



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

по професия код 481020 „Системен програмист“
специалност код 4810201 „Системно програмиране“

Тема: Електронна платформа за продажба на занаятчийски
произведения

Дипломант:

Никола Иванов Петров

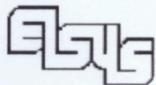
Дипломен ръководител:

Стеван Атанасов

СОФИЯ

2024

Задание



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Дата на заданието: 15.11.2023 г.
Дата на предаване: 15.02.2024 г.

Утвърждавам:
/проф. д-р инж. П. Якимов/

ЗАДАНИЕ за дипломна работа

ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ
по професия код 481020 „Системен програмист“
специалност код 4810201 „Системно програмиране“

на ученика Никола Иванов Петров от 12 В клас

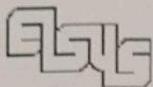
1. Тема: Електронна платформа за продажба на занаятчийски произведения
2. Изисквания:
 - Страници за влизане и регистрация за потребителите и занаятчии;
 - Каталог с произведенията на всеки занаятчия;
 - Възможност за последване на занаятчии и интеракция с произведенията им;
 - Линия на времето, която показва етапите през които е минал продуктът от идеята до неговия завършен вид;
 - Възможност за поръчки към занаятчията.
3. Съдържание 3.1 Теоретична част
 3.2 Практическа част
 3.3 Приложение

Дипломант:
/ Никола Петров /

Ръководител:
/ Стефан Атанасов /

ВРИД Директор:
/ ст. пр. д-р Веселка Христова /

Отзив



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

СТАНОВИЩЕ КЪМ ДИПЛОМНА РАБОТА

Тема на дипломната работа: Електронна платформа за продажба на занаятчийски произведения

Ученник: Никола Иванов Петров

Клас: 12B

Професия: 481020 "Системен програмист"

Специалност: 4810201 "Системно програмиране"

Дипломен ръководител: Стефан Атанасов

Никола се е справил отлично с разработката на дипломния проект, като е реализирал напълно поставените в заданието изисквания и функционалности, като е разработил уеб приложение на React, използващо възможностите на Next.js, както и Firebase за автентикация и съхранение на данни. Въпреки че Next и Firebase бяха нови технологии за него, той успя за много кратко време да се запознае с тях и да започне имплементацията на всички модули от заданието.

В началото отдели повече време на анализа на технологиите и научаването им чрез документация, видео уроци и примерни проекти.

Във всеки момент се е отнасял съвестно и отговорно към поставените задачи, които бяха изпълнени в срок и с необходимото качество, като често е надминавал минималните изисквания и е влагал повече усилия с цел да разработи отлична дипломна работа.

Предлагам за рецензент Мартин Савов, старши разработчик на софтуер в Аполика, msavov@apolica.com, 0884 686 478

Всичко изложено дотук дава основание ученикът Никола Петров да бъде допуснат до защита пред комисията за провеждане на държавен изпит за придобиване на професионална квалификация по теория и практика на специалността и комисията да оцени дипломната работа отлично.

14.02.2024

Дипломен ръководител:.....

/Стефан Атанасов/

Увод

Целта на дипломната работа е да се изработи уеб приложение наречено Wavary, което да подпомогне на занаятчии да представят своите изделия. Главната идеята и функционалност на проекта е да се съберат ръчно направени продукти на едно място за по-лесен достъп от страна на потребителите и по-добро представяне пред клиентите. В допълнение ако потребител хареса работата и стилът на даден артист може да направи запитване за изработка на продукт. По този начин уеб приложението помага и с по-лесна организация от страна на занаятчиите, защото поръчките, продуктите и цялата общност от артисти се намират на едно място.

Wavary трябва да поддържа създаването на акаунт на потребител и механизъм за обозначаването му като занаятчия. Клиентът да може да прави запитване към занаятчия, в случай че желае да му бъде изработен даден продукт. Занаятчията да има възможност за добавяне на продукт и за правене на запитване към други артисти. Продуктите на всеки занаятчия трябва да се намират в каталог като всеки артикул трябва да има заглавие, кратко описание, цена, линия на времето, която показва процесът на изработване на даденото изделие и етикети за по-лесното ориентиране на потребителите. Всяко запитване към занаятчия чака да бъде одобрено или отказано от артиста. Докато заявката е със статус “чакаща”, потребителят да може да я редактира или да я отмени. Освен това уеб приложението трябва да предлага възможността за последване на занаятчия, за да се вижда кой занаятчия е популярен. Трябва да се поддържа и възможност за харесване на продукти и съответно те да бъдат запазвани в профила на потребителя.

Първа глава

Проучване

1.1. Основни принципи и технологии за реализиране уеб приложения

1.1.1. Основни технологии за разработка на уеб приложения

HTML

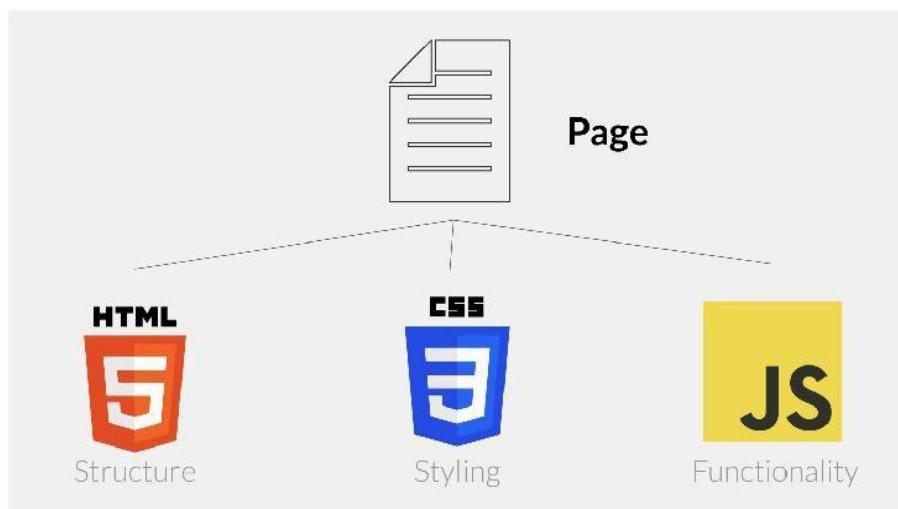
HyperText Markup Language е най-широко използваният език за писане на интернет страници. HTML използва тагове, за да маркира определени елементи от интернет страниците и да позволи тяхното представяне като заглавия, параграфи, списъци, хипервръзки и др. Уеб браузърът чете таговете, за да подреди съдържанието и изгради уеб страницата на екрана на потребителя.

CSS

Cascading Style Sheets е език за описание на стиловете. Базиран е на правила - определяте правилата, като посочвате групи от стилове, които трябва да се прилагат към определени елементи или групи от елементи на уеб страницата. Стиловете, указанi чрез CSS, определят как ще изглеждат на екрана елементите на един HTML документ (интернет страница). Пример за стилове са промяна на цвета, шрифта и големината на текста. Благодарение на този език можем да реализираме замисления ни дизайн за нашето уеб приложение.

JavaScript

JavaScript е интерпретиран език за програмиране, разпространяван с повечето уеб браузъри. Прилага се към HTML кода на интернет страница с цел добавяне на функционалност. Така нашето уеб приложение “оживява” като JavaScript ни позволява да извършваме различни действия като да комуникираме с потребителите чрез приложението при натискане на бутон или изпращане на форма.



Фиг. 1.1

1.1.2. Библиотеки за разработка на фронтенд на уеб приложения

React

React е библиотека за JavaScript, създадена и поддържана от Facebook. React улеснява изграждането на потребителски интерфейси, използвайки индивидуални парчета код, наречени компоненти. Ключовото му предимство е че всеки компонент може да се преизползва, което прави целия процес на разработка по-бърз и по-приятен. React използва виртуален DOM (фигура 1.2), който обновява само компонентите, чието състояние е било променено вместо цялата страница. JSX, препоръчваният маркъп синтаксис за React, позволява писането на HTML елементи в JavaScript, което улеснява разработката и четливостта на компонентите.

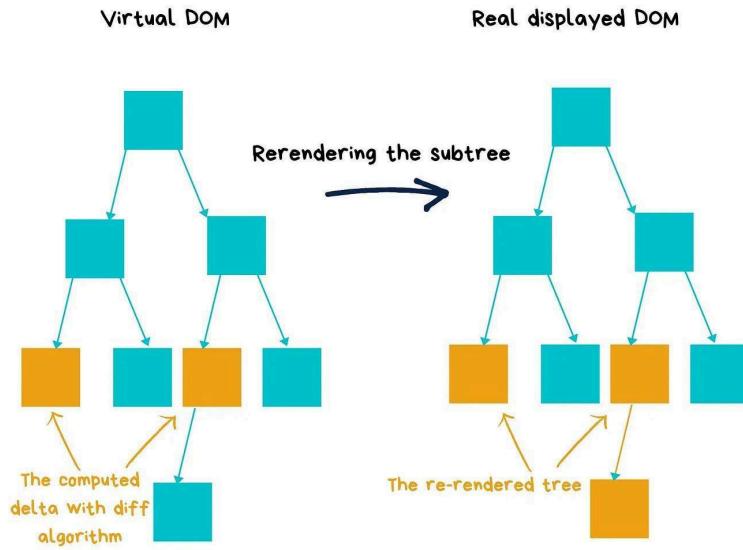


Fig. 1.2

Vue

Vue е JavaScript фреймуърк за създаване на потребителски интерфейси, който споделя някои концепции с React, но също така предлага свои уникални характеристики. Както React, така и Vue използва виртуален DOM (Фигура 1.3). Разликата е в това че Vue използва HTML шаблони вместо JSX синтаксис, което го прави достъпен и приятен за начинаещи програмисти.

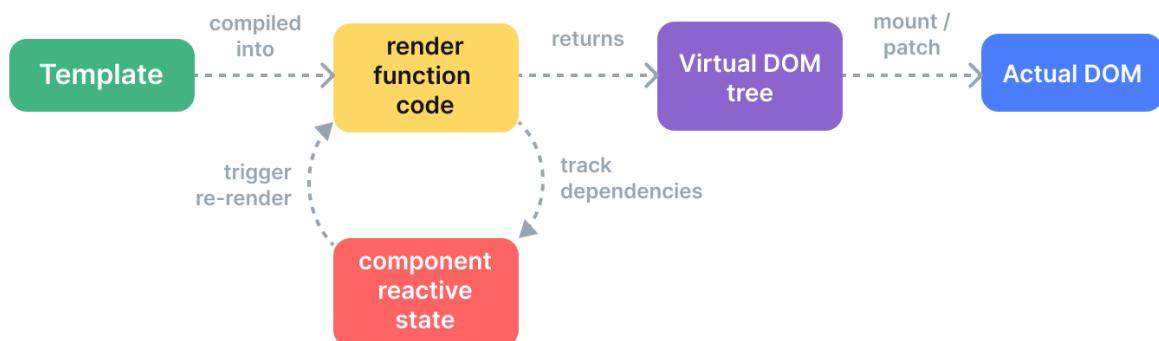


Fig. 1.3

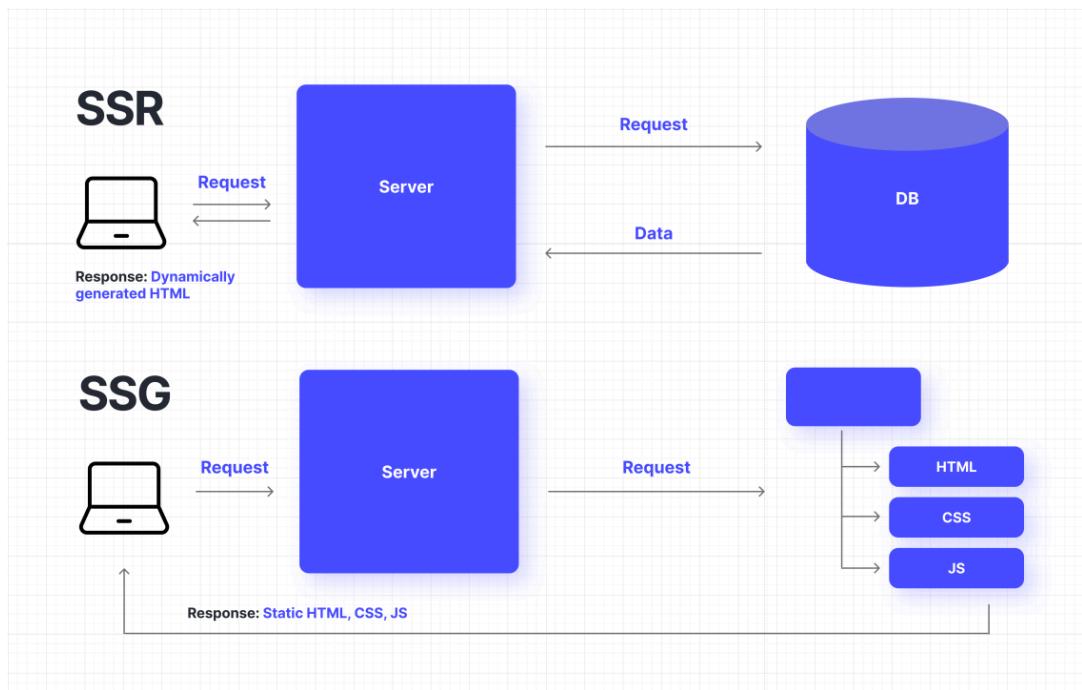
Angular

Angular е Javascript фреймуърк, създадена и поддържана от Google. Той предоставя цялостно решение за създаване на уеб приложения, базирано на компоненти. Една от ключовите разлики с React и Vue е, че Angular използва TypeScript, предоставящ статично типизиране, което подобрява сигурността и поддържаемостта на кода. За разлика от Vue и React, Angular използва реален DOM, което прави визуализацията на уеб апликации по-бавна.

Next.js

Next е React фреймуърк за изграждането на фулстак уеб апликации. Той предоставя допълнителни функции и оптимизации. Предимствата на Next са:

- Маршрутизиране - пътищата в уеб приложението са базирани на файловата система на проекта
- Визуализация - Next предоставя рендериране от страна на сървъра и статично генериране (Фигура 1.4). Рендерирането от страна на сървъра включва генериране на HTML от сървъра при всяка заявка от страна на клиента, което позволява динамичното и актуално съдържание. Статичното генериране е метод, при който уеб сайт се генерира предварително като статични HTML файлове, които се поддържат и доставят директно от CDN.
- Оптимизации за изображения, шрифтове и скриптове
- Бързо обновяване - Next обновява компонентите, които са променени и позволява на програмиста да види промяната мигновено по време на разработка.
- Поддържа JavaScript и TypeScript



Фиг. 1.4

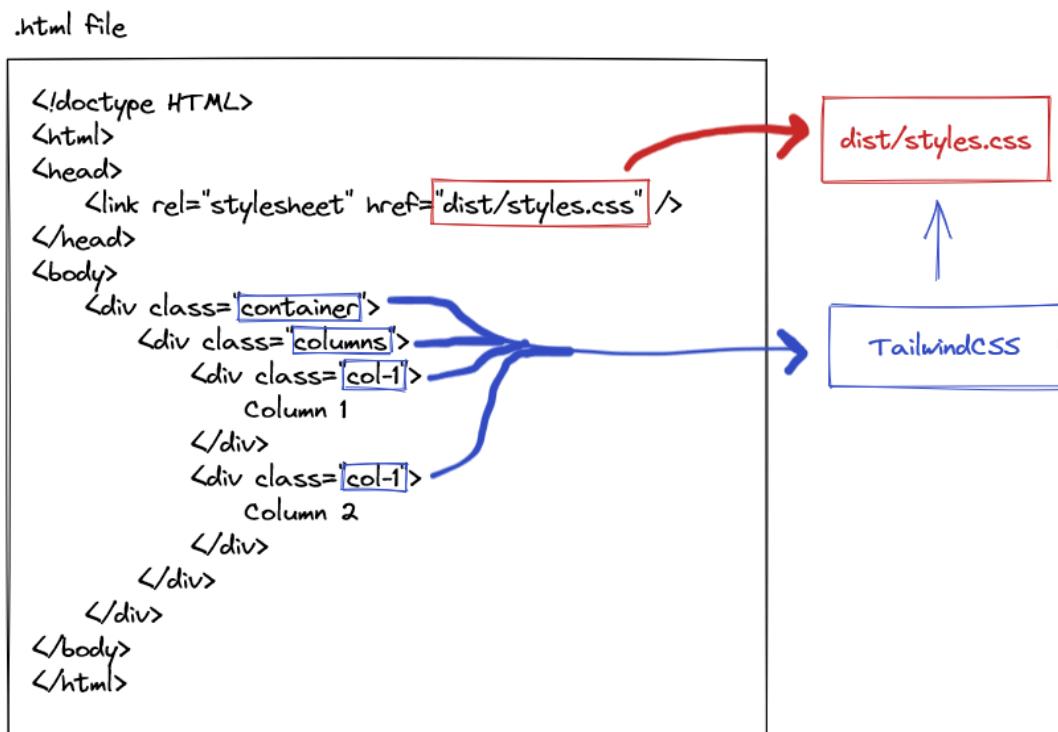
Nuxt.js

Nuxt е Vue фреймуърк за изграждането на фулстак уеб приложения. Той предоставя допълнителни функции и оптимизации. Предимствата му са подобни на Next - Маршрутизиране, базирано на файловата система на проекта, бързо обновяване, рендериране от страна на сървъра и статично генериране, но в допълнение Nuxt има автоматично импортиране на компоненти. Основната му разлика с Next е че Nuxt е предназначено за работа с Vue, докато Next с React.

Tailwind CSS

Tailwind е utility-first CSS фреймуърк, който се използва за стилово оформление на уеб приложения. Вместо предварително дефинирани CSS класове, Tailwind предоставя множество класове от базовите стилове като шрифтове, цветове, отстояния и размери. Този метод на стилово оформление позволява на програмистите бързо и лесно да създават уникални дизайни, като същевременно се осигурява гъвкавост и яснота в

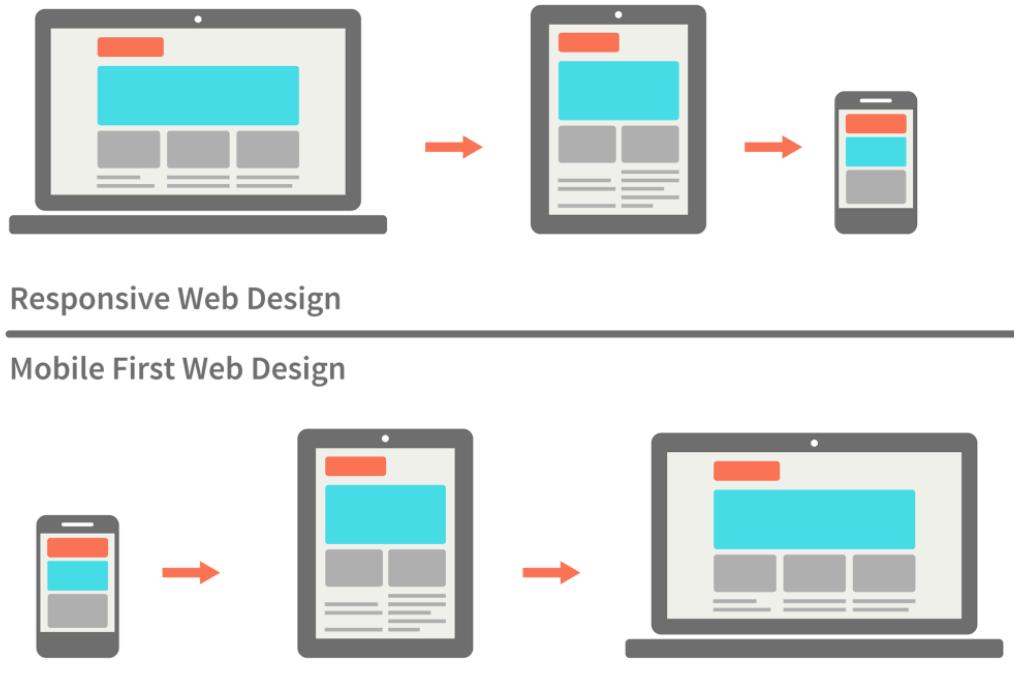
кода. Tailwind функционира чрез сканиране на всички HTML файлове, JavaScript компоненти и всички други шаблони за имена на класове. След това генерира съответните стилове и ги записва в статичен CSS файл. На фигура 1.5 е показана разликата между Tailwind и CSS съответно обозначени в синьо и червено.



Фиг. 1.5

Bootstrap

Bootstrap е CSS фреймуърк, специално създаден за разработка на фронтенд уеб приложения с "mobile-first" подход (Фигура 1.6). Основната му цел е да улесни процеса на създаване на адаптивни уеб страници, предоставяйки обширна колекция от синтаксис за дизайн на шаблони. Този инструмент предоставя готови компоненти, стилове и JavaScript плъгини, които значително оптимизират и ускоряват разработката на уеб приложението.



Фиг. 1.6

Material UI

Material UI е отворена библиотека за компоненти на React, предлагаша богат и модерен набор от готови за използване елементи, вдъхновени от Material Design на Google. Тази библиотека е създадена с цел да улесни и ускори процеса на разработка на потребителски интерфейси. Tailwind CSS и Bootstrap, от друга страна, предоставят класове и компоненти, които могат да бъдат стилизиирани по желание на програмиста.

1.1.3. Технологии за разработка на бекенд на уеб приложения

Firestore

Firebase е платформа или по-точно бекенд като услуга за разработване на приложения, която предоставя хоствани бекенд услуги, които улесняват и ускоряват процеса на разработка на приложения. Една от ключовите услуги, предоставяни от Firebase, е Firestore - база данни в реално време, предлагаща документно ориентиран подход за съхранение на данни и реално временна синхронизация между клиента и сървъра. С

автентификацията Firebase предоставя сигурен начин за управление на идентичността на потребителите в приложенията. Тази услуга поддържа различни методи за удостоверяване, включително електронна поща, социални мрежи, номера на телефони и други. Firebase също така предоставя услуга за съхранение в облак. Тази услуга е изключително полезна за съхранение на изображения, видео файлове и други мултимедийни съдържания в облака, освобождавайки ни от необходимостта за създаване и поддържане на собствени инфраструктури за съхранение.

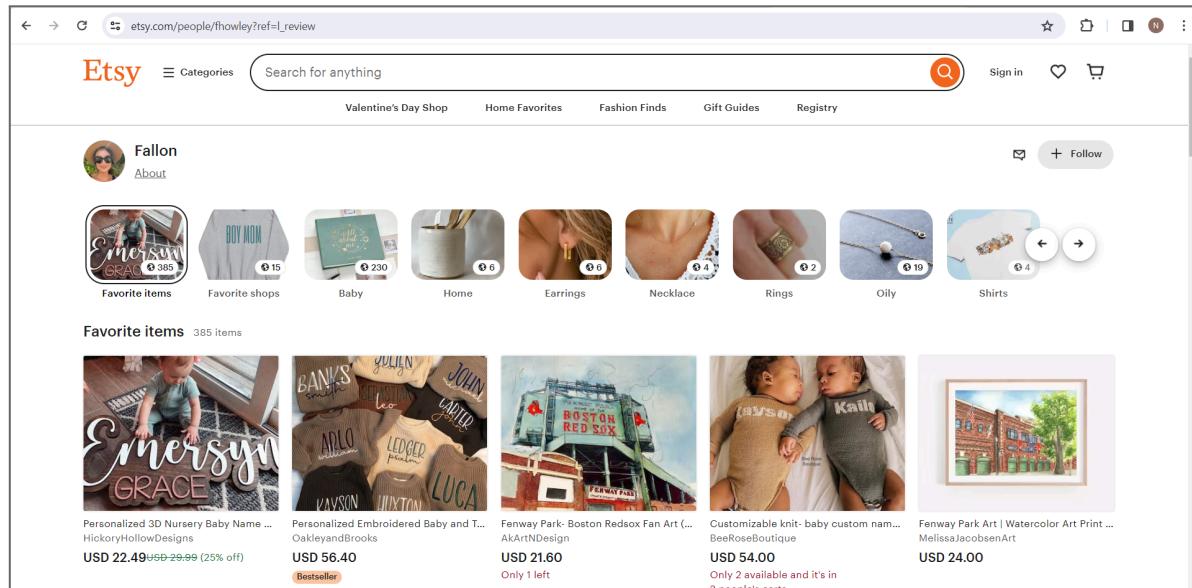
AWS Amplify

AWS Amplify е платформа или по-точно бекенд като услуга за разработване на приложения в облачна среда, като включва интегрирани услуги за бази данни, съхранение на файлове и удостоверяване. За съхранение на данни, платформата предлага гъвкавостта на NoSQL базата данни Amazon DynamoDB, която осигурява бърз и скалируем метод за съхранение и извлечане на информация. За съхранение на файлове, AWS Amplify предоставя услуга, която позволява лесно и сигурно да управлявате мултимедийни и други файлови ресурси в облака. Тази услуга може да се интегрира с различни видове файлове и предоставя средства за управление на техния животен цикъл, съхранение и достъп. Относно удостоверяването, AWS Amplify предоставя интегрирани решения за автентикация, които включват различни методи на удостоверяване. Платформата поддържа електронна поща, социални мрежи, номера на телефони и други методи, които ви позволяват да управлявате идентичността на потребителите във вашите приложения.

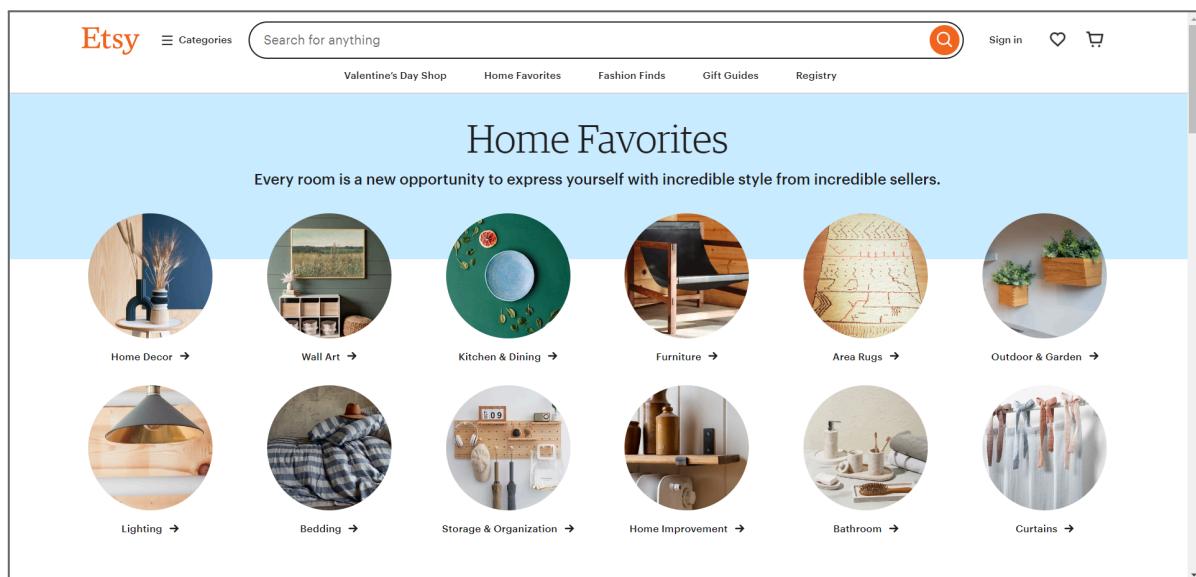
1.2. Съществуващи решения и реализации

1.2.1. Etsy

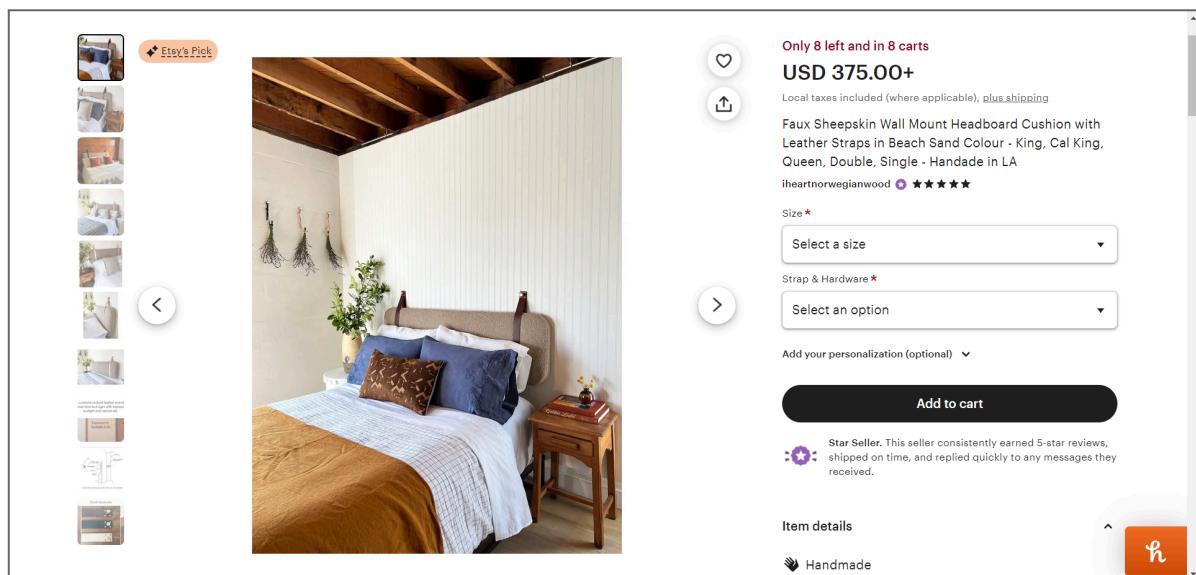
Etsy е онлайн търговска платформа, специализирана в продажбата на ръчно изработени и уникални продукти. Функционалностите на приложението се доближават до моят проект по идеята за уникалност и ръчна изработка на продукта. Разликата е че Etsy обхваща голям брой от разнообразни категории и продукти, докато Wavary е съсредоточен върху занаятчиите, което подпомага представянето на продуктите си и тяхната продажба. Още една съществена разлика е броя на акаунтите - в Wavary има 2 вида акаунти - клиент и занаятчия, докато в Etsy са 5 - потребителски акаунт, акаунт за артисти, акаунт за купувач на едро, Etsy Plus акаунт и Etsy Pro акаунт. Както в Wavary всеки занаятчия има каталог с продуктите си (фигура 1.7) в Etsy всеки артист има "about" страница, която показва интересите и продаваните продукти на артиста. На фигури 1.7, 1.8 и 1.9 е показан потребителският интерфейс на уеб приложението.



Фиг. 1.7



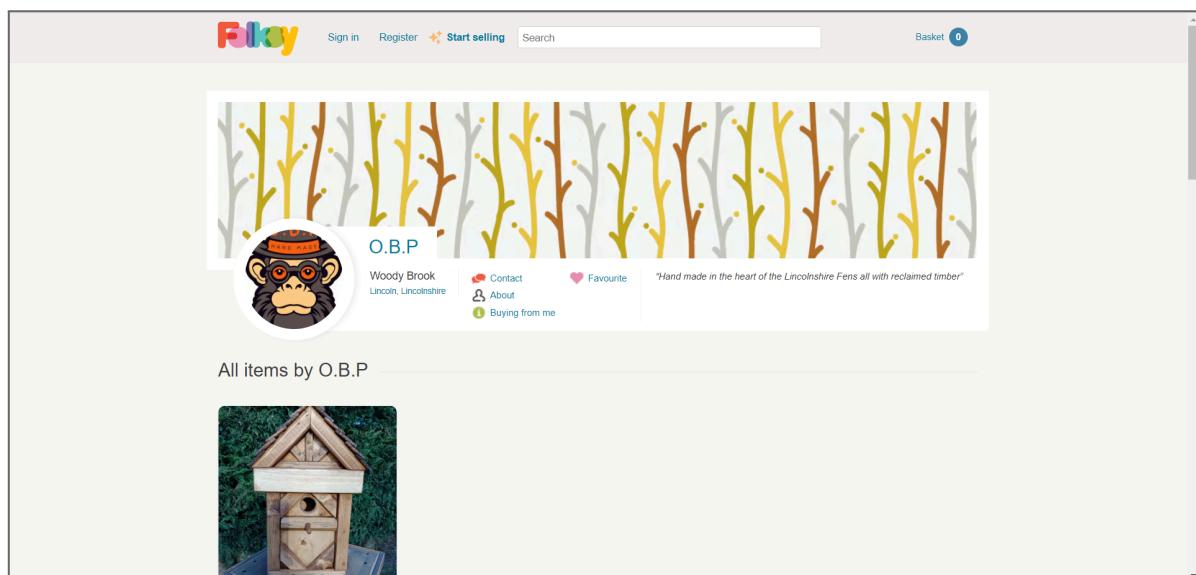
Figur. 1.8



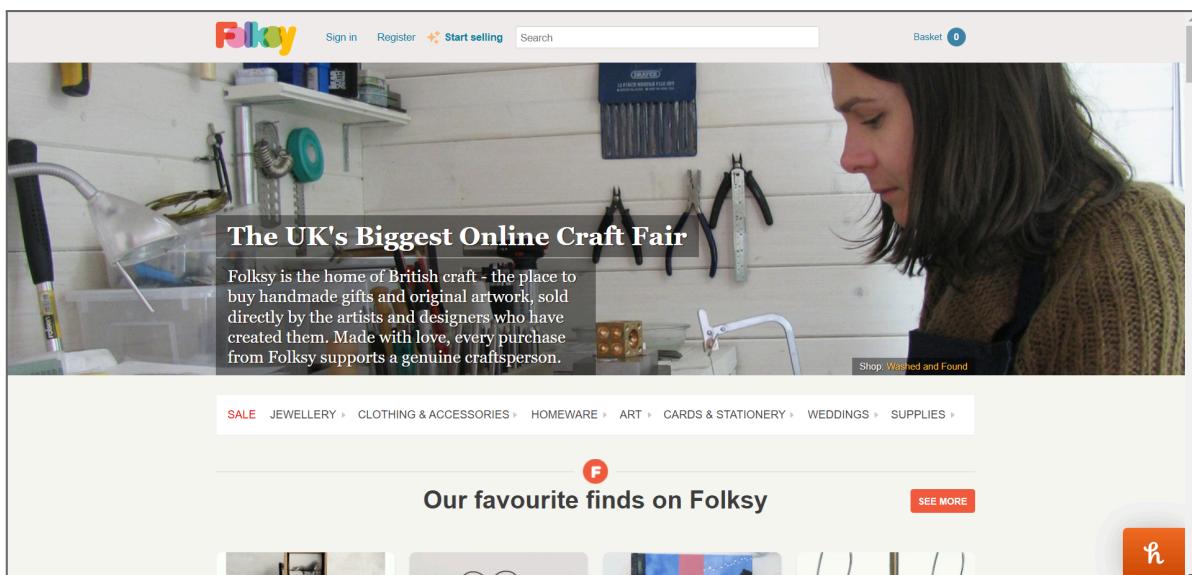
Figur. 1.9

1.2.2. Folksy

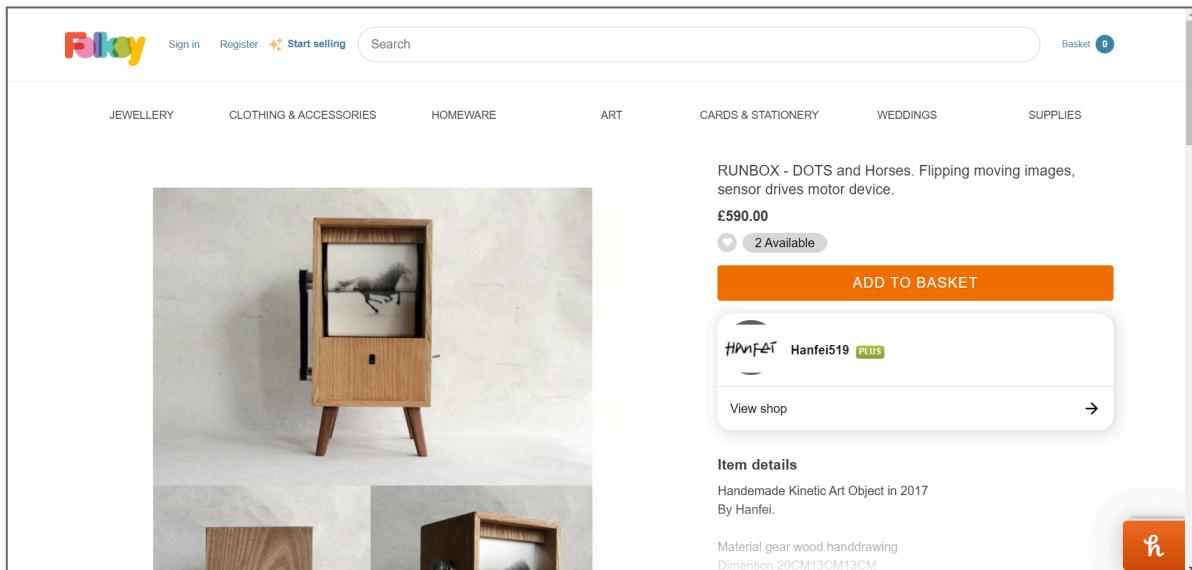
Folksy е платформа, базирана в Великобритания и специализирана в ръчно изработени продукти и дизайн. Функционалностите на приложението се доближават до моят проект по идеята за уникалност и ръчна изработка на продукта. Във Folksy продуктите са представени в магазин (фигура 1.10), докато в Wavary са представени в каталог. Платформата предоставя и начин за свързване с артиста, което се доближава до възможността за отправяне на запитване в моя проект Wavary. На фигури 1.10, 1.11 и 1.12 е показан потребителският интерфейс на уеб приложението.



Фиг. 1.10



Φυσ. 1.11



Φυσ. 1.12

Втора глава

Определяне на изисквания и технологии

2.1. Функционални изисквания към уеб базиран електронен магазин

Първото изискване към проекта е да бъде адаптивен за различните екрани на устройствата - телефони, таблети, лаптопи и компютри.

Като функционалност приложението има следните изисквания:

- Да се направят логин и регистър страници за потребители и механизъм за обозначаване като занаятчия. Проектът да поддържа възможността за посещаване като гост (потребител, който не е влезнал в профила си). Гостите имат възможността да разгледат платформата без право за интеракция нито с произведенията, нито със занаятчиите.
- Да се направи каталог с произведенията на всеки занаятчия. Представяйки продуктите си под формата на каталог, потребителите ще имат по-добра представа за стила и креативността на занаятчията.
- Възможност за последване на занаятчии и интеракция с произведенията им. Благодарение на последователите ще бъде нагледно за потребителите кой занаятчия е популярен. Харесването на изделия ще подпомогне достигане до по-голяма аудитория и съответно популяризиране на продукта и занаята.
- Изграждане на линия на времето, която показва етапите, през които е минал продуктът от идеята до неговия завършен вид. Представянето на изделието по този начин показва процесът на изработка и може да провокира интереса на клиента или на други занаятчии.

- Възможност за поръчки към занаятчията. Потребителят да може да отменя или редактира заявката си, в случай че е сгрешил, иска да допълни или да премахне информация от заявката. Занаятчията да може да одобрява или откаже поръчката в зависимост от своята заетост и интерес към даденото предложение.

2.2. Съображения за избор на програмни средства и развойната среда

2.2.1. Фронтенд технологии

JavaScript

Избрах JavaScript, понеже имам опит с него и знам, че е много популярен в света на уеб разработката. JavaScript е интерпретиран, обектно-ориентиран език, който е от съществено значение за фронтенд разработка, позволявайки динамично променяне на съдържанието на уеб страници и взаимодействие с потребителите.

React

Реших да насоча вниманието си към JavaScript библиотеката React поради нейната голяма популярност, неограничени възможности за създаване на модерни уеб приложения и опита, който съм натрупал. React улеснява изграждането на потребителски интерфейси, използвайки индивидуални парчета код наречени компоненти. Ключовото му предимство е че всеки компонент може да се преизползва, което прави целия процес на разработка по-бърз и по-приятен. Благодарение на виртуалния DOM, React обновява конкретния компонент при промяна в състоянието му вместо цялата страница, което оптимизира процеса.

Next.js

Реших да се фокусирам върху Next, като част от моят стремеж към разширяване на уменията ми в областта на уеб разработката. Next е мощен фреймуърк, използван за създаване на съвременни уеб приложения с акцент върху оптимизацията и лесната скалируемост. Той предоставя вградени възможности за рендериране от страна на сървъра и статично генериране, което подобрява ефективността на уеб приложението и оптимизира зареждането на страниците. Next предоставя по-голяма гъвкавост в маршрутизирането и разработването на динамични уеб приложения.

Tailwind CSS

Избрах Tailwind, който е CSS фреймуърк за стилизиране на страницата. Tailwind предоставя множество класове от базовите стилове като шрифтове, цветове, отстояния и размери. Този метод на стилово оформление позволява на програмистите бързо и лесно да създават уникални дизайни, като същевременно се осигурява гъвкавост и яснота в кода. Изборът на Tailwind ми позволява ефективно да изградя и персонализирам визуалния стил на уеб приложения чрез използването на тези помощни класове.

DaisyUI

Daisy е плъгин за Tailwind, който позволява по-бързата разработка на уеб приложения. Ключовата причина да избера Daisy е допълнителните класови имена и улесненото управление на светла и тъмна тема. Благодарение на допълнителните класови имена, времето за разработка намалява и се запазва последователност в цветовете и стилизирането на уеб приложението, което води до изчистения и привлекателен му дизайн.

2.2.2. Бекенд технологии

Firebase Authentication

За целите на проекта избрах Firebase Authentication поради неговите забележителни услуги. Платформата се отличава с това, че гарантира сигурността на потребителските данни чрез надеждни мерки за криптиране, което е важен аспект от управлението на потребителите. Простотата, лесно достъпните методи и удобният за използване API допринасят значително за моя избор. Допълнително Firebase Authentication поддържа различни методи за удостоверяване, включително електронна поща, социални мрежи, номера на телефони и други.

Firebase Firestore

Firebase Firestore е NoSQL база данни в облака, предоставена от платформата Firebase на Google. Тя е проектирана да съхранява и синхронизира данни в реално време между множество клиенти, което я прави отличен избор за изграждане на скалируеми и адаптивни уеб приложения. Firestore организира данните в колекции и документи, което позволява гъвкава и йерархична структура. Избрах Firestore заради лесната му употреба, тъй като опростява управлението на данните и предлага удобен за потребителя интерфейс. Освен това се стремя да се съредоточа върху фронтенда на уеб приложението, а възможностите на Firestore напълно съответстват на тази цел.

Firebase Storage

Избрах да използвам Firebase Storage за моя проект, понеже той безпроблемно се интегрира с Firebase Firestore и Firebase Authentication, обединявайки основните функционалности в една единна платформа. Този подход опростява цялостния процес на разработка. Firebase Storage служи като идеално решение за съхранение на различни файлове, включително

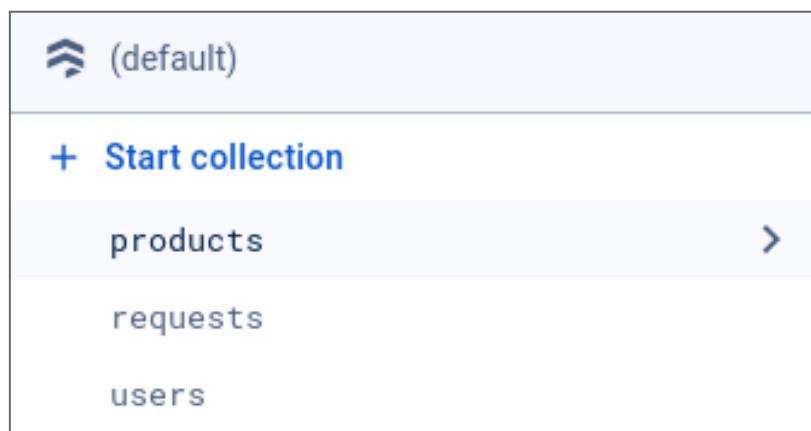
изображения на продукти и профилни снимки. Нейната скалируема и сигурна инфраструктура за съхранение в облака означава, че мога ефективно да управлявам и извличам тези файлове при необходимост, което подобрява цялостната ефективност на моето уеб приложение.

2.2.3. Интегрирана среда за разработка

Поради предишния ми опит с продуктите на JetBrains - IntelliJ, CLion, PyCharm и Android Studio реших да разработя проекта си в WebStorm. JetBrains са известни с качествените си интегрирани среди за разработка. Характерно за техните продукти е подобният графичен потребителски интерфейс, което допринася за избора ми. Благодарение на предишният ми опит с продуктите на JetBrains и подобният графичен потребителски интерфейс, изучаването и използването на WebStorm е значително улеснен процес.

2.3. Структура на база данни и облачно хранилище

Базата данни, която използвам е нерелационна и е разделена на колекции и документи. В документите се съхранява информация под формата на ключ и стойност, а те се съдържат в колекции. В моя проект има 3 основни колекции - “products”, “requests” и “users” (фигура 2.1).



Фиг. 2.1

“products” колекцията съдържа документи, именувани с уникален идентификационен номер. Всеки документ в колекцията съдържа следните полета (Фигура 2.2):

- createdDate - низ, съдържащ дата и час кога продукта е добавен в платформата.
- description - низ, съдържащ описание на продукта.
- displayImageURL - низ, съдържащ линк към изображение на продукта, съхранявано във Firebase Storage.
- likes - списък с идентификационните номера на потребителите, които са харесали продукта.
- owner - низ, съдържащ идентификационен номер на занаятчията, добавил продукта в платформата.
- price - число с плаваща запетая, съдържащо цена на продукта.
- tags - списък от тагове на продукта.
- timeline - линия на времето, показваща процеса на разработка на продукта. Това е списък с обекти (всеки обект описва един времеви диапазон от разработката на продукта), съдържащи следните полета (Фигура 2.3):
 - date - ориентировъчна дата на начало на времевия диапазон.
 - description - описание на времевия диапазон.
 - heading - заглавие на времевия диапазон.
- title - низ, съдържащ наименование на продукта.

products	J30qqkA4FSClvnIqdDos
+ Add document	+ Start collection
5PTCRmSirD96ymmo8aHQ	
6mIHNOjTaIPyphZtPSUR	
FZWAc35lxXiRH0Lrs1C	
GF5wSLOwtRRzP4amgrKm	
J30qqkA4FSClvnIqdDos	+ Add field
igqW04dB1RnPXqAM3qxG	createdDate: February 9, 2024 at 9:33:34 AM UTC+2
jbKAD5kPvW6RMa0HUlMp	description: "Enhance your productivity and elevate your workspace aesthetics with this sleek desk organizer."
setwUMoJ6tNuLw8Ttxj8	displayImageURL: "https://firebasestorage.googleapis.com/v0/b/diploma-project-2024-5f3d1.appspot.com/.../product-image.jpg"
tVvqMHYmcbs7NsJmpFhr	likes: 0
wrmM96Nk8nljQEnjr8hd	owner: "fbpcMAdwYjeZKJ1OFCFv95LizA13"
	price: "29.99"
	tags:
	timeline:
	title: "Desk organizer"

Φιγ. 2.2

▼ timeline	
▼ 0	
date: "2023-12-04"	
description: "Our skilled artisans started conceptualizing the design, ensuring every detail was carefully planned."	
heading: "Design Concept"	(string)
▼ 1	
date: "2023-12-13"	
description: "The intricate craftsmanship began, with attention to detail in every cut and finish."	
heading: "Craftsmanship"	
▼ 2	
date: "2023-12-28"	
description: "After weeks of meticulous work, the Elegant Sapphire Necklace is finally complete!"	
heading: "Completion and Release"	

Φιγ. 2.3

“requests” колекцията съдържа документи, именувани с уникален идентификационен номер. Всеки документ в колекцията съдържа следните полета (Фигура 2.4):

- craftsman - низ, съдържащ идентификационния номер на занаятчия, към когото е направена поръчката.
- description - низ, съдържащ описание на поръчката.
- status - низ, съдържащ статус на поръчката. Той може да има стойностите “waiting” за поръчка, която се разглежда от занаятчията, “canceled” ако потребителят отмени заявката, “accepted” или “denied” ако занаятчията съответно одобри или откаже поръчката.
- title - низ, съдържащ заглавието на поръчката.
- user - низ, съдържащ идентификационния номер на потребителя, който е направил запитването към занаятчия.

requests	4r5HFzAuVQ2i4I7ZFVUX
+ Add document	+ Start collection
4r5HFzAuVQ2i4I7ZFVUX	+ Add field
KnBaRUNIdFZCzAa2HGXL	craftsman: "fbpcMAdwYjeZKJ10FCFv95LlzA13"
loHevkYV7QL6HjphXbCb	description: "Hi, can you make me a fancy wooden chair?"
	status: "waiting"
	title: "Wooden chair"
	user: "hC7hhHzZxeXeTFSalD2NX6qcWhz1"

Фиг. 2.4

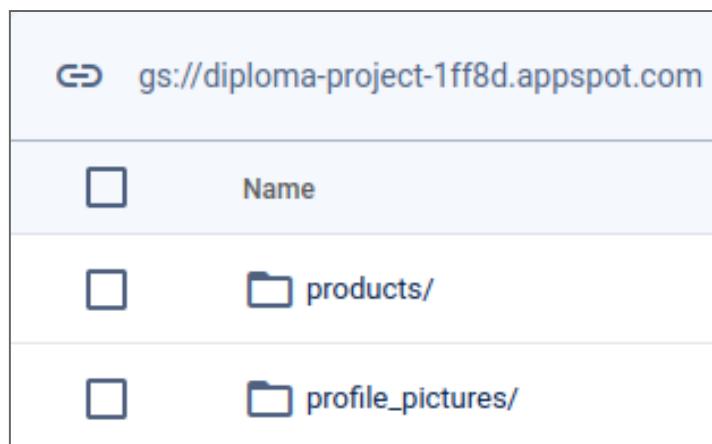
“users” колекцията съдържа документи като при всеки от тях присъстват следните полета (Фигура 2.5):

- craft - низ, съдържащ информация дали потребителят е занаятчия. При нормалният потребител е със стойност “null”.
- email - низ, съдържащ имейла на потребителя.
- followers - списък с идентификационните номера на последователите на занаятчия. При нормалния потребител този списък е празен.
- name - низ, съдържащ пълното име на потребителя.
- photoURL - низ, съдържащ линк към профилната снимка, съхранявано във Firebase Storage.
- uid - низ, съдържащ генерираното уникален идентификационен номер на потребителя от Firebase Authentication.

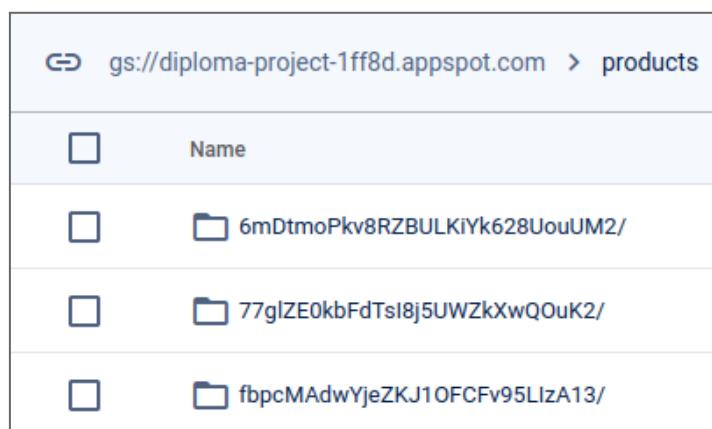
users	70dDg3C1K2v0aj6FHhHG
+ Add document	+ Start collection
2qArXmJVauACyXGJzoNn	+ Add field
4kSrS1vguiCTJvIBb891	craft: "craftsman"
70dDg3C1K2v0aj6FHhHG	email: "nikpetrov1406@gmail.com"
JntQu6MjGM2G3PennDQ2	followers
T42FeaNruQoGVvCJR4yM	0 "77glZE0kbFdTsl8j5UWZkXwQOuK2"
lsKvPlWYLOXS5aKh872b	1 "fbpcMAdwYjeZKJ10FCFv95LizA13"
xSpcHdgj92Y8mmgcVEJU	name: "Nikola Petrov"
	photoURL: "https://firebasestorage.googleapis.com/v0/b/diploma-project-1ff8d.appspot.com/.../image.jpg"
	uid: "6mDtmoPk8RZBULKiYk628UouUM2"

Фиг. 2.5

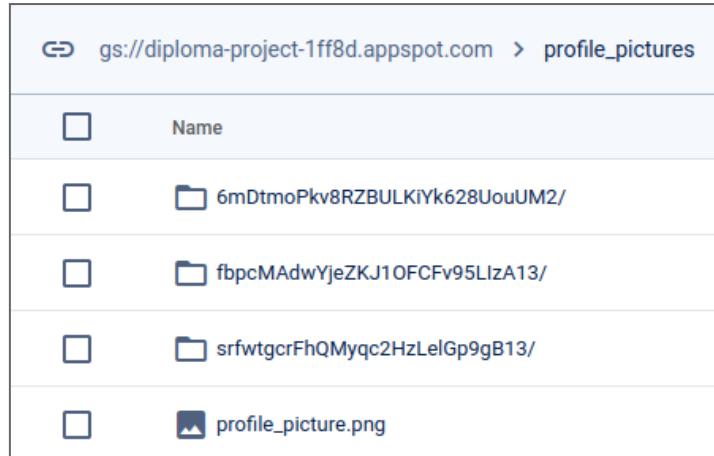
Хранилището, което използвам в моя проект, предоставя облачно съхранение за различни видове файлове. За целите на проекта се използва бъкет по подразбиране от Firebase Storage. В него се съдържат всички директории и файлове нужни за правилното функциониране на уеб приложението. Основните директории са “products” и “profile_pictures” (Фигура 2.6). Във всяка от тях се съдържат поддиректории с идентификационния номер на потребителят, на когото принадлежи съответната профилна снимка или продукт (Фигура 2.7). В директорията “profile_pictures” се съдържа и файла “profile_picture.png”, който е профилната снимка по подразбиране за всеки потребител, а във всяка поддиректория се съдържа профилната снимка на потребителя (Фигура 2.8).



Фиг. 2.6

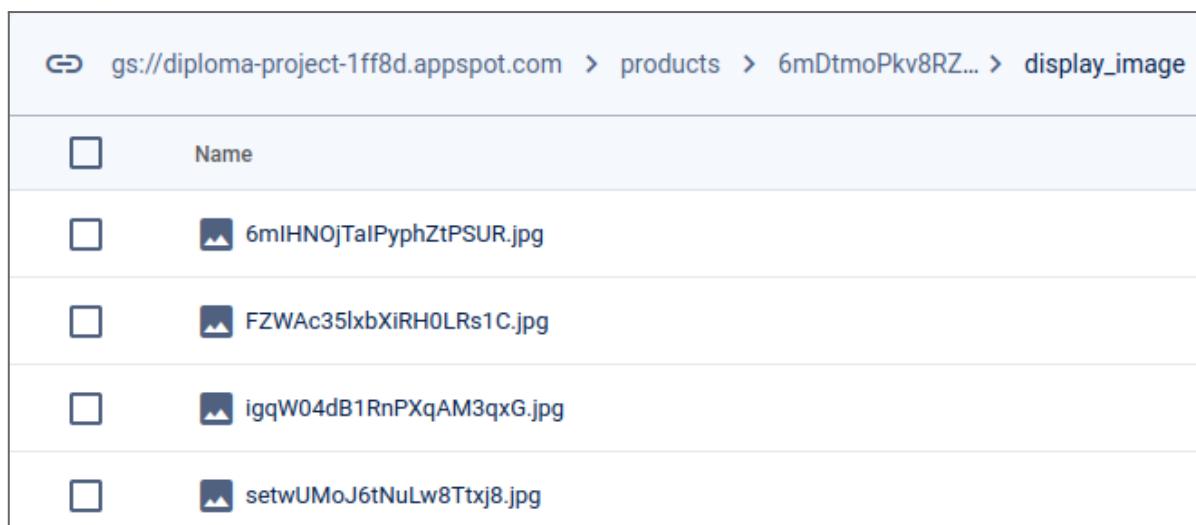


Фиг. 2.7



Фиг. 2.8

В директорията “products” във всяка поддиректория с идентификационния номер на потребителят, на когото принадлежи съответния продукт има директория именувана “display_image”, която съдържа всички изображения на продуктите на дадения потребител (Фигура 2.9). Файловете са именувани с идентификационния номер на продукта.



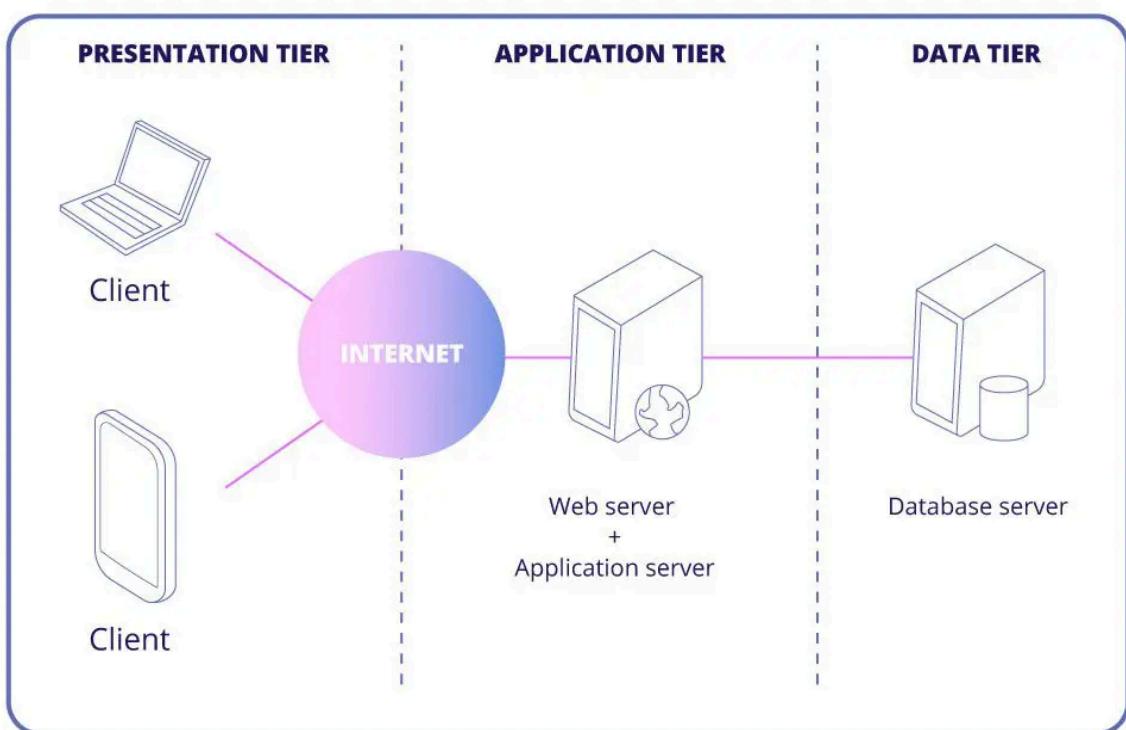
Фиг. 2.9

Трета глава

Програмна реализация

3.1. Архитектура на разработваното уеб приложение

Уеб приложението се базира на модела клиент-сървър (три слоева архитектура) като слоевете са - презентационен, междинен и слой за данни (Фигура 3.1). Презентационния слой е отговорен за графичния потребителски интерфейс и интеракциите на потребителя. Междинния слой обработва събраната информация от презентационния слой като може да добавя, изтрива или променя информацията в слоя за данни. Слоя за данни е отговорен за съхранението на данните обработени от приложението.

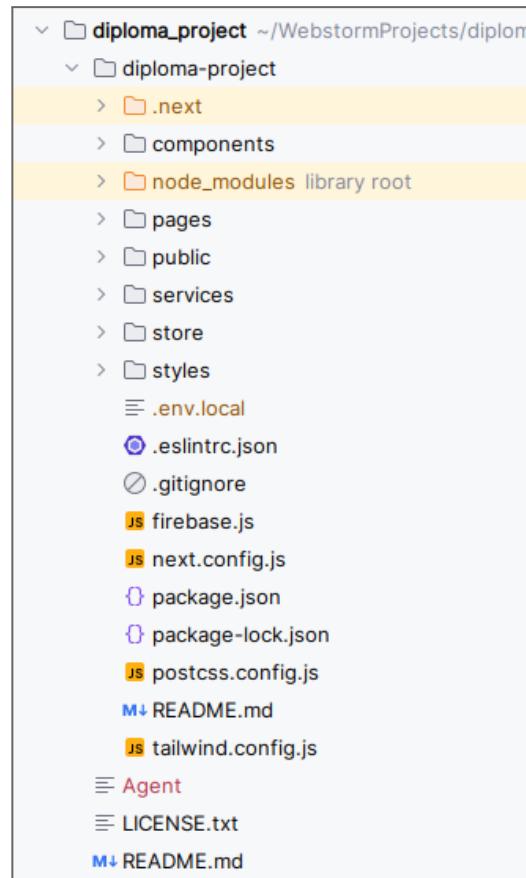


Фиг. 3.1

3.2. Структура на проекта в WebStorm

В проекта има няколко основни директории и файлове (Фигура 3.2):

- components - директория, съдържаща React компонентите, използвани в проекта.
- pages - директория, съдържаща страници на уеб приложението.
- public - директория, съдържаща статични файлове.
- services - директория, съдържаща API методи за комуникация с Firebase.
- store - директория, съдържаща zustand хранилище за глобалното състояние.
- firebase.js - файл, съдържащ конфигурацията на Firebase.
- package.json - файл, съдържащ основна информация за използваните допълнителни софтуерни инструменти.
- tailwind.config.js - файл, съдържащ конфигурацията на Tailwind и daisyUI.



Фиг. 3.2

- package-lock.json - файл, съдържащ подробна информация за използваните допълнителни софтуерни инструменти.
- tailwind.config.js - файл, съдържащ конфигурацията на Tailwind и daisyUI.

3.3. Състояние на уеб приложението

Състоянието на уеб приложението бива два вида - глобално и локално. За целите на проекта използвах `zustand` за глобално състояние, който ми позволява да съхранявам информация за състоянието в хранилище, което може да бъде достъпено от всеки компонент. Това спомага с прегледността и намалява обема на сурс кода. Само едно хранилище се използва в проекта - “`userStorage.js`” (Фиг. 3.3). То съдържа информация за потребителя и вида на экрана (дали е тъчскрийн или не). Информацията, която се съхранява за потребителя се базира на класа “`User`”.

```
JavaScript
import { create } from "zustand";
import { User } from "../services/user";

export const useUserStore = create((set) => ({
    user: new User(),
    setUser: (newUser) => set(() => ({ user: newUser })),
    isTouchEnabled: null,
    setIsTouchEnabled: (newValue) => set(() => ({ isTouchEnabled: newValue })),
}));
```

Фиг. 3.3

3.4. Автентикация на уеб приложението

Уеб приложението използва `Firebase Authentication` за управление на идентичността на потребителите. Във файла “`_app.jsx`”, за да се следи потребителя дали е удостоверил автентичността си използва метода “`onAuthStateChanged()`”, който следи за състоянието на автентикация на потребителя. Когато потребителя влезе в профила си информацията му се взима от базата данни и се запазва в глобалния `state` на уеб приложението заедно с информацията за экрана (Фигура 3.4). В допълнение токена за достъп се записва в сесийното хранилището на браузъра, за да позволи

по-бързата проверка дали потребителя с неговото състояние на автентикация има право да достъпва различните страници чрез маршрутите на уеб приложението. Когато потребителя излезе от профила си метода “onAuthStateChanged()” засича промяната и изтрива информацията от глобалното състояние и от сесийното хранилището.

JavaScript

```
const MyApp = ({ Component, pageProps }) => {
  const { user, setUser, setIsTouchEnabled } = useUserStore((state) => ({user: state.user, setUser: state.setUser, setIsTouchEnabled: state.setIsTouchEnabled}));

  useEffect(() => {
    setIsTouchEnabled(window.matchMedia("(pointer: coarse)").matches);
    let unsubscribeSnapshot = null
    const unsubscribeAuth = getAuth(auth).onAuthStateChanged((currentUser) => {
      if (currentUser) {
        // Store the user data in the global state
        getUserByEmail(currentUser.email).then((data) => {
          setUser({
            name: data.name,
            email: data.email,
            photoURL: data.photoURL,
            uid: data.uid,
            interests: user.interests,
            requests: user.requests,
            craft: data.craft,
            orders: user.orders,
            catalog: user.catalog,
            followers: data.followers,
          });
          currentUser.getIdToken().then((token) => {
            sessionStorage.setItem('accessToken', token);
          });
        });
      }
    });
  });
}
```

Фиг. 3.4

Конфигурирането на Firebase Authentication се извършва във файла “firebase.js” (Фигура 3.5). Първо се инициализира конфигурацията на Firebase, която се състои от поетата “apiKey”, “authDomain”, “projectId”, “storageBucket”, “messagingSenderId”, “appId” и “measurementId”, които са необходими за инициализиране на Firebase приложението. Информацията за конфигурация на Firebase приложението се пази в environment променливи, за да не се следят от git и да не се качват в GitHub. За да се компилира приложението тези променливи на средата за изпълнение се вкарват в средата за внедряване. След като се инициализира приложението се взимат инстанции на услугите, които използваме в проекта - Firebase Authentication, Firebase Firestore и Firebase Storage.

JavaScript

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
import { getStorage } from "firebase/storage";

const firebaseConfig = {
  apiKey: process.env.NEXT_PUBLIC_FIREBASE_API_KEY,
  authDomain: process.env.NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
  storageBucket: process.env.NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.NEXT_PUBLIC_FIREBASE_APP_ID,
  measurementId: process.env.NEXT_PUBLIC_FIREBASE_MEASUREMENT_ID
};

const app = await initializeApp(firebaseConfig);

export const auth = await getAuth(app);
export const firestore = await getFirestore(app);

export const storage = await getStorage(app);

export default app;
```

Фиг. 3.5

3.5. Фронтенд на уеб приложение



Фиг. 3.6

Фронтенд на уеб приложението е изградено чрез страници и компоненти. Маршрутизирането между различните страници е базирано на структурата на поддиректориите и файловете в директорията "pages" (Фигура 3.6). Така на база на пътищата до файловете в директорията "pages", Next генерира маршрути, например ако имаме файла "profile.jsx" с път "pages/profile.jsx" то маршрута ще бъде "/profile". Изключение е "index.jsx" файла, който Next интерпретира като "/". Когато не знаем предварително точните имена на сегментите и искаме да създадем маршрути от динамични данни, може да използваме

динамични сегменти, които се попълват по време на заявка или се визуализират предварително по време на билд процеса. Динамичен сегмент може да се създаде, като името на файла или директория се постави в квадратни скоби: [segmentName]. Например, [id] или [slug]. В моя проект динамични сегменти са файловете "[id].jsx" в директориите "catalog" и "product", защото не мога предварително да определя всички възможни маршрути за продуктите и каталогите на занаятчиите.

index.jsx

Началната страница се намира във файла с път “pages/index.jsx” и маршрута му е съответно “/”. Чрез server-side rendering (използвайки метода “getServerSideProps()”) информацията за новите продукти (променливата “newProducts”), тенденция на продукти (променливата “trendingProducts”) и разнообразие на продукти (променливата “discoverProducts”) се извлича и обработва от сървъра и се подава на клиента в json формат (Фигура 3.7). В React компонента “Home”, информацията се преобразува от json формат в масиви от React компонентите - “AvatarIcon”, “TrendingCard” и “ProductCard” (Фигура 3.8). Масивите с React компонентите се използват за изобразяване на продуктите на страницата.

```
JavaScript
export async function getServerSideProps() {
    return Promise.all([getNewProducts(), getTrendingProducts(),
    getDiscoverProducts()]).then(([newProducts, trendingProducts,
    discoverProducts]) => {

        const top5TrendingProducts = trendingProducts.sort((a, b) => b.likes.length -
        a.likes.length).slice(0, 5);

        return {
            props: {
                newCrafts: newProducts,
                trending: top5TrendingProducts,
                discover: discoverProducts,
            }
        }
    });
}
```

Фиг. 3.7

JavaScript

```
const Home = ({newCrafts, trending, discover}) => {
  const [newProducts, setNewProducts] = useState(null);
  const [trendingProducts, setTrendingProducts] = useState(null);
  const [discoverProducts, setDiscoverProducts] = useState(null);
  const [toggleDrawerContent, setToggleDrawerContent] = useState(true);

  useEffect(() => {
    let products = [];

    newCrafts.map((item, index) => {
      products.push(<AvatarIcon key={index} img={item.photoURL}
username={item.name} catalogHref={`/catalog/${item.userID}`}
productHref={`/product/${item.productID}`}/>);
    });

    setNewProducts(products);
  }, [newCrafts]);

  useEffect(() => {
    let products = [ ];

    trending.map((product, index) => {
      products.push(<TrendingCard key={index} product={product}/>);
    });

    setTrendingProducts(products);
  }, [trending]);

  useEffect(() => {
    let products = [ ];

    discover.map((product, index) => {
      products.push(<ProductCard key={index} product={product}
productId={product.id} inCatalog={false}/>);
    });

    setDiscoverProducts(products);
  }, [discover]);
}
```

Фиг. 3.8

signin.jsx

Страницата за вход се намира във файла с път “pages/signin.jsx” и маршрута му е съответно “/signin”. В “useEffect” се проверява дали потребителя е автентициран и ако е се пренасочва към началната страница. Когато се предаде логин формата се извиква метода “onSubmit”, където ако успешно се влезе в профила, потребителя се пренасочва към началната страница (Фигура 3.9).

```
JavaScript
const SignIn = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [fetchingToken, setFetchingToken] = useState(true);
  const router = useRouter();

  const onSubmit = (event) => {
    event.preventDefault();

    loginWithEmailAndPassword(email, password)
      .then(() => {
        // Login is successful
        void router.push('/');
      })
      .catch((error) => {
        // Login failed, set the error message
        errorToast(error.message);
      });
  };

  useEffect(() => {
    // redirect if user is authenticated
    let accessToken = sessionStorage.getItem("accessToken");
    if(accessToken){
      void router.replace("/");
    } else {
      setFetchingToken(false);
    }
  }, []);
}
```

Фиг. 3.9

signup.jsx

Страницата за регистрация се намира във файла с път “pages/signup.jsx” и маршрута му е съответно “/signup”. В “useEffect” се проверява дали потребителя е автентициран и ако е се пренасочва към началната страница. Когато се предаде формата за регистрация се прави проверка дали паролите, въведени от потребителя, са еднакви и ако са се извиква метода “onSubmit”, където ако успешно се регистрира, потребителя се пренасочва към началната страница (Фигура 3.10). В допълнение при въвеждане на пълното име се прави проверка дали е до 20 символа.

```
JavaScript
const onSubmit = (event) => {
    event.preventDefault();

    if (passwordOne === passwordTwo) {
        registerWithEmailAndPassword(fullName, email, passwordOne)
            .then((user) => {
                addUser(fullName, email, user.uid)
                    .then(() => {
                        void router.push("/");
                    })
                    .catch((error) => {
                        // Failed to add user, set the error message
                        errorToast(error.message);
                    });
            })
            .catch((error) => {
                // Register failed, set the error message
                errorToast(error.message);
            });
    } else {
        errorToast("Passwords do not match!");
    }
};
```

Фиг. 3.10

profile.jsx

Профилната страницата се намира във файла с път “pages/profile.jsx” и маршрута му е съответно “/profile”. В трите “useEffect” се извлича информацията за интересите, запитванията и поръчките (Фигура 3.12). Данните се преобразуват от обекти в React компоненти за по-лесната визуализация в компонента “ItemScroll”. След преобразуването на информацията се записва в променливите на състояние - “interests”, “requests”, “orders”, за да се следи за промени. Ако има промяна в полетата на сегашния потребител - “currentUser.interests”, “currentUser.requests” или “currentUser.orders”, данните се извличат наново. Поръчки имат само занаятчиите, а при нормалните потребители полето с поръчки не се показва. В допълнение на тази страница потребителите имат възможност да се обозначат като занаятчици чрез натискане на бутона “Become craftsman” (Фигура 3.11). При натискане на бутона се извиква функцията “BecomeCraftsman()”, която чрез помощни функции цели да обнови стойността на полето “craft” в базата данни.

JavaScript

```
const BecomeCraftsman = () => {
  UpdateProfileToCraftsman("craftsman")
    .then(() => {
      successToast("Successfully became a craftsman!");
    })
    .catch(() => {
      errorToast("Error: Couldn't become a craftsman! Refresh the page and
try again!");
    });
};
```

Фиг. 3.11

```

JavaScript
useEffect(() => {
  if(currentUser.uid !== null) {
    getInterests(currentUser).then((products) => {
      let interestsList = [];

      products.map((product) => {
        interestsList.push(<ProductCard key={product.id} product={product}
          productId={product.id} inCatalog={false}/>);
      });

      setInterests(interestsList);
    });
  }
}, [currentUser.uid, currentUser.interests]);

useEffect(() => {
  if(currentUser.uid !== null) {
    getRequests(currentUser).then((products) => {
      let requestsList = [];

      products.map((product) => {
        requestsList.push(<RequestCard key={product.id} request={product}/>);
      });

      setRequests(requestsList);
    });
  }
}, [currentUser.uid, currentUser.requests]);

useEffect(() => {
  if(currentUser.uid !== null && currentUser.craft !== null) {
    getOrders(currentUser).then((products) => {
      let ordersList = [];

      products.map((product) => {
        ordersList.push(<RequestCard key={product.id} request={product}/>);
      });

      setOrders(ordersList);
    });
  }
}, [currentUser.uid, currentUser.orders]);

```

Фиг. 3.12

catalog/[id].jsx

Страницата за каталога на занаятчията се намира във файла с път “pages/catalog/[id].jsx” и маршрута му е съответно “/catalog/[id]”. Всеки занаятчия може да бъде последван чрез бутона “Follow” и отпоследван от бутона “Unfollow”. И двата бутона при натискане извикват функцията “onFollow()”, чиято цел е да обнови статуса на последовател на сегашния потребител (Фигура 3.13). Това се случва като добави или махне идентификационния му номер от масива с последователи на занаятчията.

```
JavaScript
const onFollow = () => {
  let followers = [...craftsmanFollowers];
  const isFollower = craftsmanFollowers.includes(currentUser.uid);
  if(isFollower) {
    followers = followers.filter((uid) => uid !== currentUser.uid);
  } else {
    followers.push(currentUser.uid);
  }

  UpdateFollowers(craftsman.uid, followers)
    .then(() => {
      setCraftsmanFollowers(followers);
      successToast(`Successfully ${isFollower ? "unfollowed" : "followed"} ${craftsman.name}!`);
    })
    .catch(() => {
      errorToast(`Error: Couldn't ${isFollower ? "unfollow" : "follow"} ${craftsman.name}! Refresh the page and try again!`);
    });
}
```

Фиг. 3.13

При искане на страницата от клиента, сървъра извлича данните за каталога чрез заложения идентификационен номер във въведения URL. Това се случва във функцията “getServerSideProps()” (Фигура 3.14). Извлечените данни се поддават на компонента “Catalog” в json формат. Понеже каталогът е масив от низове, съдържащи идентификационните номера на продуктите в “useEffect” се извличат данните за всяко едно изделие.

```

JavaScript
export async function getServerSideProps({ params }) {
  return getUserById(params.id).then((userData) => {
    // if user is not a craftsman return notFound - 404 page
    if(userData.craft !== null) {
      const jsonUser = {
        name: userData.name,
        email: userData.email,
        photoURL: userData.photoURL,
        uid: userData.uid,
        craft: userData.craft,
        followers: userData.followers,
        interests: [],
        requests: [],
        orders: [],
        catalog: []
      }
      return {
        props: {
          craftsman: jsonUser,
        }
      }
    } else {
      return {
        notFound: true,
      }
    }
  });
}

```

Фиг. 3.14

product/[id].jsx

Страницата за продукта се намира във файла с път “pages/product/[id].jsx” и маршрута му е съответно “/product/[id]”. По същия начин като при страницата за каталога, информацията за продукта се извлича от страна на сървъра чрез функцията “getServerSideProps()”. Ако продукта не е намерен се връща “notFound”, при което Next показва страницата “404.jsx”. Възможно е да се харесват продукти като това се случва при извикването на функцията “onLike()” (Фигура 3.15).

```

JavaScript
const onLike = () => {
  let interests;

  if(isLiked){
    product.likes = product.likes.filter(uid => uid !== currentUser.uid);
    interests = currentUser.interests.filter(id => id !== product.id);
  } else {
    product.likes.push(currentUser.uid);
    interests = [...currentUser.interests, product.id];
  }

  updateLikesOfProduct(product.id, product.likes)
    .then(() => {
      setCurrentUser({
        name: currentUser.name,
        email: currentUser.email,
        photoURL: currentUser.photoURL,
        uid: currentUser.uid,
        interests: interests,
        requests: currentUser.requests,
        craft: currentUser.craft,
        orders: currentUser.orders,
        catalog: currentUser.catalog,
        followers: currentUser.followers,
      });
      setIsLiked(() => !isLiked);
    })
    .catch((error) => {
      errorToast(error.message);
    });
  );
};

}

```

Фиг. 3.15

404.jsx

Страницата 404 служи за уведомяване на потребителя че страницата, която търси не съществува. В нея единствената функционалност е да се върнеш обратно на миналата страница, използвайки “router” на Next.

500.jsx

Страницата 500 служи за уведомяване на потребителя при случила се грешка на сървъра.

NewProduct.jsx

Компонент представляващ форма за добавяне на нов продукт. Съдържа полета за въвеждане на заглавие, описание, цена, тагове, изображение и линия на времето на продукта. Функцията “onImageChange()” проверява дали избраното изображение е “png” или “jpeg” и го записва в променливата за състояние “image”. Валидирането на входните данни на потребителя става още при тяхното записване в променливите за състояние. Когато потребителя реши да добави продукта се извиква функцията “onSubmit”, която чрез помощни функции, добавя продукта в базата данни (Фигура 3.16).

```
JavaScript
const onSubmit = (event) => {
    event.preventDefault();
    if(!submitting) {
        setSubmitting(true);

        addProduct(title, description, image, Math.floor(parseFloat(productPrice)
* 100) / 100, productTimeline, productTags)
            .then(() => {
                setTitle("");
                setDescription("");
                setImage(null);
                setProductPrice("");
                setProductTimeline([]);
                setProductTags([]);

                successToast("Successfully added new product!");
                setSubmitting(false);

                // reset the form
                event.target.reset();
            })
            .catch((errorMessage) => {
                errorToast(errorMessage);
            });
    }
}
```

Фиг. 3.16

about.jsx

Страницата служи за информиране на потребителите относно уеб приложението. Описва се какво представлява платформата и основните услуги, които предлага.

NewRequest.jsx

Компонент представляващ форма за правене на поръчка към занаятчия. Съдържа полета за въвеждане на заглавие и описание на запитването. Валидирането на входните данни на потребителя става още при тяхното записване в променливите за състояние (Фигура 3.17). Когато потребителя реши да направи поръчката си се извиква функцията “onSubmit”, която чрез помощни функции, добавя запитването в базата данни (Фигура 3.18).

```
JavaScript
onChange={(e) => {
    const requestTitle = e.target.value;

    if(requestTitle.length > 20) {
        if(!toast.isActive(toastId.current)) {
            toastId.current = errorToast("Error: Title can't be more than
20 characters!");
        }
    } else {
        setTitle(requestTitle);
    }
}}
```

Фиг. 3.17

```

JavaScript
const onSubmit = (event) => {
    event.preventDefault();

    setSubmitting(true);

    addRequest(title, description, currentUser, craftsman.uid, "waiting")
        .then(() => {
            setTitle("");
            setDescription("");

            successToast("Successfully added new request!");
            setSubmitting(false);

            // reset the form
            event.target.reset();
        })
        .catch((errorMessage) => {
            errorToast(errorMessage);
        });
}

```

Фиг. 3.18

3.6. Бекенд на уеб приложението

За бекенд на уеб приложението ползвам Firebase, които ми предоставя чрез своя API много лесни за използване функции за достъп до базата данни, автентификацията и хранилището за изображения. Функциите от API съм извадил в мои функции в директорията “services”. Така ако се наложи миграция на базата данни, автентификацията или хранилището ще се променят само моите функции намиращи се в директорията “services” като останалата част от кода ще остане същия. В допълнение, когато логиката за достъп до бекенда е изнесена в отделни файлове кода изглежда по-четлив и по-изчистен.

loginWithEmailAndPassword.js

Съдържа функция, в която при подадени имейл и парола се влиза в профила на дадения потребител.

logOut.js

Съдържа функция, в която се излиза от профила на сегашния потребител.

registerWithEmailAndPassword.js

Съдържа функция, в която при подадени пълно име, имейл и парола се регистрира потребителя и автоматично влиза в новосъздадения си профил.

user.js

Съдържа класът “User” и различни функции свързани с колекцията “users” в базата данни:

- addUser() - създава документ за потребителя след като се регистрира.
- getUserByQuery() - връща информация за потребителя по подадената заявка.
- getUserByEmail() - подава заявката на getUserByQuery() като цели да вземе информацията за потребител с подадения имейл.
- getUserById() - подава заявката на getUserByQuery() като цели да вземе информацията за потребител с подадения идентификационен номер.
- getUserDoc() - при подаден идентификационен номер на потребител връща референция към документа му в базата данни.
- UpdateProfilePictureDB() - обновява URL на профилната снимка в базата данни.
- UpdateProfilePicture() - обновява профилната снимка в хранилището за изображения, след което извиква UpdateProfilePictureDB(), за да обнови URL на изображението в базата данни.
- UpdateProfileToCraftsman() - обновява статуса от потребител на занаятчия.
- UpdateFollowers() - обновява последователите на занаятчията.

product.js

Съдържа класът “Product” и различни функции свързани с колекцията “products” в базата данни:

- updateDisplayImageURLOfProduct() - обновява изображението на продукт по идентификационния му номер.
- addProduct() - създава документ за продукта в базата данни и добавя изображението на продукта в хранилището.
- getProduct() - при подаден идентификационен номер на продукта връща данните му.
- updateLikesOfProduct() - обновява харесванията на продукта при подаден идентификационен номер и нов масив с харесвания.
- getProductsByListType() - извлича продукти от базата данни в масив на база на даден критерии.
- getCatalog() - извиква getProductsByListType() с критерии да търси по owner.
- getInterests() - извиква getProductsByListType() с критерии да търси по това дали в харесванията се съдържа идентификационния номер на потребител.
- getNewProducts() - извлича новите продукти на занаятчиите. Продукт е означен като нов, когато е добавен преди по-малко от 24 часа.
- getTrendingProducts() - извлича популярните продукти за тази седмица.
- getDiscoverProducts() - извлича други продукти, които са добавени преди поне един ден и не по-дълго от един месец.

requests.js

Съдържа класът “Request” и различни функции свързани с колекцията “requests” в базата данни:

- addRequest() - създава документ за поръчката.
- getRequest() - извлича информация за запитването при подаден идентификационен номер на поръчката.
- getRequestsByListType() - извлича запитвания от базата данни в масив на база на даден критерии.
- getRequests() - извиква getRequestsByListType() с критерии да търси в потребителите, които са направили запитвания. Търси се по идентификационен номер на потребител.
- getOrders() - извиква getRequestsByListType() с критерии да търси в занаятчиите, които са получили поръчки. Търси се по идентификационен номер на потребител.
- updateRequest() - обновява заглавието и описанието на запитване.
- updateStatus() - обновява статуса на поръчката.
- deleteRequest() - по подаден идентификационен номер на поръчка, проверява дали документа, който ѝ съответства съществува и ако да го изтрива. След изтриване на документа се проверява сегашния потребител дали е бил клиент или занаятчия. Ако е бил клиент се обновява масива “requests” в глобалното състояние на сегашния потребител. Ако е бил занаятчия се обновява масива “orders” в глобалното състояние на сегашния потребител.

Четвърта глава

Ръководство на потребителя

4.1. Инсталация на уеб приложението

Уеб приложението може да бъде разгледано по два начина - чрез инсталация и изпълняване на кода или чрез достъпване на <https://diploma-project.vercel.app/>. Инсталацията става като се клонира следното репо https://github.com/ItsRizee/diploma_project. Препоръчително е да се използва версия v18.16.1 на Node. След клонирането трябва да се създаде файл с име “.env.local” в главната директория на проекта както е показано във фигура 3.2 със съдържанието показано на фигура 4.1. Следващата стъпка е да се навигираме до главната директория на проекта “diploma-project” (Фигура 4.2) в конзолата и да се изпълни команда “npm install”, за да изтегли всички допълнителни софтуерни инструменти, използвани за разработката на уеб приложението. Последната стъпка е да бийлднете и изпълните кода чрез командите “npx next build” и “npx next start” и да достъпим уеб приложението на <http://localhost:3000>.

```
Unset
NEXT_PUBLIC_FIREBASE_API_KEY=AIzaSyAmy-dyZAi5rTk3MtLuj5n17S3F1lQU4SU
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN=diploma-project-1ff8d.firebaseio.com
NEXT_PUBLIC_FIREBASE_PROJECT_ID=diploma-project-1ff8d
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET=diploma-project-1ff8d.appspot.com
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=1040105921693
NEXT_PUBLIC_FIREBASE_APP_ID=1:1040105921693:web:b34a038c4003989cdd0874
NEXT_PUBLIC_FIREBASE_MEASUREMENT_ID=G-6JC66HJBVP
```

Фиг. 4.1

A screenshot of a terminal window. At the top, there are tabs for 'Terminal' and 'Local'. Below the tabs, the terminal prompt shows the user's name 'rize' followed by the host 'rize-HP-Pavilion-Laptop-15-eh1xxx', the directory '~/WebstormProjects/diploma_project/diploma-project\$', and a dollar sign '\$'. The background of the terminal is light gray.

Фиг. 4.2

4.2. Графичен потребителски интерфейс

Графичния потребителски интерфейс на уеб приложението е сравнително обемен, пореди факта че има тъмна и бяла тема и различна визия за всеки вид устройства - телефони, таблети и компютри. В тази подглава ще се фокусираме върху визията за компютри като ще се демонстрират смесено бяла и тъмна тема. Уеб приложението има 3 различни изгледа - за гости (това са хора без акаунти), потребители и занаятчии. Ще разгледаме страниците, които са еднакви и за трите изгледа, а именно логин и регистър страниците (Фигури 4.3, 4.4, 4.5 и 4.6). И двата екрана се състоят от полета, изискващи различни входни данни и лилав бутон за завършване на влизането или регистрацията. Освен това страниците имат бутон, който сочи към другата - на логин страницата тъмносивият бутон "sign up" навигира до регистър страницата, а на регистър страницата тъмносивият бутон "sign in" навигира до логин страницата.

The image shows a light-themed user interface for a web application. On the left, there is a 'Login to Your Account' section with fields for Email (email@gmail.com) and Password (password123), and a purple 'SIGN IN' button. In the center, there is a 'New here?' section with a dark grey 'Sign Up' button. Below it, a text block says 'Sign up and discover a great amount of new crafts and opportunities'.

Фиг. 4.3

The image shows a light-themed user interface for a web application. On the right, there is a 'Create Free Account' section with fields for Full name (Melvin Simonds), Email (email@gmail.com), Password (password123), Confirm Password (password123), and a purple 'SIGN UP' button. In the center, there is a 'One of us?' section with a dark grey 'Sign In' button. Below it, a text block says 'If you already have an account just sign in and discover a great amount of new crafts'.

Фиг. 4.4

The image shows a dark-themed user interface for a web application. On the left, there is a 'Create Free Account' section with fields for Full name (Melvin Simonds), Email (email@gmail.com), Password (password123), Confirm Password (password123), and a purple 'SIGN UP' button. In the center, there is a 'One of us?' section with a dark grey 'Sign In' button. Below it, a text block says 'If you already have an account just sign in and discover a great amount of new crafts'.

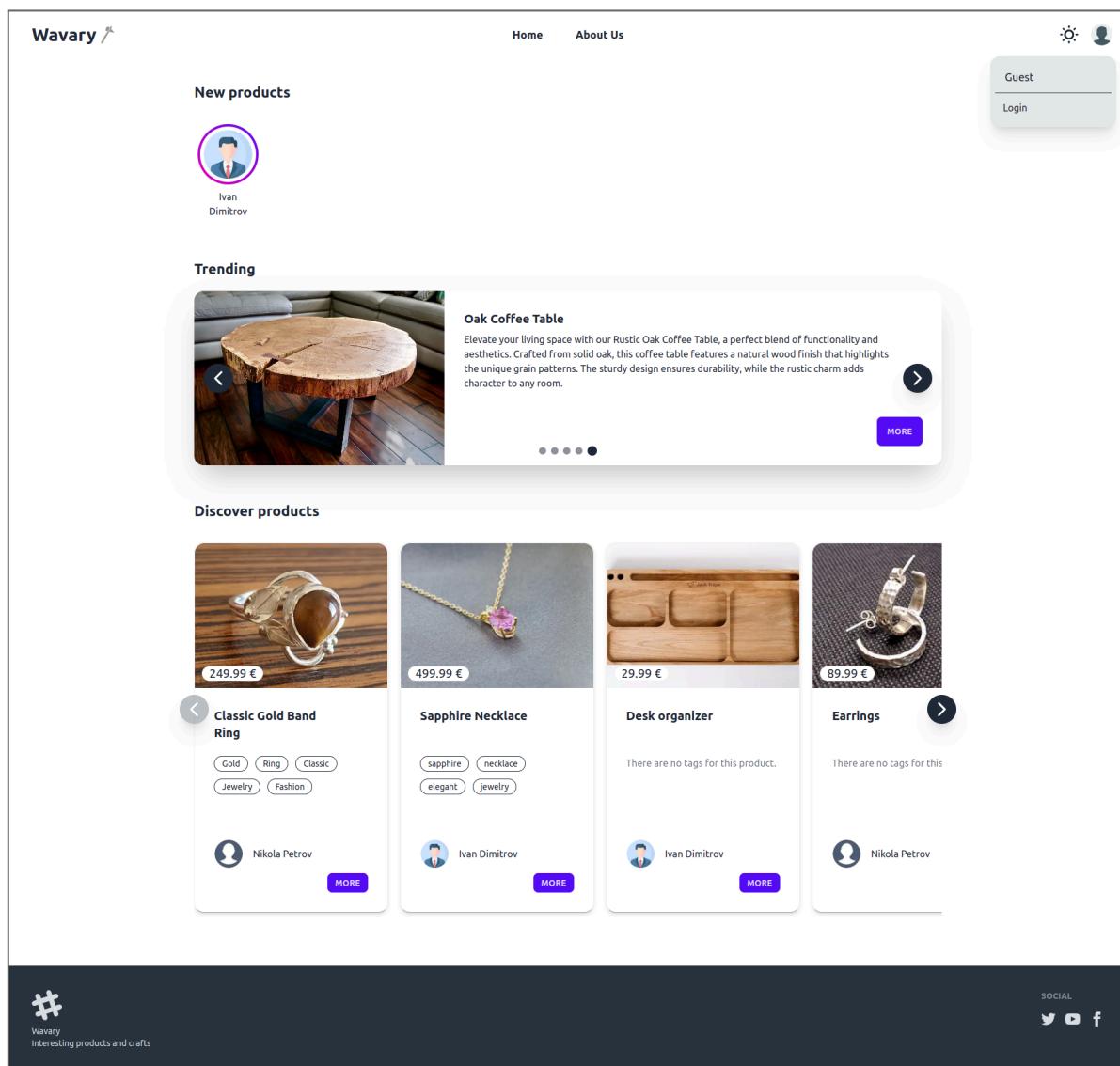
Фиг. 4.5

The image shows a dark-themed user interface for a web application. On the left, there is a 'Login to Your Account' section with fields for Email (email@gmail.com) and Password (password123), and a purple 'SIGN IN' button. In the center, there is a 'New here?' section with a dark grey 'Sign Up' button. Below it, a text block says 'Sign up and discover a great amount of new crafts and opportunities'.

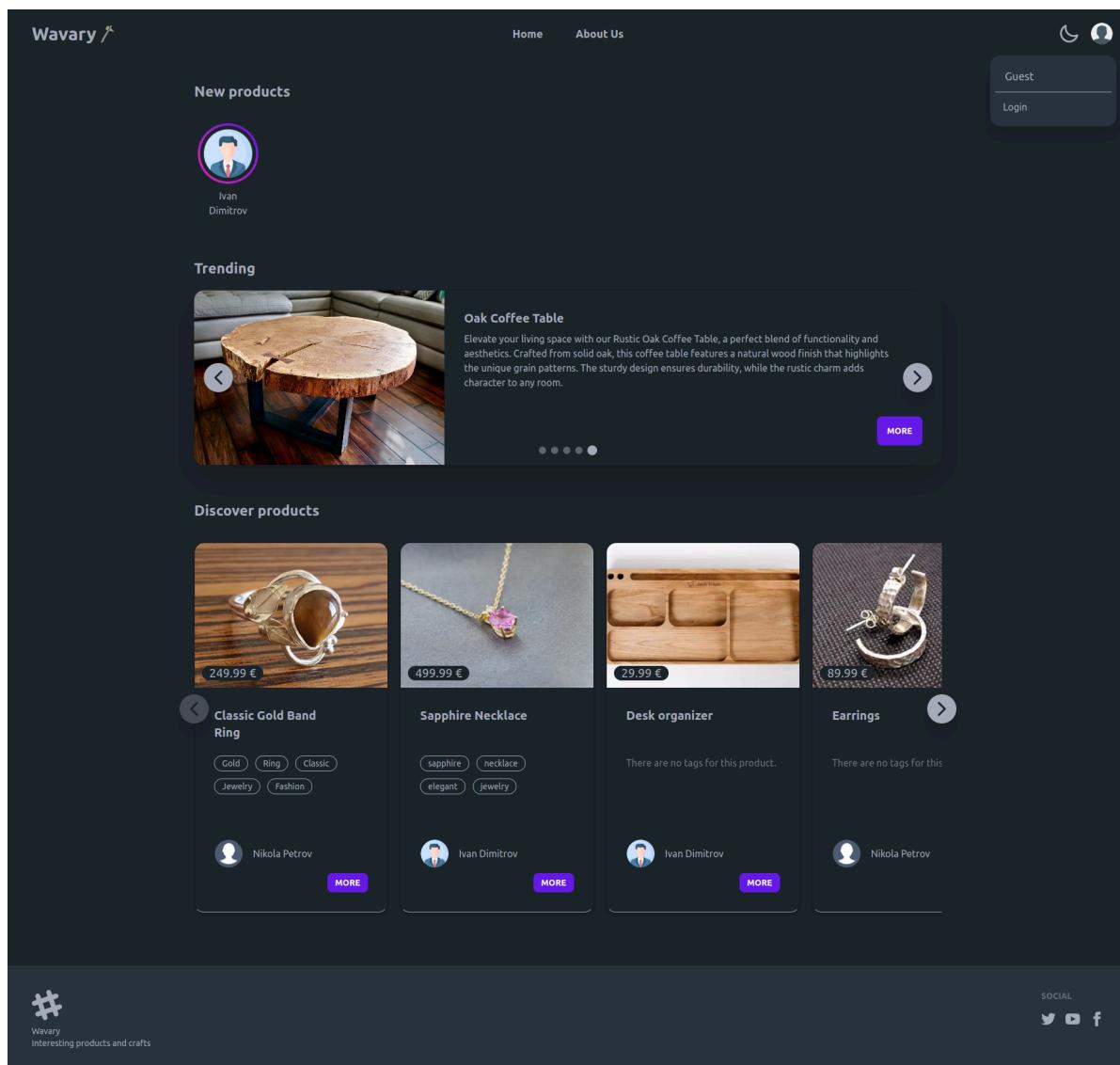
Фиг. 4.6

4.2.1. Графичен потребителски интерфейс за гости

Първо ще погледнем какво представлява графични потребителски интерфейс за гости на уеб приложението. Идеята на госта е да разгледа какво предлага платформата. Понеже няма никакъв вид идентификация няма право да взаимодейства с уеб приложението като добавя, променя или трябва съдържание независимо дали са продукти или поръчки. Цялостната идея на гостите е да добият представа за платформата и след това ако желаят да създадат свой акаунт. На първо място гостите виждат началната страница (Фигури 4.7 и 4.8).



Фиг. 4.7

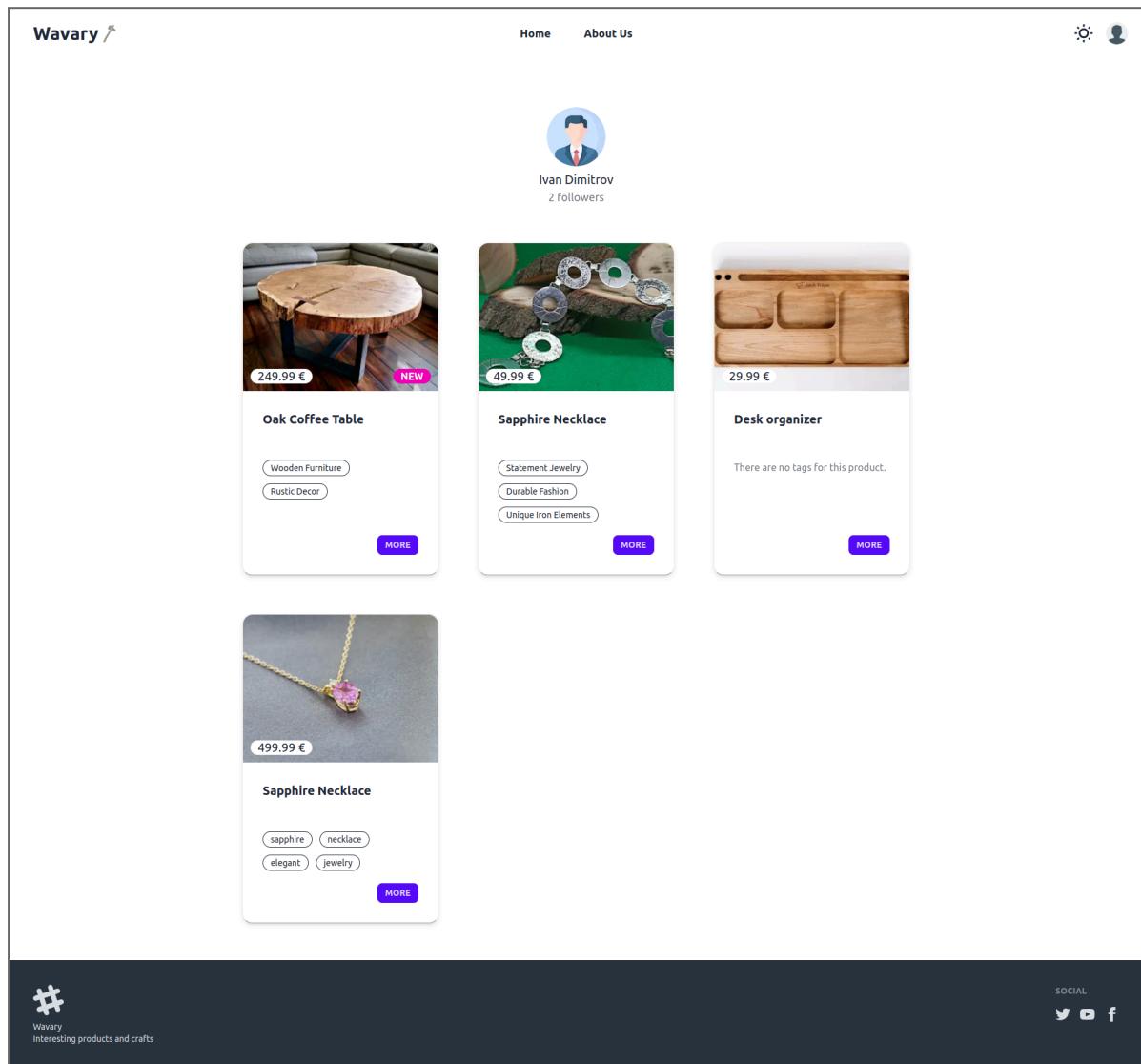


Фиг. 4.8

Заглавната страница е еднаква като структура за всички потребители и занаятчии, единствената разлика е в навигационната лента. В него имаме 2 страници - “home” и “about”, бутон за смяна между тъмната и светла тема и профилна снимка по подразбиране, натискайки я се показва падащо меню с бутон, който ни препраща към логин страницата. На началната страница се забелязват 3 слайдера - нови продукти, популярни продукти и други продукти. Новите продукти за изделия добавени в платформата в последните 24 часа. Популярните продукти са изделията с най-много харесвания за последната седмица. Други продукти са всякакви продукти, които са на поне ден и на не повече от месец от добавянето си. В слайдера

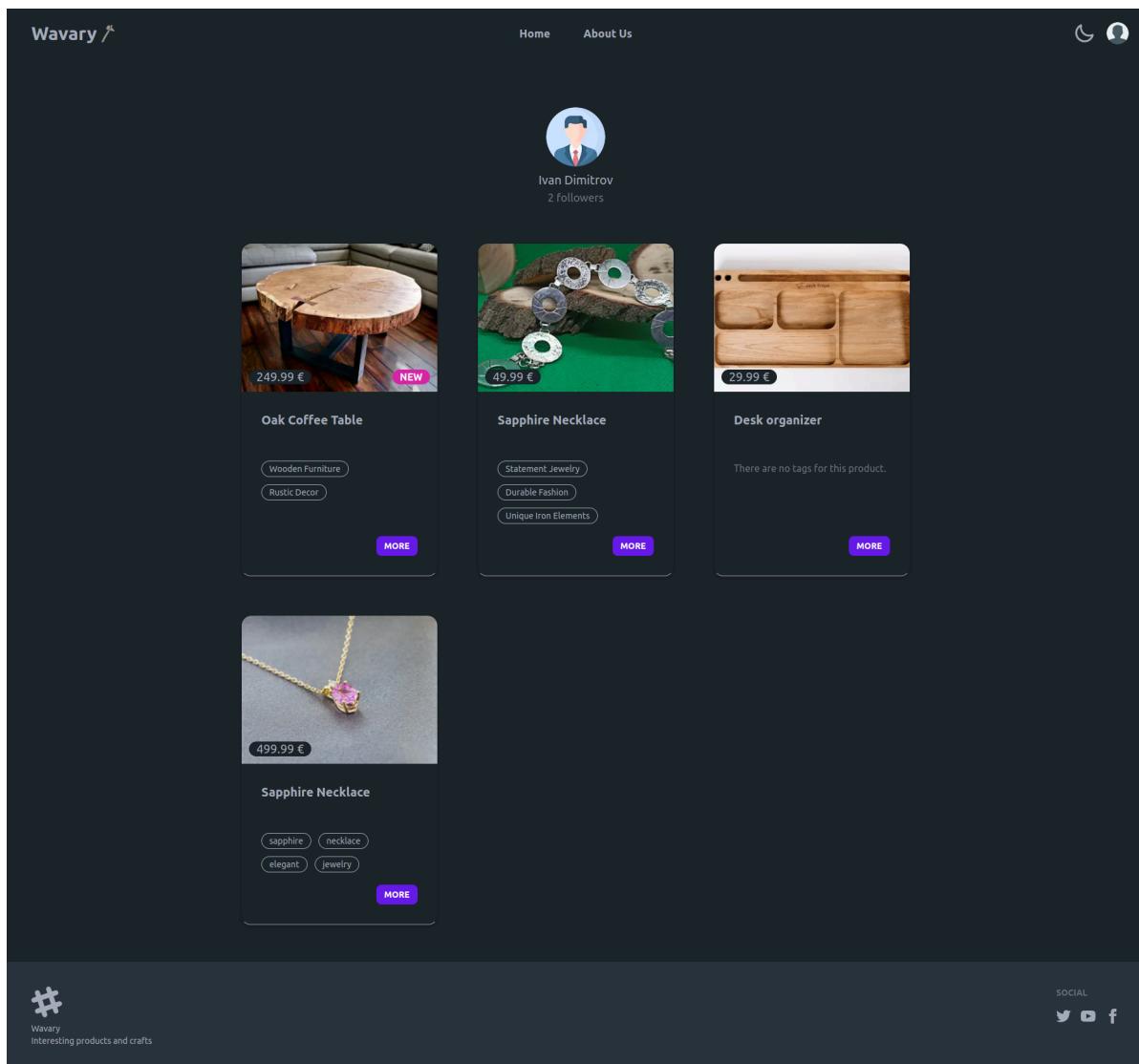
с новите продукти при натискане на името на занаятчия се озовавате на страницата с каталога му, а при натискане на профилната му снимка оградена с розово-лилав градиент в същия слайдер се озовавате на страницата на новият продукт. В слайдерите за популярни и други продукти са изобразени изделия със съкратена информация (карта), за по-бързо запознаване с детайлите. Ако искате да разгледате по-подробно продукта може да натиснете лилавият бутона “more”, който ще ви отведе до страницата на продукта.

Страницата за каталога (Фигури 4.9 и 4.10) съдържа същата навигационна лента като началната страница.



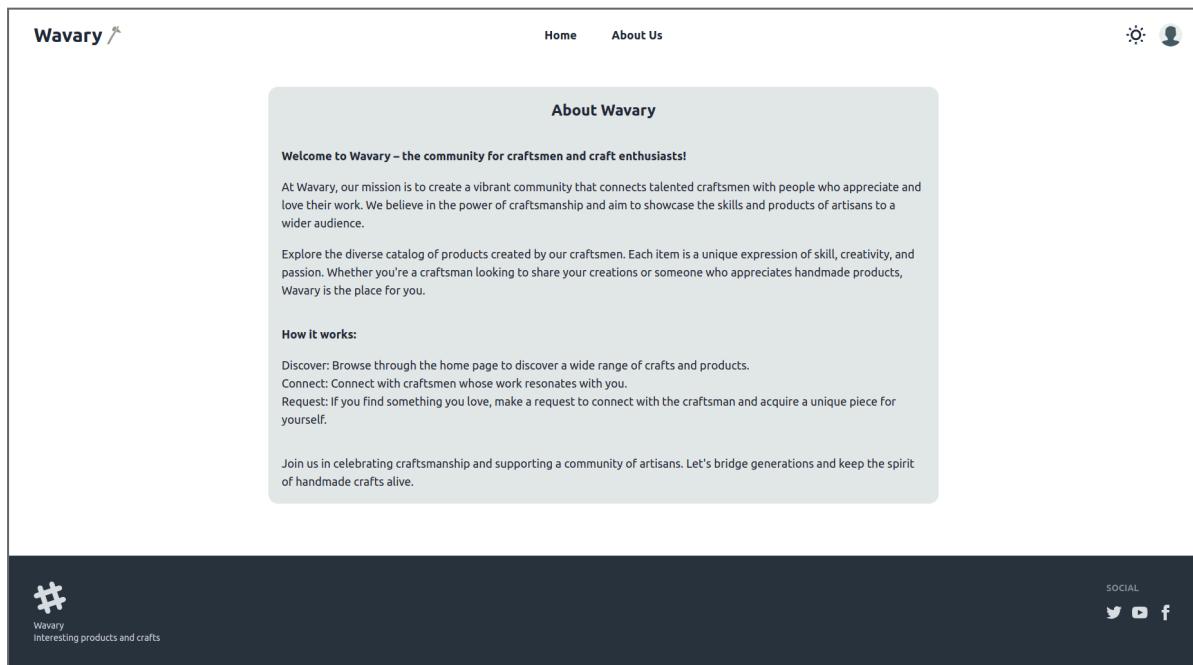
Фиг. 4.9

Съдържанието на страницата е съставено от името, профилна снимка, броя на последователи и продуктите на занаятчията. Всеки продукт е представен в съкратена форма (карта) така че да позволи бързото запознаване с него от страна на потребителя. В допълнение имат етикета “new”, когато изделието е добавено преди по-малко от 24 часа. И както и на началната страница, продуктите имат лилав бутон “more”, който води до страницата на продукта, където можем да се запознаем в детайл с информацията за изделието.

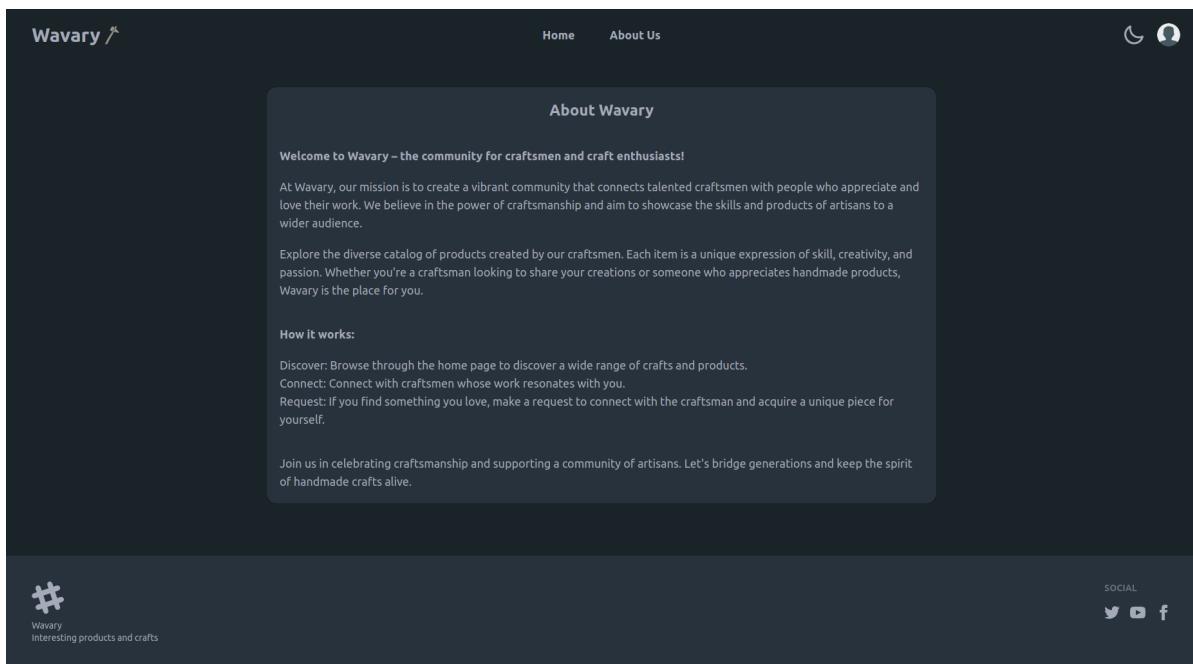


Фиг. 4.10

Преминаваме към “about” страницата (Фигури 4.11 и 4.12). Нейната идея е да обясни на потребителите, гостите и занаятчиите целта на уеб приложението и неговите функционалности.

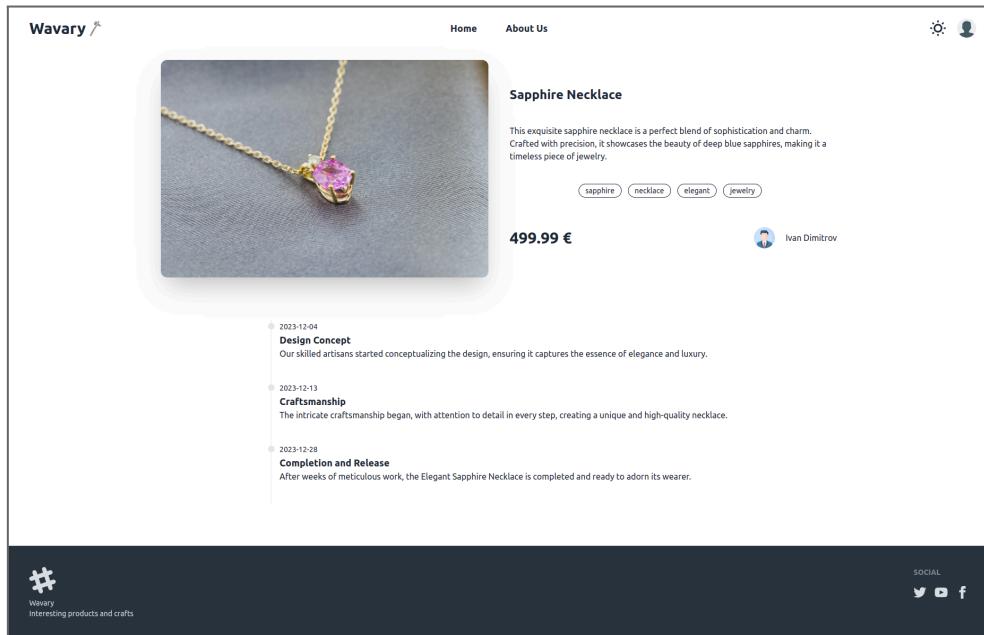


Фиг. 4.11

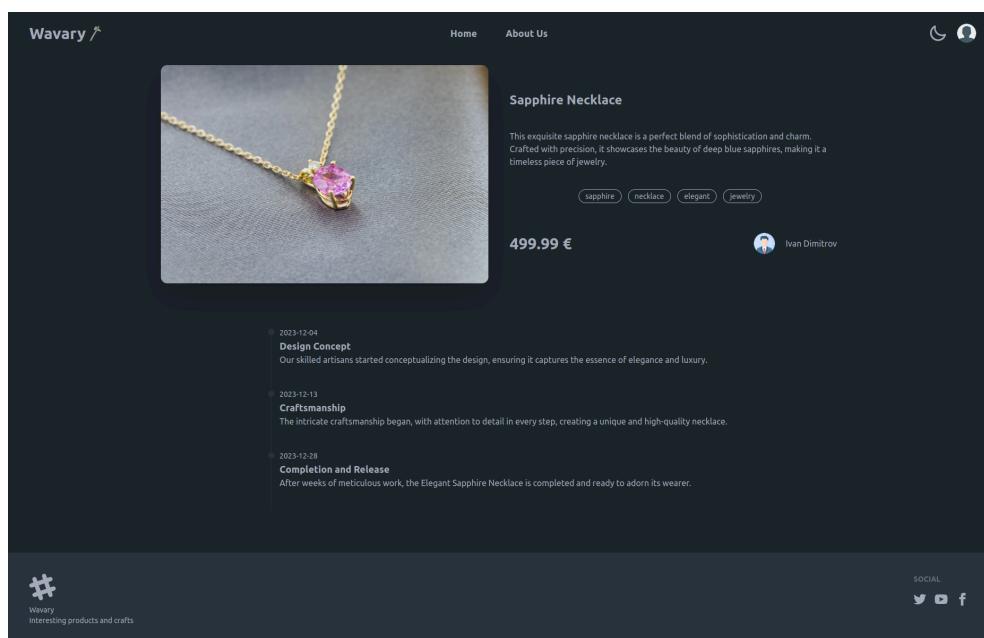


Фиг. 4.12

Страницата на продукта е малко по-сложна поради големия обем на информация. Тя съдържа изображение, заглавие, описание, тагове, цена и линия на времето на продукта, както и профилна снимка и име на занаятчията (Фигури 4.13 и 4.14). Както при другите страници, когато се натисне профилната снимка на занаятчията гостът се препраща към каталога с изделията на артиста.



Фиг. 4.13



Фиг. 4.14

4.2.2. Графичен потребителски интерфейс за потребители

Потребителите вече имат акаунти и това им позволява да взаимодействат по повече начини с графичния потребителски интерфейс за разлика от гостите. Първата забележима разлика е че продуктите могат да бъдат харесвани (Фигури 4.15 и 4.16).

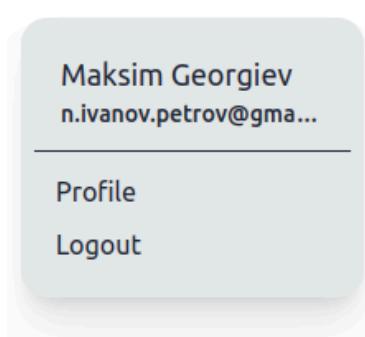


Фиг. 4.15



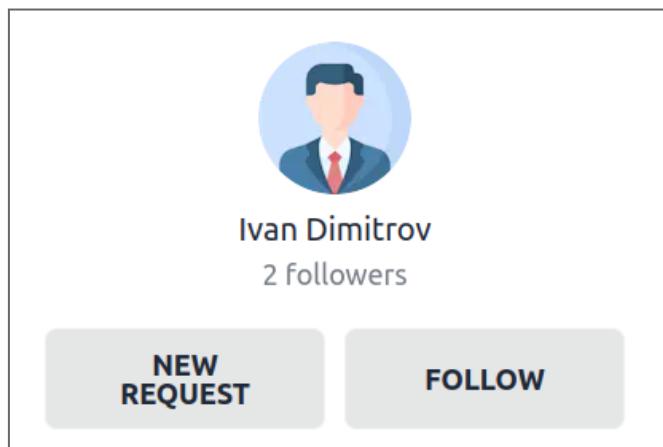
Фиг. 4.16

Друга разлика се забелязва при падащото меню в навигационната лента, където вече се изписват пълното име и имейлът на потребителят (Фигура 4.17). Освен това имаме бутон, свързан с профилната страница и бутон за излизане от профила.



Фиг. 4.17

Освен това в страницата на каталога се забелязват 2 нови бутона - за нова поръчка и за последване (Фигура 4.18). Бутона за нова поръчка отваря панел с полета за заглавие и описание на поръчката (Фигури 4.19 и 4.20), когато сме готови с въвеждането на поръчката при натискане на лилавият бутон “add request” поръчката се отправя към занаятчията.



Фиг. 4.18

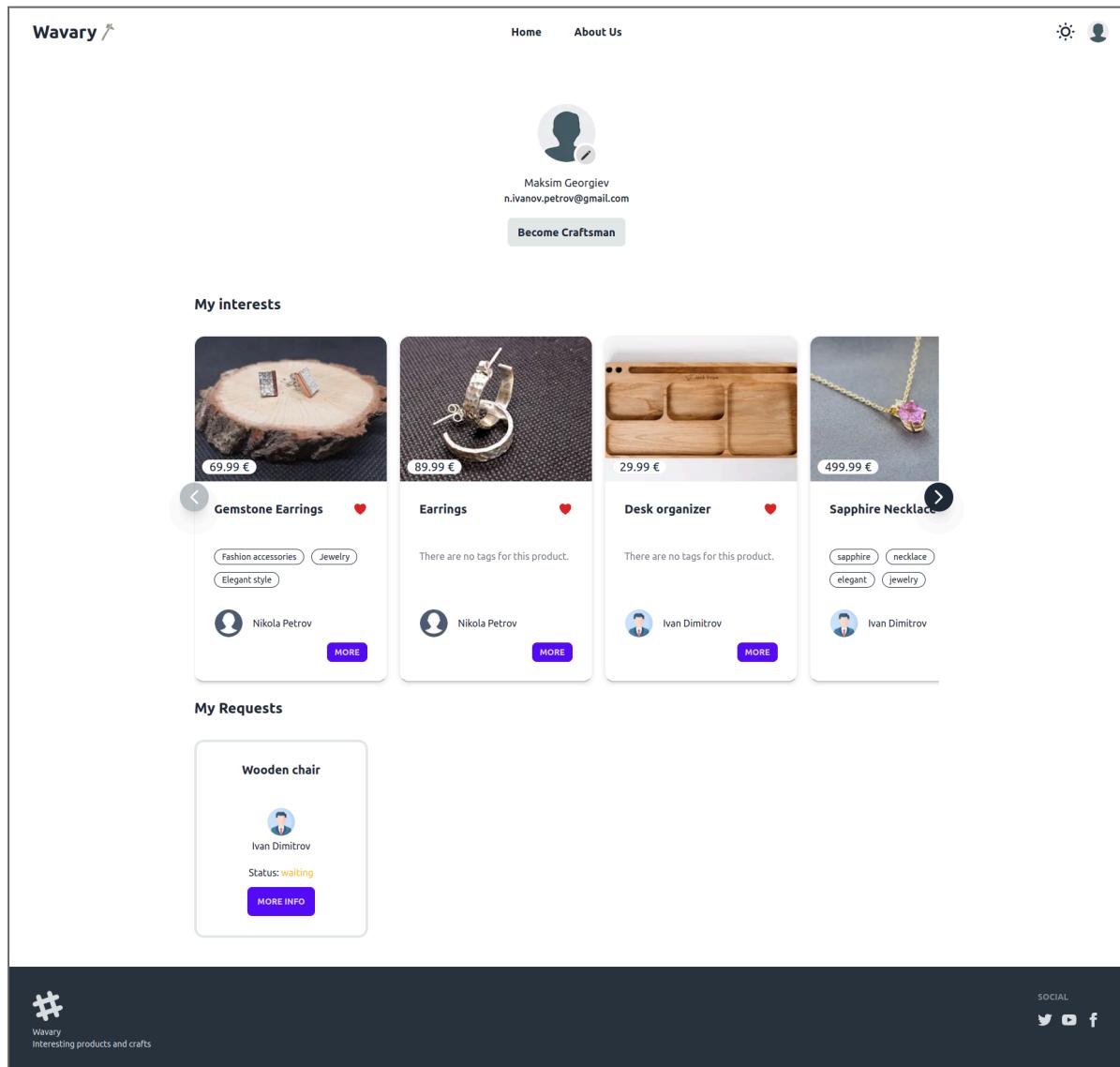
A light-colored 'New Request' form with fields for 'Title' and 'Description', and a purple 'ADD REQUEST' button at the bottom.

Фиг. 4.19

A dark-themed 'New Request' form with fields for 'Title' and 'Description', and a purple 'ADD REQUEST' button at the bottom.

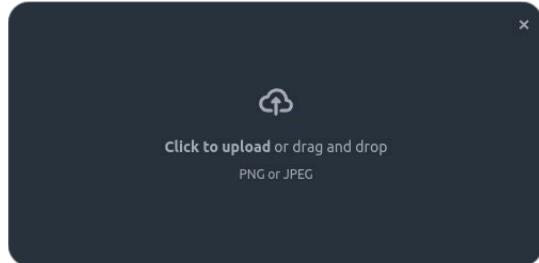
Фиг. 4.20

Най-голямата разлика между госта и потребителя е присъствието на профилната страница, тя съдържа информация за потребителя - пълно име, имейл и профилна снимка, както и бутона за обозначаване като занаятчия “become craftsman”, бутона за сменяне на профилната снимка и 2 слайдера - един за харесаните продукти и втори за направените поръчки (Фигура 4.21).



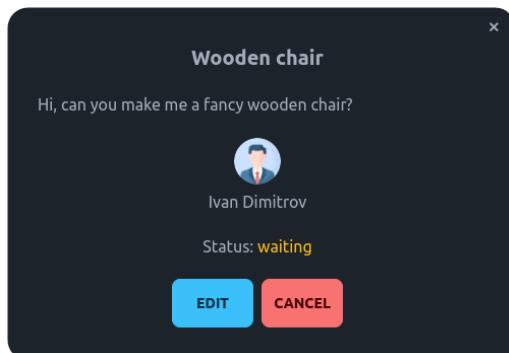
Фиг. 4.21

При натискане на бутона с молива, който се намира точно до профилната снимка, се показва диалогов прозорец, в който може да се избере профилна снимка чрез плъзгане или чрез натискане на прозореца (Фигура 4.22).

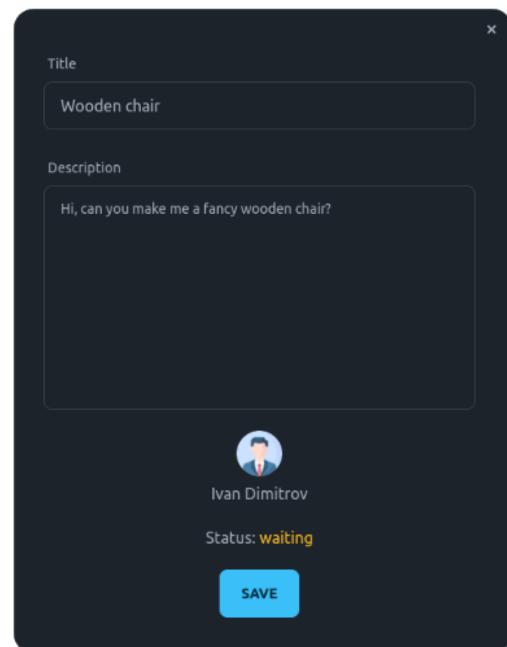


Фиг. 4.22

Диалогов прозорец се показва и при натискане на бутона “more info”, намиращ се на всяка една карта на поръчката (Фигура 4.23). В него се описват детайлите на поръчката като съответно ти позволява да я редактирате при натискане на бутона “edit” или отмените при натискане на бутона “cancel”. Когато правите промени по запитването след като свършите трябва да натиснете бутона “save”, за да запазите редакцията си (Фигура 4.24).



Фиг. 4.23



Фиг. 4.24

4.2.3. Графичен потребителски интерфейс за занаятчии

Разликите се виждат още в навигационната лента - имаме нов бутон “New Product” (Фигура 4.25), който отваря панел за добавяне на нов продукт.



Фиг. 4.25

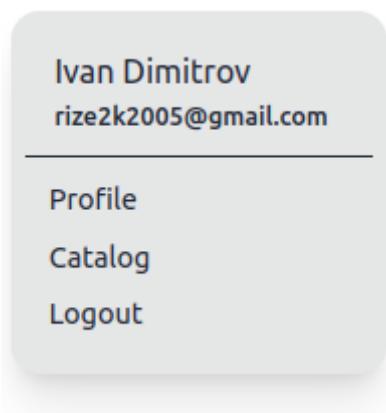
В панела за добавяне на нов продукт има няколко ключови елемента - полета за входни данни за заглавие, изображение, описание, цена, линия на времето и тагове (Фигура 4.26).

A screenshot of a "New Product" form. The form has several input fields: "Title" (empty), "Display Image" (button labeled "CHOOSE FILE" with "No file chosen"), "Description" (empty), "Price €" (empty), "Timeline" (button labeled "ADD SECTION"), "Tags - Not required" (button labeled "ADD TAG" next to an empty input field), and a large gray placeholder area. At the bottom is a prominent purple "ADD PRODUCT" button.

Заглавието не може да бъде повече от 25 символа, а описанието не може да бъде повече от 500 символа. Цената може да е както цяло число, така и число с плаваща запетая до втория знак (всеки следващ се игнорира). Таговете могат да бъдат най-много 5 за да не се пренасити картата на продукта. Полето за изображението приема само jpeg и png формати. Линията на времето има избираем брой на секции. Всяка секция има заглавие, описание и дата. Заглавието не може да бъде повече от 40 символа, а описанието повече от 300 символа. В допълнение датата не може да бъде в бъдещето.

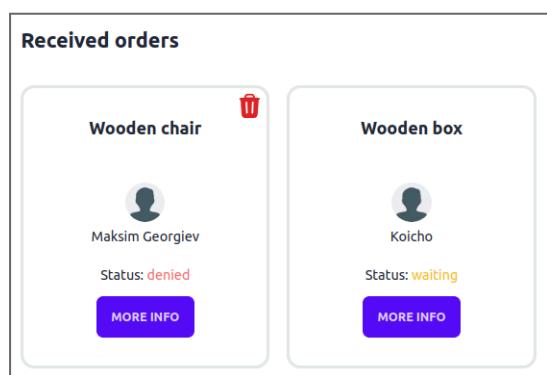
Фиг. 4.26

Друга забележима разлика е че вече като занаятчия имаме достъп до своя каталог чрез падащото меню (Фигура 4.27). Важно е да се отбележи, че занаятчиите, които се навигират до своя каталог нямат бутоните за нова поръчка и последване. В допълнение не могат да харесват свои произведения, за да не се запазват в профила им.

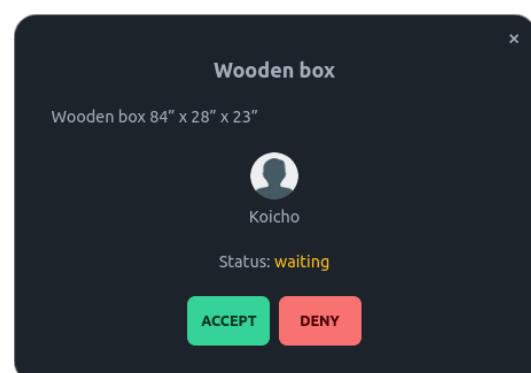


Фиг. 4.27

Последната значителна разлика в графичния потребителски интерфейс е в профилната страница, където за занаятчиите е видим и третия слайдер за получените поръчки (Фигура 4.28). Всяка получена поръчка при занаятчията му позволява да я одобри или откаже в зависимост неговата заинтересованост (Фигура 4.29).



Фиг. 4.28



Фиг. 4.29

Заключение

Уеб приложението е напълно способно да помогне за популяризиране на занаятите. Платформата е направена, за да спомогне изграждането на занаятчийска общност. По този начин занаятчиите могат да се наслаждават на изделияя, да организират поръчките си и да представят своите творби на клиентите си в заинтригиващ и детайллен вид. Има още поле за развитие на уеб приложението, някои неща, които планирам да добавя са:

- Плащане с кредитни и дебитни карти. Това ще направи платформата по-практична, понеже няма само да представя продукта с цената, а и ще има възможност за директно закупуване.
- Търсачка за продукти. Ако се добави търсачка клиентите ще могат да намерят по-лесно търсените от тях изделияя.
- Нотификации. При добавяне на нотификации, платформата ще придобие още по-близка визия на социална мрежа, което ще допринесе за формирането на занаятчийската общност.
- Наддаване за изделияя. Услугата ще се допринесе за активността на потребителите и ще направи платформата по-интересна.
- Търсене по занаяти. Ако се добави страница с продукти само от един занаят, хората ще могат да открият повече изделияя по своите интереси. В допълнение може да се добавят занаяти, които потребителя харесва, в началната страница, за да се подобри популяризирането на занаятите и изделияята.

Литература

- HTML, CSS и JavaScript -
<https://blog.hubspot.com/marketing/web-design-html-css-javascript>
- React - <https://react.dev/reference/react>
- Vue - <https://vuejs.org/guide/introduction.html>
- Angular - <https://angular.io/guide/what-is-angular>
- Next - <https://nextjs.org/docs>
- Nuxt - <https://nuxt.com/docs/getting-started/introduction>
- Nest - <https://docs.nestjs.com/>
- Tailwind - <https://tailwindcss.com/docs/installation>
- Bootstrap - <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- Material UI - <https://mui.com/material-ui/getting-started/>
- Daisy UI - <https://daisyui.com/>
- Zustand - <https://docs.pmnd.rs/zustand/getting-started/introduction>
- Firebase - <https://firebase.google.com/docs/build>
- AWS Amplify - <https://docs.amplify.aws/>
- Etsy - <https://www.etsy.com/>
- Folksy - <https://folksy.com/>

Често използвани термини

- DOM (Document Object Model) - представя структурата и съдържанието на HTML документ под формата на йерархично дърво.
- фреймуърк - структура от предварително дефинирани кодове или библиотеки, предназначена да улесни и ускори процеса на разработка на софтуер.
- библиотека - колекция от готов код, който програмистите могат да включат в своите приложения.
- рендериране от страна на сървъра - възможността на сървъра да изпраща като отговор напълно изобразена страница. По този начин се преобразува HTML файла и браузърът получава готовата визуализация за потребителя.
- рендериране от страна на клиента - HTML кодът се изпраща на браузъра на клиента. След това JavaScript кодът в браузъра обработва и преобразува HTML кода в HTML страница.
- статично генериране - HTML страниците се генерират предварително на сървъра.
- CDN (Content Delivery Network) - мрежа от сървъри, разпределени по целия свят. Нейната цел е да доставя съдържание (като изображения, JavaScript файлове, CSS файлове) до клиентите от най-близкия сървър.
- фронтенд - частта от приложението, с която потребителят взаимодейства директно.
- бекенд - частта от приложението, която се изпълнява на сървъра.
- състояние - данни, които се променят и влияят на графичния потребителски интерфейс.

- API - интерфейс, който позволява на приложенията да взаимодействат помежду си. Определя набор от правила и функции за достъп до данни и услуги.
- URL - адрес, който определя местоположението на ресурс в интернет.
- json формат - формат за обмен на данни, който е лесен за четене и писане както от хора, така и от машини.
- фулстак уеб апликация - приложение, чийто разработчик е компетентен във фронтенд и бекенд разработката.
- уеб браузър - посредник между потребителя и уеб сървъра. Когато потребителят въведе URL адрес в браузъра, той изпраща HTTP заявка до сървъра. Сървърът връща HTML код, който браузърът интерпретира и визуализира като уеб страница.

Съдържание

Задание.....	2
Отзив.....	3
Увод.....	4
Първа глава.....	5
Проучване.....	5
1.1. Основни принципи и технологии за реализиране уеб приложения...5	
1.1.1. Основни технологии за разработка на уеб приложения..... 5	
1.1.2. Библиотеки за разработка на фронтенд на уеб приложения..... 6	
1.1.3. Технологии за разработка на бекенд на уеб приложения.....11	
1.2. Съществуващи решения и реализации..... 13	
1.2.1. Etsy..... 13	
1.2.2. Folksy..... 15	
Втора глава.....	17
Определяне на изисквания и технологии.....	17
2.1. Функционални изисквания към уеб базиран електронен магазин...17	
2.2. Съображения за избор на програмни средства и развойната среда. 18	
2.2.1. Фронтенд технологии..... 18	
2.2.2. Бекенд технологии..... 20	
2.2.3. Интегрирана среда за разработка..... 21	
2.3. Структура на база данни и облачно хранилище..... 21	
Трета глава.....	28
Програмна реализация.....	28
3.1. Архитектура на разработваното уеб приложение.....28	
3.2. Структура на проекта в WebStorm..... 29	

3.3. Състояние на уеб приложението.....	30
3.4. Автентикация на уеб приложението.....	30
3.5. Фронтенд на уеб приложение.....	33
3.6. Бекенд на уеб приложението.....	45
Четвърта глава.....	49
Ръководство на потребителя.....	49
4.1. Инсталация на уеб приложението.....	49
4.2. Графичен потребителски интерфейс.....	50
4.2.1. Графичен потребителски интерфейс за гости.....	51
4.2.2. Графичен потребителски интерфейс за потребители.....	57
4.2.3. Графичен потребителски интерфейс за занаятчии.....	61
Заключение.....	63
Литература.....	64
Често използвани термини.....	65