштење података који ће моћи да подржи потребе наше апликације. У наредним поглављима ћемо се бавити конкретном имплементацијом MySQL-а у нашем пројекту, укључујући дизајн базе података, креирање табела и писање упита.

3.5 Ant Design

3.5.1 Увод

Ant Design је популарна UI библиотека компоненти отвореног кода за React апликације. У овом поглављу ћемо се детаљније упознати са Ant Design-ом, његовим кључним карактеристикама и предностима, као и његовом улогом у изградњи корисничког интерфејса наше апликације.

Ant Design је изабран због своје богате колекције висококвалитетних, прилагодљивих и приступачних компоненти које прате савремене дизајн трендове. Његова употреба ће нам омогућити да брзо и ефикасно изградимо атрактиван и функционалан кориснички интерфејс, без потребе да креирамо сваку компоненту од нуле.

3.5.2 Кључне Карактеристике

3.5.2.1 Богата колекција компоненти

Ant Design нуди широк спектар компоненти, укључујући:

- Основне компоненте: дугмад, иконице, мреже (grids), форме, итд.
- Навигационе компоненте: менији, breadcrumbs, pagination, итд.
- Компоненте за приказ података: табеле, листе, календари, итд.
- Повратне информације и интеракције: модални прозори, обавештења, tooltips, итд.
- И још много тога

Ова богата колекција компоненти покрива већину уобичајених потреба при изградњи корисничког интерфејса, што нам штеди време и труд.

3.5.2.2 Прилагодљивост (Customizability)

Ant Design компоненте су високо прилагодљиве. Можемо мењати њихов изглед и понашање путем различитих `propertija` (`props`) које им прослеђујемо. Такође, можемо користити Ant Design-ов систем за тематизирање како бисмо глобално прилагодили изглед свих компоненти у апликацији.

3.5.2.3 Приступачност (Accessibility)

Ant Design посвећује велику пажњу приступачности, што значи да су компоненте дизајниране тако да буду доступне и корисницима са инвалидитетом. Ово је важан аспект сваке модерне web апликације и помаже нам да изградимо инклузиван производ.

3.5.2.4 Интернационализација (I18n)

Ant Design подржава интернационализацију, што нам омогућава да прилагодимо нашу апликацију различитим језицима и културама.

3.5.2.5 Подршка за TypeScript

Ant Design пружа одличну подршку за TypeScript, што нам омогућава да користимо статичку типизацију и боље алате за развој.

3.5.3 Предности коришћења Ant Design-a

3.5.3.1 Бржи развој

Коришћењем готових компоненти из Ant Design-a можемо значајно убрзати процес развоја корисничког интерфејса. Не морамо да трошимо време на креирање сваке компоненте од нуле, већ се можемо фокусирати на имплементацију логике и функционалности апликације.

3.5.3.2 Конзистентан дизајн

Ant Design компоненте прате конзистентан дизајн систем, што обезбеђује да наш кориснички интерфејс изгледа професионално и уједињено. Ово побољшава корисничко искуство и олакшава навигацију кроз апликацију.

3.5.3.3 Приступачност

Ant Design-ове компоненте су дизајниране са приступачношћу на уму, што нам помаже да изградимо апликацију која је доступна свима, без обзира на њихове способности.

3.5.3.4 Активна заједница и подршка

Ant Design има велику и активну заједницу, што значи да постоји обиље ресурса, документације и подршке. Такође, Ant Design тим редовно објављује ажурирања и нове функционалности.

3.5.4 Ant Design у контексту овог пројекта

У овом пројекту, Ant Design ће бити коришћен за изградњу визуелних елемената корисничког интерфејса. Користићемо његове компоненте за креирање форми, табела, менија, модалних прозора и других елемената.

Ant Design ће нам омогућити да брзо и ефикасно изградимо атрактиван, функционалан и приступачан кориснички интерфејс који ће пружити одлично корисничко искуство.

У наредним поглављима ћемо видети конкретне примере коришћења Ant Design компоненти у нашој апликацији и како их можемо прилагодити нашим потребама.

3.6 Axios

3.6.1 Увод

Axios је популарна JavaScript библиотека која се користи за слање HTTP захтева са клијентске стране (обично из web прегледача) ка серверу. У контексту нашег пројекта, Axios ће бити кључна компонента за комуникацију између React frontend-а и Node.js/Express.js backend-а.

3.6.2 Кључне карактеристике

- **Једноставност:** Axios пружа једноставан и интуитиван API за слање различитих врста HTTP захтева (GET, POST, PUT, DELETE).
- **Поддршка за Promise:** Axios користи Promise објекте, што омогућава чистији и лакши за читање код приликом обраде асинхронних захтева.
- **Аутоматска трансформација података:** Axios аутоматски конвертује JSON податке у JavaScript објекте и обрнуто, што олакшава рад са подацима.
- **Заштита од XSS напада:** Axios има уграђене механизме за заштиту од XSS (Cross-Site Scripting) напада.
- **Прекид захтева:** Axios омогућава прекид HTTP захтева уколико је то потребно.

3.6.3 Axios у контексту овог пројекта

У овом пројекту, Axios ће бити коришћен на React frontend-у како би се слали HTTP захтеви ка Node.js/Express.js backend-у. Ови захтеви ће омогућити frontend-у да приступи подацима из MySQL базе података, ажурира податке, креира нове уносе и извршава друге акције које захтевају интеракцију са сервером.

Axios ће нам омогућити да ефикасно и једноставно организујемо комуникацију између frontend-а и backend-а, чиме ћемо осигурати да наша апликација буде респонзивна и динамична.

3.7 TypeScript

3.7.1 Увод

TypeScript је open-source програмски језик који је развио Microsoft, а представља надскуп (superset) JavaScript-а. Другим речима, сваки валидан JavaScript код је уједно и валидан TypeScript код.

TypeScript додаје статичку типизацију (static typing) JavaScript-у, што значи да можемо дефинисати типове података за променљиве, функције, параметре и повратне вредности.

У овом поглављу ћемо се детаљније упознати са TypeScript-ом, његовим кључним карактеристикама, предностима и његовом улогом у развоју наше апликације. TypeScript је изабран због своје способности да побољша квалитет кода, олакша откривање грешака и побољша алате за развој, што ће нам помоћи да изградимо робуснију и лакше одрживу апликацију.

3.7.2 Кључне карактеристике

3.7.2.1 Статичка типизација

Најважнија карактеристика TypeScript-а је статичка типизација. То значи да можемо дефинисати типове података за променљиве, функције, параметре и повратне вредности. Ово нам омогућава да:

- **Откријемо грешке у коду раније:** TypeScript компајлер ће проверити типове података у нашем коду и пријавити грешке ако постоји неусклађеност типова. Ово нам помаже да откријемо потенцијалне проблеме пре него што покренемо апликацију.
- **Побољшамо читљивост кода:** Типови података чине код јаснијим и лакшим за разумевање, посебно када радимо на великим пројектима или у тиму.
- **Искористимо боље алате за развој:** TypeScript омогућава интегрисаним развојним окружењима (IDE) да пружи боље функционалности као што су аутоматско допуњавање кода (code completion), проверу типова у реалном времену и рефакторинг.

3.7.2.2 Компатибилност са JavaScript-ом

TypeScript је дизајниран да буде компатибилан са JavaScript-ом. То значи да можемо постепено уводити TypeScript у постојеће JavaScript пројекте, без потребе да преписујемо цео код одједном. Такође, можемо користити све постојеће JavaScript библиотеке и framework-ове у TypeScript пројекту.

3.7.2.3 Објектно-оријентисано програмирање (ООП)

TypeScript подржава кључне концепте објектно-оријентисаног програмирања, као што су класе, интерфејси, наслеђивање и полиморфизам. Ово нам омогућава да пишемо структуриранији и модуларнији код, што је посебно корисно при раду на већим пројектима.

3.7.2.4 Декоратори

TypeScript подржава декораторе, који су специјалне функције које се могу користити за модификовање класа, метода, проперија и параметара. Декоратори се често користе у framework-овима као што је Angular за додавање додатних функционалности или мета података.

3.7.2.5 Генерички типови (Generics)

TypeScript подржава генеричке типове, који нам омогућавају да пишемо флексибилнији и поновно употребљиви код. Генерички типови нам омогућавају да дефинишемо функције и класе које могу да раде са различитим типовима података, без потребе да пишемо посебну имплементацију за сваки тип.

3.7.3 Предности коришћења TypeScript-a

3.7.3.1 Побољшање квалитета кода

Статичка типизација у TypeScript-у помаже нам да откријемо грешке у коду раније, што доводи до смањења броја грешака у продукцији и побољшања укупног квалитета кода.

3.7.3.2 Повећање продуктивности

Бољи алати за развој које пружа TypeScript, као што су аутоматско допуњавање кода и провера типова у реалном времену, помажу нам да пишемо код брже и ефикасније.

3.7.3.3 Олакшано одржавање кода

Читљивији и структуриранији код који омогућава TypeScript олакшава разумевање и модификацију кода, посебно када радимо на великим пројектима или у тиму.

3.7.3.4 Постепена адаптација

TypeScript се може постепено уводити у постојеће JavaScript пројекте, што смањује ризик и омогућава нам да искористимо предности TypeScript-a без потребе да преписујемо цео код одједном.

3.7.4 TypeScript у контексту овог пројекта

У овом пројекту, TypeScript ће бити коришћен и на frontend-у (React.js) и на backend-у (Node.js/Express.js). Ово ће нам омогућити да:

- **Откријемо грешке раније:** TypeScript ће нам помоћи да откријемо потенцијалне проблеме са типовима података пре него што покренемо апликацију, што ће смањити број грешака у продукцији.
- **Побољшамо читљивост кода:** Типови података ће учинити наш код јаснијим и лакшим за разумевање, што ће олакшати сарадњу у тиму и будуће одржавање апликације.
- **Искористимо боље алате за развој:** TypeScript ће омогућити нашим IDE-има да пруже боље функционалности као што су аутоматско допуњавање кода и рефакторисање, што ће убрзати процес развоја.

TypeScript помаже да се изгради робуснија, лакше одржива и скалабилнија апликација, што је посебно важно када се ради на комплекснијим пројектима.

4. ИЗРАДА АПЛИКАЦИЈЕ

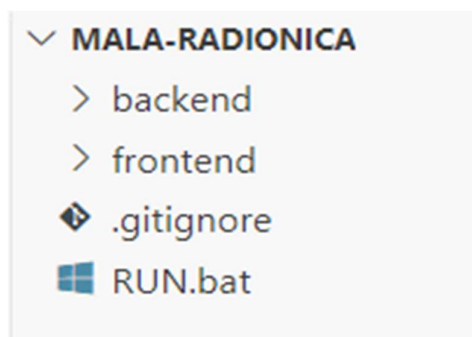
У овом поглављу детаљно ћемо описати израду full-stack web апликације за праћење послова, робе и финансија мале радионице. Циљ апликације је да омогући ефикасно управљање задацима, производима и плаћањима у радионици кроз интерактивни кориснички интерфејс и повезаност са базом података. Користили смо React.js за фронтенд, Node.js и Express.js за Backend, и MySQL за базу података. Систем је дизајниран тако да кориснику пружи лакоћу у уносу и прегледу података.

4.1 Структура апликације

4.1.1 Преглед главног директоријума

Апликација "Mala Radionica" је организована у два главна дела: frontend и backend, који су смештени у одговарајуће фасцикле унутар пројекта.

RUN.bat скрипта за брзо покретање зависности апликације у Windows окружењу



Слика 1. Главни директоријум

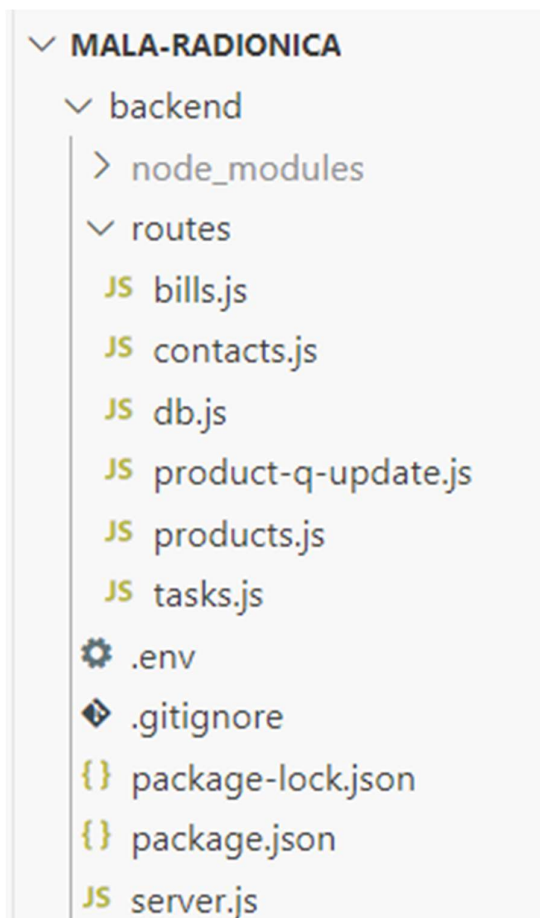
```

RUN.bat
1  @echo off
2  title Starting Development Servers...
3  cd frontend
4  start cmd.exe /k "npm run dev"
5  cd ../backend
6  start cmd.exe /k "npm start"
7  timeout /t 5 /nobreak > nul
8  start http://localhost:5173
9  echo Servers started.
10 pause
```

Слика 2. Run.bat скрипта

4.1.2 Преглед “backend” директоријума

Директоријум **backend** садржи кључне датотеке и руте које омогућавају интеракцију са MySQL базом података и управљање подацима:



Слика 3. Backend директоријум

node_modules под директоријум садржи све потребне зависности и њиме управља npm (node-package manager)

Руте су организоване по функционалностима:

bills.js: управљање рачунима

contacts.js: управљање контактима

db.js омогућава повезивање са MySQL базом података. Свака датотека у фасцикли **routes** дефинише специфичне API ендпоинте за креирање, читање, ажурирање и брисање података (CRUD операције).

products-q-update.js: управљање којичином производа

products.js: управљање производима

tasks.js: управљање пословима

db.js: омогућава повезивање са MySQL базом података.

Поред тога, ту су и датотеке:

.gitignore: садржи листу датотека и директоријума који се игноришу од стране git контроле верзије кода

.env: садржи поверљиве информације попут стрингова за повезивање са базом података.

server.js: скрипта која покреће сервер коришћењем Express.js.

Оваква организација директоријума и фајлова помаже да се омогући, раздвајање одговорности и лакше праћење кода.

4.1.3 Преглед “frontend” директоријума:

У корену директоријума се налазе кључне датотеке које претстављају компоненте које се користе широм frontend дела апликације:

index.html представља почетну страну апликације.

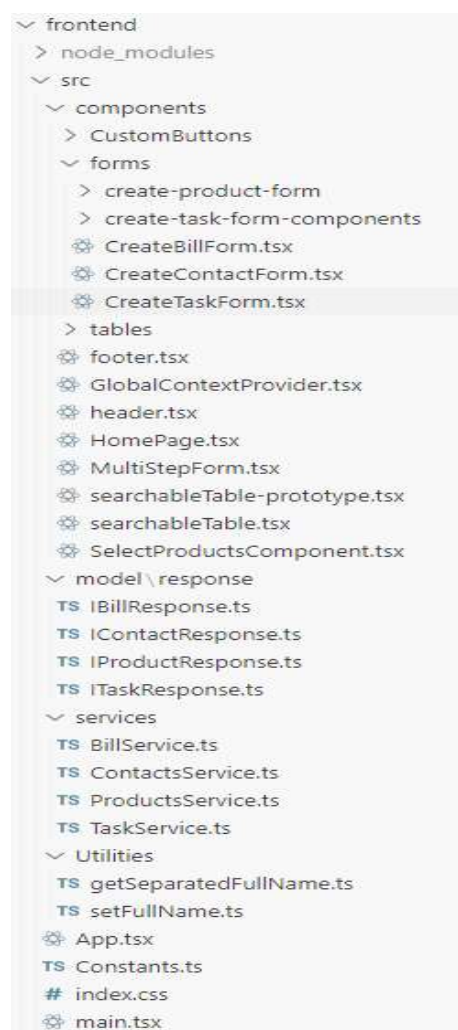
App.tsx: Главна компонента апликације која обједињује све остале компоненте и представља корен апликације.

main.tsx: Улазна тачка апликације која рендерује главну компоненту (App.tsx) у DOM.

index.css: Глобални CSS фајл који дефинише стилове који се примењују на целу апликацију.

GlobalContextProvider.tsx: Омогућава коришћење React Context API-ја за управљање глобалним стањем апликације, чинећи га доступним свим компонентама.

Унутар frontend директоријум се налазе следећи поддиректоријуми:



components:

Главне компоненте као што су **header.tsx**, **footer.tsx**, и **HomePage.tsx**, **MultiStepForm.tsx**.

tables:

Компоненте за приказ података у табелама, подељене по функционалностима (нпр. **bills-table-components**, **contacts-table-components**).

forms:

Компоненте које се користе за унос података у апликацију, укључујући форме за креирање производа (**create-product-form**), рачуна (**CreateBillForm.tsx**), и контаката (**CreateContactForm.tsx**), као и послова (**CreateTaskForm.tsx**).

services:

Фајлови који дефинишу комуникацију са backend-ом користећи Axios-а, попут **BillService.ts**, **ContactsService.ts**, и **ProductsService.ts**.

model/response:

Садржи моделе одређених типова података (нпр. **IBillResponse.ts**, **IProductResponse.ts**), што олакшава типизацију и употребу у TypeScript окружењу.

Слика 4. Frontend директоријум

Оваква организација директоријума и фајлова помаже да се омогући, раздвајање одговорности и лакше праћење кода.

4.2 Frontend развој

React.js је кључна технологија која омогућава изградњу динамичких страница са различитим компонентама. Његова декларативна природа и могућност поновног коришћења компоненти значајно су убрзали развој и олакшали одржавање корисничког интерфејса. React.js је у сржи frontend дела пројекта "Мала Радионица", пружајући структуру и функционалност за кориснички интерфејс.

Ant Design библиотека је кључна за визуелни идентитет и функционалност frontend-а овог пројекта. Пошто су блиско повезани објаснићемо их заједно.

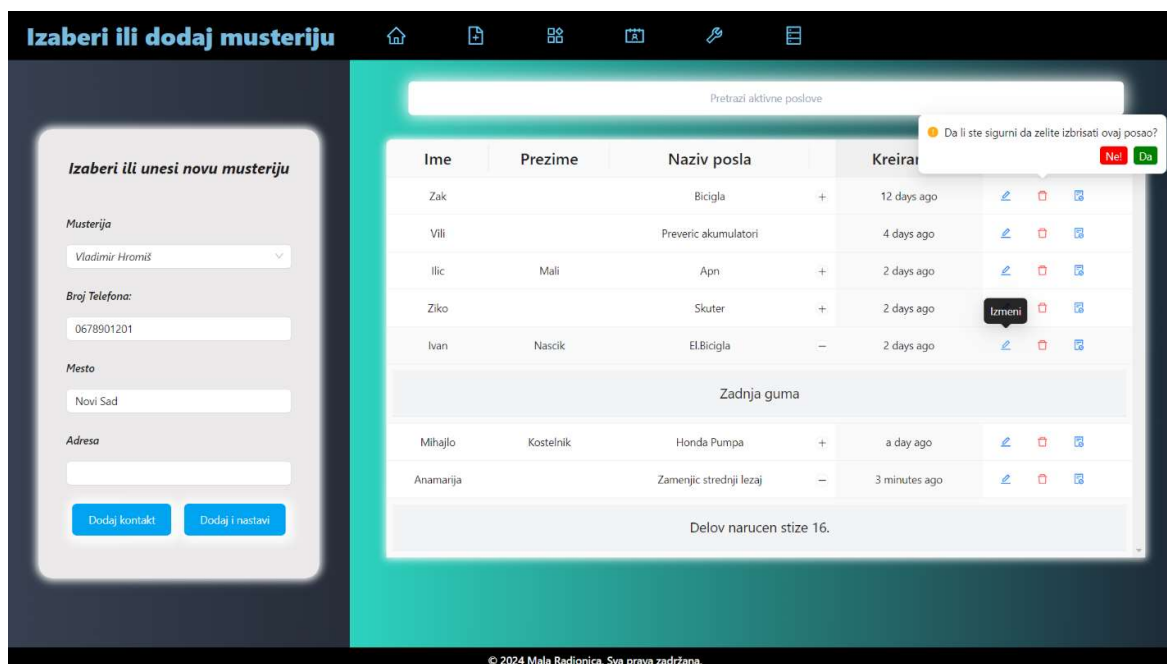
4.2.1 Почетна страница

Приказује табеларни преглед свих активних послова, укључујући релевантне информације о сваком послу: име и презиме муштерије, назив посла, датум креирања и акционе дугмиће за измену, брисање посла и његову наплату.

Изнад табеле је поље за претрагу које омогућава кориснику да брзо пронађе жељени посао филтрирањем приказа.

Као што можемо видети слици почетне странице апликације, кориснику су представљене три кључне функционалности о којима ћемо речи нешто више у следећим поглављима.

- **Навигациона трака**
- **Форма у три корака**
- **Листа приказа свих, активних послова уз могућност филтрирања помоћу претраге**



Слика 5. Почетна страница

React код који омогућава приказ почетне странице можемо видети на следећој слици:

```
import MultiStepForm from './MultiStepForm'
import TasksList from './tables/task-table-components/TaskList'

const HomePage = () => {
  return (
    <main className="flex xl:flex-row flex-col ">
      <section className="w-2/6 xl:p-10 lg:p-5 overflow-y-auto ">
        <MultiStepForm />
      </section>

      <section className="w-4/6 ">
        <TasksList />
      </section>
    </main>
  )
}

export default HomePage
```

Слика 6. React код почетне странице

Кључни елементи:

1. Импортује две компоненте: **MultiStepForm** и **TasksList**.
2. Враћа JSX структуру:

main: Главни контејнер странице, користи Tailwind CSS класе за флексибилни распоред (flex, xl:flex-row, flex-col).

Два section елемента:

Први заузима 2/6 ширине екрана на великим екранима (w-2/6), има padding и омогућава вертикално скривање (overflow-y-auto). У њему се приказује **MultiStepForm** компонента, за унос података у више корака.

Други заузима 4/6 ширине екрана (w-4/6) и приказује **TasksList** компоненту, за листинг задатака.

Укратко:

HomePage компонента креира основни распоред почетне странице, са формом за унос података са леве стране и листом задатака са десне стране, прилагођавајући се различитим величинама екрана.

4.2.1.1 React код који дефинише форму у 3 корака :

Кључне карактеристике:

1. Управљање стањем:

Користи **useGlobalContext** hook за приступ дељеним подацима и функцијама из контекста апликације. Ово омогућава компоненти да комуницира са другим деловима апликације и да ажурира глобално стање.

currentPage променљива прати тренутно активни корак форме.

setHeaderTitle функција се користи за динамичко ажурирање наслова форме у зависности од тренутног корака.

2. Ефекти приликом рендеровања:

Два **useEffect** hook-а се користе за извршавање одређених акција приликом рендеровања компоненте и промена у стању:

Први **useEffect** се извршава само једном приликом иницијалног рендеровања и поставља почетни наслов и тренутну страницу форме.

Други **useEffect** се извршава сваки пут када се промени вредност **currentPage** и ажурира наслов форме у складу са тренутним кораком. Користи низ **titles** за мапирање корака на одговарајуће наслове.

3. Приказивање садржаја форме:

renderFormPart функција одређује која компонента форме треба да се прикаже на основу вредности **currentPage**.

Користи низ **forms** који садржи компоненте за сваки корак форме.

Враћа одговарајућу компоненту из низа на основу **currentPage**, или **null** ако је **currentPage** ван опсега.

4. Враћање JSX-а:

Компонента на крају позива **renderFormPart** функцију и враћа њен резултат, што значи да ће се приказати одговарајућа компонента форме у зависности од тренутног корака.

4.2.1 Форма у три корака

Дизајнирана да би динамички кроз интуитиван интерфејс спровела корисника кроз све кораке потребне за унос нових података у малој радионици, као што су креирање нових послова, додавање нових контаката или евидентирање нових производа.

Код са слике дефинише функционалну React компоненту под називом MultiStepForm која даје структуру форме у три корака.

```
import { useEffect } from 'react'
import CreateBillForm from './forms/CreateBillForm'
import CreateContactForm from './forms/CreateContactForm'
import CreateTaskForm from './forms/CreateTaskForm'
import { useGlobalContext } from './GlobalContextProvider'

export default function MultiStepForm() {
  const { currentPage, setFormTitle, formTitleString, setHeaderTitle,
    setCurrentPage } = useGlobalContext()

  useEffect(() => {
    setHeaderTitle(formTitleString)
    setCurrentPage(0)
  }, [])

  useEffect(() => {
    const titles = ['Izaberi ili dodaj musteriju', 'Unesi podatke o poslu',
      'Napлата:']
    setFormTitle(titles[currentPage] || '')
  }, [currentPage])

  const renderFormPart = () => {
    const forms = [<CreateContactForm />, <CreateTaskForm />, <CreateBillForm
      callback={() => {}} />]
    return forms[currentPage] || null
  }

  return renderFormPart()
}
```

Слика 7. React код који дефинише форму у три корака

4.2.2 Први корак унос или одабир контакта:

Омогућава кориснику да одабере постојећи контакт или унесе нов.

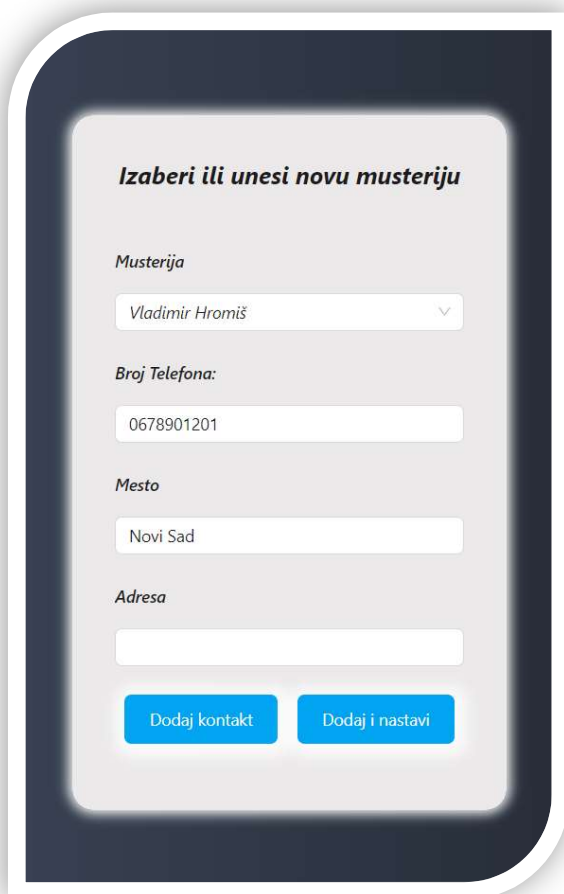
Поље за унос мустерије омогућава интерактивну претрагу у реалном времену, филтрирајући листу резултата на основу унетог текста.

Уколико се унос поклапа са постојећим купцем, корисник може да одабере тог купца са филтриране листе.

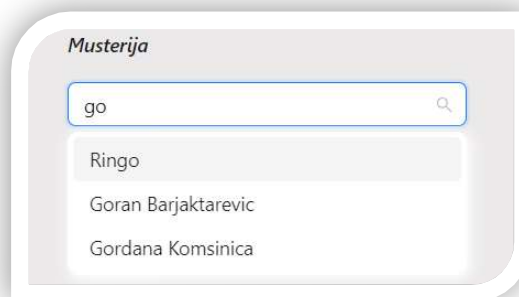
Након одабира, остала поља у форми се аутоматски попуњавају подацима из базе који припадају том купцу. У том случају, дугме за следећи корак мења наслов у „Настави” уместо „Додај и настави”.

Ако се унети подаци не поклапају са постојећим купцима, поља остају доступна за **унос нових података**, који ће бити сачувани у бази након потврде.

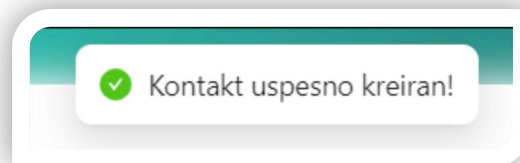
Додавање новог контакта без прелажења на следећи корак може се обавити једноставним притиском дугмета **Додај контакт** уместо **Додај и Настави** које нас води у следећи корак наплате.



Слика 10. Први корак форме



Слика 8. Унос или одабир контакта



Слика 9. Валидација успешног уноса у базу

4.2.3 Други корак Унос детаља о послу:

Омогућава кориснику да дефинише назив и детаљан опис посла.

Датум креирања посла је подразумевано постављен на тренутни датум, али корисник може да га промени по потреби.

Прекидач Status: служи за означавање статуса посла - да ли је завршен или не. Уколико је посао завршен, отвара се додатна опција за унос датума и времена завршетка и омогућава се прелазак на следећи корак - наплату. У супротном, посао се додаје на листу активних послова

Ако корисник има потребу изменити датум почетка или завршетка посла, то може лако учинити уз интерактивну компоненту која долази уз AntD. Ова компонента обезбеђује брзу селекцију потребног момента кроз неколико интерактивних елемената:

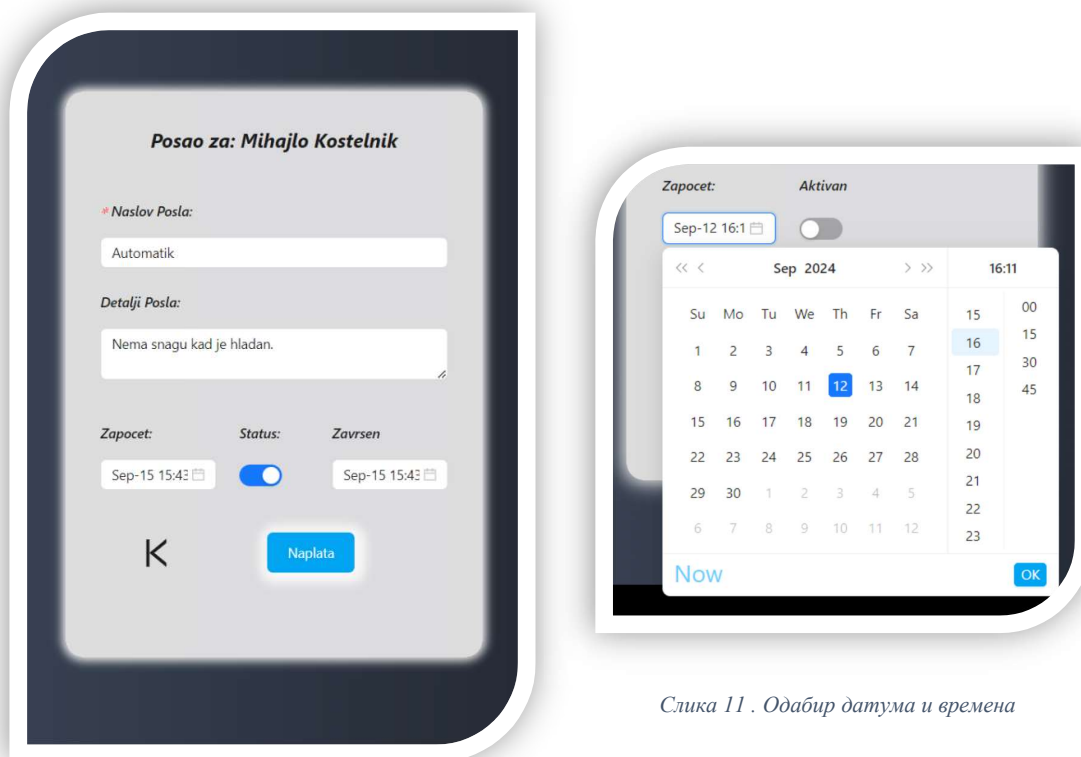
Стрелице за навигацију: Корисник може да се креће кроз месеце и године користећи стрелице.

Приказ времена: Омогућава кориснику да унесе или изабере тачно време.

Дугме "Now": Аутоматски поставља тренутни датум и време.

Дугме "OK": Потврђује избор и затвара компоненту.

Календар: Приказује месечни преглед са данима, омогућавајући кориснику да кликом изабере жељени датум и време.



Слика 11 . Одабир датума и времена

Слика 12. Други корак форме

4.2.4 Трећи корак Унос података о наплати (за завршене послове):

Омогућава кориснику да креира наплату за пружене услуге и употребљене делове.

Слика 13. Трећи финални корак форме

У наслову: Приказује име клијента за кога се креира наплата.

Цена услуге: Поље за унос цене пружене услуге.

Прекидач за означавање статуса плаћања

Датум плаћања: Поље за унос или избор датума плаћања.

Дугме:

"←": Враћање на претходни корак.

"Наплата": Покреће процес генерисања/слања наплате. Након успешног генерисања наплате, ажурира количину употребљених производа у табели "производи" како би се осигурало да увек приказује реално стање залиха.

Употребљени делови: Секција за унос детаља о кориштеним деловима.

Падајућа листа са претрагом за избор производа из базе података.

Поље за унос количине, **ограничено** на расположиву количину.

Приказ цене по јединици и укупне цене за сваки део као и њихов збир.

Израчунавање: Аутоматски приказује укупну цену изабраних делова.

Слика 14. Одабир утрошених производа

4.2.5 Унос новог производа

Компонента за унос новог производа дизајнирана је са фокусом на једноставност коришћења и прегледност. Формулар је вертикално оријентисан, са јасно дефинисаним пољима за унос података. Дугмад "Потврди" и "Откажи" су визуелно истакнута и лако доступна кориснику.

Улога: прикупља следеће информације о производу:



Назив производа: Обавезно поље за унос

Произвођач: Поље за унос назива произвођача.

Модел: Поље за унос модела производа.

Количина: Обавезно поље за унос количине производа на стању, са ограничењем на целобројне вредности.

Цена: Обавезно поље за унос цене производа, са ограничењем на целобројне позитивне вредности.

Слика 15. Форма за унос производа

SKU: Поље за унос јединствене шифре производа (Stock Keeping Unit).

Након уноса свих података, корисник може да потврди унос кликом на дугме "Потврди" или да откаже унос кликом на дугме "Откажи". Валидација унетих података се врши аутоматски, а кориснику се приказују јасне поруке у случају грешке, укључујући и проверу да ли је унети SKU (јединствени идентификатор производа) заузет. Уколико је SKU заузет, приказује се порука о грешци у искачућем прозору (popup-у).

Потврда уноса покреће асинхрони позив серверу (`ProductsService.createProductEntry`), који чува податке о новом производу у бази података. У случају успешног уноса, кориснику се приказује порука о успеху, а формулар се ресетује. У случају грешке, кориснику се приказује одговарајућа порука о грешци.

4.2.6 Табеле и Модални прозори

Апликација "Мала радионица" користи табеле за приказ и управљање кључним подацима: производима (**ProductsList**), задацима (**TaskList**), рачунима (**BillsList**) и контактима (**ContactsList**).

Свака од ових компоненти дели заједничку функционалност:

Приказ листе: Приказује податке из базе у прегледној табели.

Претрага: Омогућава корисницима брзу претрагу података по различитим критеријумима

Измена: Кликком на дугме "Измени", отвара се модални прозор за једноставну измену података.

Брисање: Кликком на дугме "Обриши", одмах након потврде покреће се процес брисања контакта

Табела производи: пружа преглед свих делова у систему, омогућавајући њихово лако претраживање, измену и брисање.

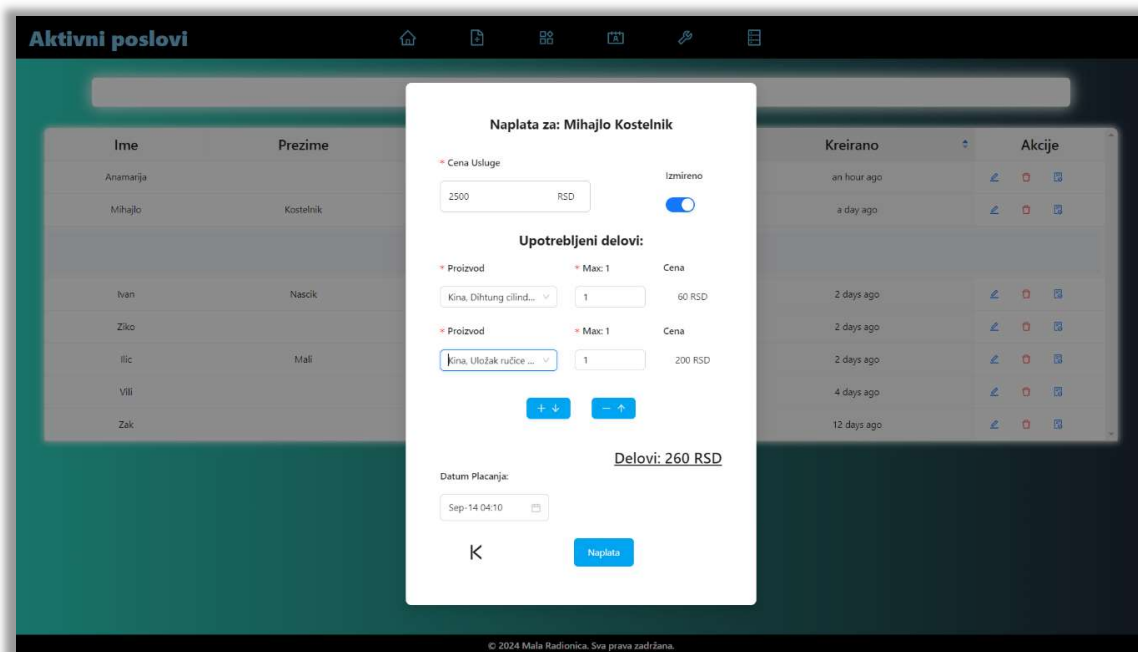
Proizvod	Proizvodjac	Model	Cena	Kolicina	SKU	Izmeni	Ukloni
Sp.26x1.75 (47-559)	Trayal	d-2	850 RSD	0	3364012	Izmeni	Ukloni
Un.27.5x1.90-2.30 50/57-584 av	Mitas	polybag	450 RSD	1	3340044	Izmeni	Ukloni
Retrovizori Veneco Master 4/Ultra AMP	Kina		1200 RSD	1	4600014	Izmeni	Ukloni
Bužir crni običan	Kina		70 RSD	25	3404408	Izmeni	Ukloni
Lulica za v-brake	Kina		60 RSD	10	3404000	Izmeni	Ukloni
Sp.26x1.1/2 (40-635) crna guma	velotyre	pah free	650 RSD	2	3330120	Izmeni	Ukloni
Set handlefz za montažu na volan za dečja sedla	Kina	tattoo plus	2500 RSD	0	3709041	Izmeni	Ukloni
Sedlo Veneco Master 4/Ultra AMP	Kina		1550 RSD	0	4600013	Izmeni	Ukloni
Kočnica zadnja doboš sa bravom Harmony	Kina		2600 RSD	1	4600418	Izmeni	Ukloni
Kočnica i brava zadnjeg točka mini power	Kina		1900 RSD	1	4600072	Izmeni	Ukloni
Zadnja doboš kočnica 110 za Extreme i Pulse	Kina		2450 RSD	1	4600209	Izmeni	Ukloni
Šolja zadnja mtb 3/8" bw	Kina		60 RSD	5	3205302	Izmeni	Ukloni
Pločice za disk kočnice za avdi	Kina		750 RSD	0	3405012	Izmeni	Ukloni

Ukupno 101 proizvoda < 1 2 3 4 5 *** 8 >

© 2024 Mala Radionica. Sva prava zadržana.

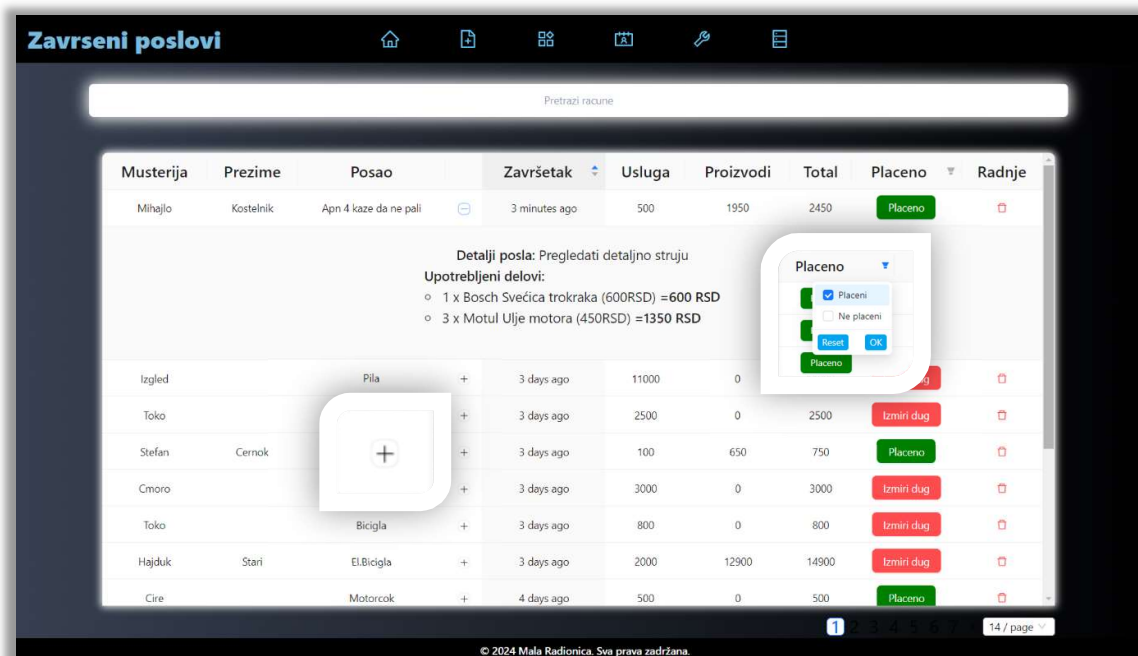
Слика 16. Табела Производи

Табела активни послови: Ова компонента има предходно поменуте функционалности и додатну акцију "Наплата" која у модалу отвара форму за креирање рачуна за изабрани задатак, коју смо у претходном поглављу детаљно описали.



Слика 17. Табела активни послови

Табела завршени послови: Ова компонента има додатну функционалност "Означи као плаћено" и филтере за преглед плаћених/неплаћених рачуна. Као и додатне детаље које можемо погледати за сваки листиг притиском дугмета + приказаном на слици.



Слика 18. Табела завршени послови

4.2.6.1 Целокупан дизајн

Све компоненте су дизајниране са фокусом на једноставност и прегледност. Дугмад за акције су јасно видљива. Модални прозори за измену података су интуитивни и олакшавају унос и ажурирање информација.

Иако постоје мање разлике у функционалностима, све компоненте за приказ табела у апликацији "Мала радионица" деле заједничку структуру и дизајн, што доприноси конзистентном корисничком искуству.

Ant Design библиотека је омогућила брз развој лепо дизајнираног и конзистентног корисничког интерфејса. Њене компоненте су се показале као флексибилне и лако прилагодљиве специфичним потребама апликације.

4.2.6.2 Структурирање UI-а кроз компоненте:

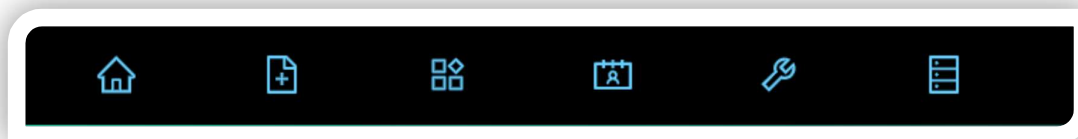
Модуларност: Пројекат је подељен на мање, независне компоненте, што олакшава развој, тестирање и одржавање.

Примери компоненати су **Header**, **HomePage**, **CreateNewProductFormComponent**, **ProductsList**, **TaskList**, **BillsList**, **Footer**, итд.

4.2.6.3 Рутирање

У компоненти **Header** се користи **Link** компонента из react-router-dom библиотеке за креирање линкова ка различитим деловима апликације. Сваки линк има дефинисану "to" особину која одређује путању (/ , /ProductCreate, /ProductList, итд.). Када корисник кликне на линк, react-router-dom ће се побринути да се прикаже одговарајућа компонента у зависности од URL-а.

Овај приступ омогућава једноставну навигацију кроз различите делове апликације, а истовремено поједностављује развој и одржавање кода.



Слика 19. Header компонента

4.2.7 Интеракција са backend-ом користећи axios:

Frontend апликација комуницира са backend-ом путем HTTP захтева користећи Axios библиотеку. За сваки руту у апликацији (рачуни, контакти, задаци, производи) дефинисан је сервис (**BillService**, **ContactsService**, **TaskService**, **ProductsService**) који садржи функције за извршавање CRUD (Create, Read, Update, Delete) операција над тим ентитетом.

Сваки сервис типично имплементира следеће функције:

getAll[Ентитет](): Шаље GET захтев да би преузела листу свих ентитета са backend-а.

Пример функције getAll из руте **ContactsService**

```
const getAllCustomers = async () => {return await
axios.get<IContactsResponse[]>(baseUrl + '/contacts').catch((error) => {throw
axios.isAxiosError(error) ? error.response?.data : error})}
```

Слика 20. Функција getAll из руте ContactService

create[Ентитет](data): Шаље POST захтев са подацима о новом ентитету да би га креирала на backend-у.

Пример функције create из руте **BillService**

```
const createBill = async (data: IBillResponse) => {
  return await axios.post<IBillResponse>(baseUrl + '/bills', data).catch((error)
=> {throw axios.isAxiosError(error) ? error.response?.data : error})}
```

Слика 21. Функција create из руте BillService

update[Ентитет](id, data): Шаље PUT захтев са ажурираним подацима о ентитету.

Пример функције update из руте **ProductService**

```
const getProductById = async (id: string | number) => {try {const response =
await axios.get<IProduct>(`${baseUrl}/Products/${id}`)return response.data} catch
(error) {console.error('Error fetching product:', error)throw error}}
```

Слика 22. Функција update из руте ProductService

get[Ентитет]ById(id): Шаље GET захтев да би преузела један ентитет са одређеним ID-јем.

Пример функције getById из руте **ProductService**

```
const getProductById = async (id: string | number) => {try {const response =
await axios.get<IProduct>(`${baseUrl}/Products/${id}`)return response.data} catch
(error) {console.error('Error fetching product:', error)throw error }}
```

Слика 23. Функција getById из руте ProductService

delete[Ентитет](id): Шаље DELETE захтев да би обрисала ентитет са одређеним ID-јем.

Пример функције delete из руте **ProductService**

```
const deleteProduct = async (id: number) => {return await
axios.delete<IProduct[]>(baseUrl + '/Products/' + id).catch((error) => {throw
axios.isAxiosError(error) ? error.response?.data : error})}}
```

Слика 24. Функција delete из руте ProductService

Поред ових генеричких CRUD функција, неки сервиси имају и додатне функције специфичне за тај ентитет. На пример:

BillService има функцију **markAsPaid** за означавање рачуна као плаћеног.

```
const markAsPaid = async (id: number) => {
  return await axios.put(`${baseUrl}/bills/${id}`).catch((error) => {
    throw axios.isAxiosError(error) ? error.response?.data : error})}
```

Слика 25. Функција markAsPaid из руте BillService

ProductsService има функцију **updateMultipleProducts**: Шаље POST захтев ка /quantitySubtract руту, да би ажурирала количину производа након креирања рачуна. Backend би требало да обради податке о продатим производима, ажурира залихе у бази података и врати потврду о промени или разлог грешке ако до ње дође.

```
const updateMultipleProducts = async (productsToUpdate: { id: number; quantity:
number }[]) => {try {const response = await
axios.put<IProduct[]>(`${baseUrl}/Products/updateMultiple`, productsToUpdate)
return response.data} catch (error) {console.error('Error updating multiple
products:', error)throw error}}
```

Слика 26. Функција UpdateMultipleProducts из руте ProductService

Све функције у сервисима користе async/await синтаксу за асинхроно руковање захтевима и одговорима. Такође, све функције обрађују потенцијалне грешке (catch блок) како би се спречило рушење апликације у случају проблема са backend-ом.

Све функције користе async/await синтаксу за асинхроно руковање захтевима и одговорима. Такође, све функције обрађују потенцијалне грешке (catch блок) како би се спречило рушење апликације у случају проблема са backend-ом.

4.2.8 TypeScript за побољшање квалитета кода

TypeScript је фундаментални део овог frontend пројекта, доприносећи његовој робустности и одрживости. Ево конкретних примера његове примене:

4.2.8.1 Дефинисање типова података:

Интерфејси за моделе података: У фолдеру `src/model` налазе се интерфејси попут **ITaskResponse**, **IProductResponse**, **IContactResponse** и **IBillResponse**. Ови интерфејси дефинишу структуру података које апликација размењује са backend-ом.

```
export interface IContactsResponse
{
  id: number
  firstName: string
  lastName: string
  phoneNumber: string
  city: string
  address: string }

```

Слика 27. Пример интерфејса *IContactResponse*

4.2.8.2 Типизација променљивих и функција:

У целокупном коду пројекта, променљиве, параметри функција и повратне вредности су типизирани.

Пример (TaskList.tsx):

```
const [editingTask, setEditingTask] = useState<ITaskResponse>({} as
ITaskResponse);
const handleEdit = (record: ITaskResponse) => { ... };

```

Слика 28. Пример типизације у *TaskList* компоненти

4.2.8.3 Зашто је коришћен TypeScript-а:

Спречавање грешака: TypeScript-ова провера типова спречава додељивање некомпатибилних вредности променљивама, што смањује број грешака током извршавања кода.

Побољшана читљивост кода: Типизација чини код читљивијим и лакшим за разумевање, јер је јасно дефинисано које типове података функције очекују и враћају.

Лакше рефакторисање: Промене у коду су безбедније уз TypeScript, јер ће се грешке услед некомпатибилних типова открити током компилације.

4.3 Backend развој

Backend део ове апликације је, задужен за обраду података, пословну логику и комуникацију са базом података.

4.3.1 Обрада HTTP захтева

Backend део апликације је конфигурисан да обрађује различите HTTP захтеве:

GET: За преузимање података са сервера.

POST: За слање нових података на сервер.

PUT: За ажурирање постојећих података на серверу.

DELETE: За брисање података са сервера.

Сваки HTTP захтев се усмерава на одговарајућу API руту дефинисану у Express.js апликацији. Backend део затим обрађује захтев, комуницира са базом података и враћа одговор frontend-у у JSON формату.

4.3.2 Express.js API руте и њихове функционалности

У наставку следи преглед свих API рута у овом пројекту, са фокусом на њихову улогу и начин имплементације. Поједине руте биће илустроване конкретним примерима кода помоћу слика кода.

4.3.2.1 Управљање контактима:

GET /contacts: Преузима све контакте из базе.

POST /contacts: Креира нови контакт.

PUT /contacts/:id: Ажурира постојећи контакт.

DELETE /contacts/:id: Брише контакт.

4.3.2.2 Управљање производима:

GET /products: Преузима све производе.

GET /products/:id: Преузима један производ на основу ID-ја

POST /products: Креира нови производ

PUT /products/:id: Ажурира постојећи производ

DELETE /products/:id: Брише производ.

4.3.2.3 Управљање рачунима:

GET /bills: Преузима све рачуне, укључујући податке о контакту, послу и коришћеним производима

POST /bills: Креира нови рачун, уз израчунавање укупне цене.

PUT /bills/:bill id: Ажурира постојећи рачун и повезани задатак

DELETE /bills/:id: Брише рачун.

4.3.2.4 Управљање задацима:

POST /Tasks je ruta koja unosi novi zadatak u bazu podataka koristeći informacije iz захтева. Пример кода са са слике приказује имплементацију POST /tasks руте која додаје нови задатак у базу и враћа га као JSON одговор. Или ако дође до грехе враћа се информативна порука о грешци која се приказује у конзоли ради лакшег проналажења њеног узрока.

```
router.post("/", (req, res) => {  
    const { id, contact_id, job_name, job_description, creation_date } = req.body;  
    const insertQuery = "INSERT INTO `tasks` ( `contact_id`,  
    `job_name`, `job_description`, `creation_date`) VALUES ( ?, ?, ?, ?);";  
    db.query(insertQuery, [contact_id, job_name, job_description, creation_date],  
        (err, result) => {if (err) {res.status(500).json({ error: `Error inserting  
task: ${err.message}` });} else {const selectQuery = "SELECT * FROM tasks WHERE  
id = ?";db.query(selectQuery, [result.insertId], (err, newTask) => {if (err)  
{res.status(500).json({ error: `Error retrieving new task: ${err.message}` });}  
else {console.log("Newly added task:",  
newTask[0]);res.json(newTask[0])};}});}});});
```

Слика 29. Имплементација `POST /tasks` руте

4.3.2.5 Ажурирање количине производа:

POST /products: Ажурира количине више производа истовремено, водећи рачуна о расположивом стању.

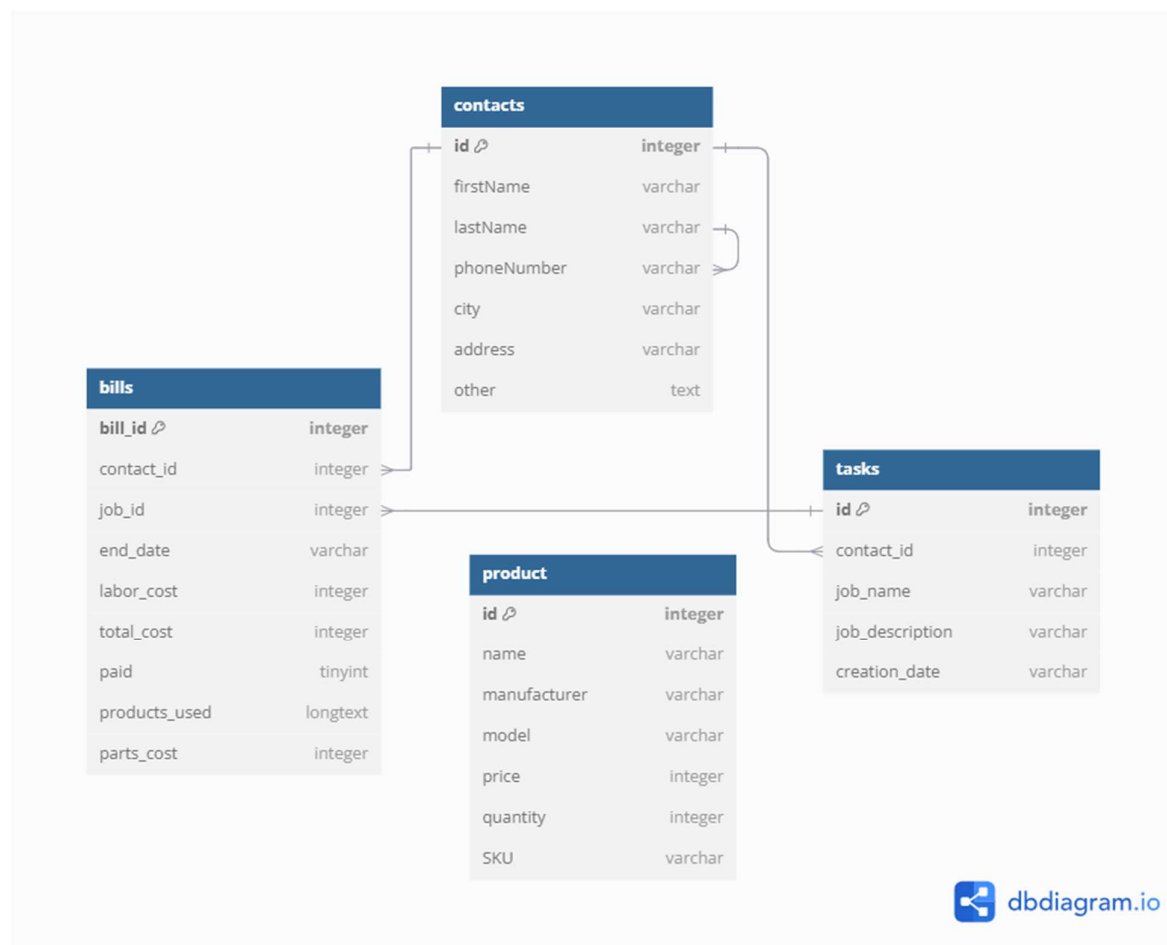
GET /Tasks: Додава листу свих задатака који немају повезан **bill_id**, додајући име и презиме клијента из табеле **contacts** на основу **contact_id** који је сачуван приликом креирања посла.

```
const selectQuery = `SELECT t.*, c.firstName,c.lastName
  FROM tasks t LEFT JOIN contacts c ON t.contact_id = c.id
  LEFT JOIN bills b ON t.id = b.job_id
  WHERE b.job_id IS NULL;`;
```

Слика 30. Имплементација руте GET/Task

4.3.3 MySQL база података

База података под називом **mala-radionica** састоји се од неколико табела које су међусобно повезане и служе за управљање пословима, производима, контактима и рачунима у малој радионици. Свака табела има своју структуру и улогу у систему.



Слика 31. Визуелни приказ базе података

[6]

bills ↔ **contacts**: Сваки рачун (bills) је повезан са једним контактом (contacts) путем поља **contact_id**. То значи да сваки рачун мора имати одговарајућег купца.

bills ↔ **tasks**: Сваки рачун (bills) је повезан са одређеним послом (tasks) путем поља **job_id**, што омогућава праћење који посао је фактурисан.

tasks ↔ **contacts**: Сваки посао (tasks) је повезан са једним контактом (contacts) путем поља **contact_id**. То омогућава праћење који купац је наручио одређени посао.

Дијаграм базе података је креиран помоћу алата dbdiagram.io.

5. ЗАКЉУЧАК

Израда full-stack веб апликације за праћење послова, робе и финансија "мале радионице" омогућила ми је да продубим своје знање о различитим аспектима развоја софтвера, користећи технологије React.js, Node.js, Express.js и MySQL. Ова комбинација технологија показала се као ефикасан избор за креирање интерактивног корисничког интерфејса, стабилног бекенда и поуздане базе података.

Током рада на пројекту, суочио сам се са изазовима везаним за структуру кода и међусобне везе између модела. Недостатак детаљног планирања у почетној фази довео је до тога да апликација има комплексну структуру која не омогућава лаку интеграцију нових функционалности. Иако сам успео да решим главне проблеме и постигнем функционалан систем, сада схватам да би било корисно да сам у старту посветио више времена планирању архитектуре апликације, посебно моделовању података.

Током развоја сам имао више идеја које нисам стигао или могао да имплементирам у преосталом времену, управо због структуре кода која није довољно флексибилна за ефикасну интеграцију тих нових функционалности. Сваки пут када се дода нова функционалност, модели који подржавају те функције морају се мењати и прилагођавати. Ово је кључна лекција коју сам научио — што више функција апликација добија, то је важније да модели буду добро осмишљени од почетка, како би се избегли проблеми са проширивошћу и одржавањем.

У будућим пројектима, посветићу посебну пажњу планирању архитектуре пре него што започнем са писањем кода. Израда UML дијаграма и других визуелних приказа односа између модела и функционалности би ми значајно уштедела време и поједноставила процес интеграције нових идеја. Тиме бих избегао ситуације у којима касније промене доводе до преобимних и неефикасних измена у постојећој структури кода.

Ово искуство ми је показало колико је важно добро осмислити сваки аспект апликације, како би процес развоја био што ефикаснији, а апликација одржива и спремна за будуће надоградње.

6. ЛИТЕРАТУРА

1. React. (n.d.). Introducing JSX. pristupljeno 3. avgusta 2024. sa <https://react.dev/learn/writing-markup-with-jsx>
2. Vukićević, N. (n.d.). JavaScript i DOM. _codeblog.rs_. 3. avgusta 2024. sa https://codeblog.rs/clanci/javascript_i_dom
3. Express/Node introduction - Learn web development | MDN (mozilla.org), pristupljeno 6. avgusta 2024.
4. <https://dev.mysql.com/doc/refman/8.4/en/mysql-acid.html> pristupljeno 11. avgusta 2024. (MySQL 8.4 Reference Manual :: 17.2 InnoDB and the ACID Model)
5. 14. Scott, B., & Neil, T. (2009). _Designing Web Interfaces_. O'Reilly Media, Inc.
6. dbdiagram.io. (Pristupljeno 1. septembra 2024). <https://dbdiagram.io/>

Порекло слика:

Слика 1 - 30: Ауторска фотографија

Слика 31 dbdiagram.io. (Pristupljeno 1. septembra 2024). <https://dbdiagram.io/>

ИЗЈАВА О АУТОРСТВУ

Ја, ниже потписани студент (име и презиме) _____, изјављујем да је овај завршни рад плод самосталног рада. Коришћена литература је наведена на крају рада.

У Новом Саду, _____.____.20__.

Студент (потпис):
