



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ-СОФИЯ

Факултет Компютърни системи и технологии

ДИПЛОМНА РАБОТА

ТЕМА:

*Разработване на система за онлайн
търговия*

Дипломант:

Красимир Митков, КСИ, фак.№ 121219117

Дипломен ръководител:

проф. д-р инж. Милена Лазарова

София 2023

Съдържание

Списък с таблици.....	3
Списък с фигури.....	3
Увод.....	5
Глава 1 Постановка на дипломната работа, цели и задачи	7
1.1. Въведение.....	7
1.2. Цел	8
1.3. Сравнение	9
1.4. Постановяне на задачи за решаване.....	12
1.4.1. Списък със задачи за изпълнение.....	12
Глава 2 Проектиране на системата	14
2.1. Описание на приложението	14
2.2. Описание на потребителски интерфейс.....	16
2.3. Описание на избор на архитектура	22
2.3.1. Цялостна архитектура	22
2.3.2. Архитектура на сайта.....	23
2.3.3. Архитектура на мобилното приложение.....	24
2.3.4. Архитектура на API-а	25
2.4. UML диаграми.....	27
2.4.1. Use-case диаграма.....	27
2.4.2. Sequence диаграма.....	28
2.5. Описание на базата данни.....	29
2.5.1. ER диаграма на базата данни.....	22
2.6. Описание на технологии и програмни средства.....	33
2.6.1. Разработване на сайта	33
2.6.2. Разработване на мобилното приложение.....	36
2.6.3. Разработване на API-а	37
2.6.4. Разработване на API-а за известяване на мобилното приложение	39
2.6.5. Програмни среди	40
2.7. Сигурност	41
Глава 3 Програмна реализация	42
3.1. Класови диаграми.....	42
3.2. Реализация на автентификация и авторизация.....	44
3.3. Реализация на работа с продукти	46
3.4. Реализация на работа с обекти	47
3.5. Реализация на поръчка и плащания	49

3.6.	Реализация на комуникация между мобилното приложение и сървъра.....	52
3.7.	Реализация на обработката на поръчки в мобилното приложение	53
Глава 4 Ръководство за ползване		54
4.1.	Софтуерни и хардуерни изисквания	54
4.2.	Визуализация на потребителски интерфейс и ръководство.....	54
4.3.	Тестване	68
Заклучение		72
Източници		74
Приложение код		75
1.	Добавяне на нов магазин	75
	Controller.....	75
	Service	76
	Component.....	77
2.	Визуализиране на всички букети	82
	Controller.....	82
	Service	82
	Component.....	83
3.	Нова поръчка	84
	Controller.....	84
	Service	84
	Component.....	86

Списък с таблици

Таблица 1.3.1 Сравнителен анализ на онлайн магазини за продажба на букети.....	10
Таблица 4.3.1 Матрица на функционалните изисквания.....	69

Списък с фигури

Фигура 2.2.1 Начална страница на bouquet.bg.....	16
Фигура 2.2.2 Начална страница на мобилното приложение	17
Фигура 2.2.3 Страници за регистрация, логин и забравена парола.....	18
Фигура 2.2.4 Страница за магазините.....	18
Фигура 2.2.5 Дизайн на страницата за кошницата	19
Фигура 2.2.6 Страница за конкретен обект	20
Фигура 2.2.7 Дизайн на страницата за добавяне на букети и приключване на поръчки.....	20
Фигура 2.2.8 Дизайн на страница за поръчки (мобилно).....	21
Фигура 2.2.9 Дизайн на страница за поръчка (мобилно).....	21
Фигура 2.3.1 Графични представяне на цялостната архитектура	22
Фигура 2.3.2 Графични представяне на MVVM архитектурата	25
Фигура 2.4.1 Use-case диаграма на системата bouquet.bg	27
Фигура 2.4.2 Sequence диаграма представяща поръчката.....	28
Фигура 2.5.1 ER диаграма на базата данни Bouquet.DB.....	32
Фигура 2.6.1 Диаграма как работи Hamarín.....	36
Фигура 3.1.1 Класова диаграма на главния екран на мобилното приложение.....	42
Фигура 3.1.2 Класова диаграма на моделите.....	43
Фигура 4.2.1 Начална страница на системата bouquet.bg.....	55
Фигура 4.2.2 Страница за вход в системата bouquet.bg (уеб)	56
Фигура 4.2.3 Страница за вход в системата bouquet.bg (мобилно)	57
Фигура 4.2.4 Първа страница за регистрация на потребители	58
Фигура 4.2.5 Страница за регистрация на частно лице	59
Фигура 4.2.6 Страница за регистрация на компания.....	59
Фигура 4.2.7 Страница с всички магазини	60
Фигура 4.2.8 Страница на магазин.....	61
Фигура 4.2.9 Страница за добавяне на букет	62
Фигура 4.2.10 Страница за кошницата.....	62
Фигура 4.2.11 Страница за поръчка	63
Фигура 4.2.12 Страница с разплащателни карти	64
Фигура 4.2.13 Страница за добавяне нова карта.....	64

Фигура 4.2.14 Страница история на поръчки.....	65
Фигура 4.2.15 Начална страница на мобилното приложение	66
Фигура 4.2.16 Страница за поръчка (мобилно)	66
Фигура 4.2.17 Меню мобилно приложение	67
Фигура 4.2.18 Настройки на мобилното приложение.....	67

Увод

В съвременния бързо развиващ се свят, технологиите играят ключова роля в промяната на начина, по който хората се свързват, комуникират и осъществяват търговия. От историческа гледна точка, търговията винаги е била важен елемент от човешката цивилизация, променяйки се под влиянието на средствата за комуникация и иновациите в информационните технологии. В този контекст, се налага разработването на съвременни решения, които да отговарят на нуждите на бизнеса и да облекчат преживяването на потребителите.

В днешния бързо заселен свят, времето е ценно. Хората търсят начини да оптимизират ежедневните си задачи и да ги извършват с минимално усилие. В този контекст, онлайн платформите за търговия играят решаваща роля, като предоставят възможността за лесно и удобно пазаруване, без да е необходимо физическо присъствие. Независимо дали става въпрос за книги, електроника или дори цветя, онлайн магазините предлагат бърз и ефективен начин за пазаруване, което отговаря на съвременните потребителски предпочитания.

Тази дипломна работа има за цел да разгледа разработването на система за онлайн търговия, насочена към цветарски магазини. Чрез създаването на уеб магазин и мобилно приложение за обработка на поръчки, ще се обсъдят ключовите аспекти на проектирането и изграждането на такава система. Въз основа на добре утвърдени методологии и съвременни технологии, целта е да се предостави интегрирано и удобно решение, което да позволи на множество цветарски магазини да разширят своя бизнес в онлайн средата.

В следващите глави на дипломната работа ще бъдат разгледани подробности свързани със създаването на системата.

- Глава 1 – ще анализира целите на разработката. Обозначаване на областта на работа и тематиката на разработванията. Ще бъде представен сравнителен анализ между системи от подобен тип по различни показатели. В края на главата ще бъдат поставени задачите за изпълнение в процеса на разработване на крайния продукт.

- Глава 2 – ще се фокусира върху проектирането на системата, архитектурата на приложението, диаграмите за реализирането му, описание на базите данни, използваните технологии, алгоритми и подходи.
- Глава 3 – тук ще бъде поместена програмната реализация на системата. Документиране на нейните класове и методи.
- Глава 4 – ръководство за ползване. В нея ще бъдат описани всички нужни софтуерни и хардуерни изисквания за коректната работа на системата, снимки на интерфейса, както и процесът по тестването ѝ.

Чрез изграждането на тази система, целта е да се допринесе за улесняването на търговията и оптимизирането на бизнес процесите в сферата на цветарските магазини, като се предостави на клиентите удобна и ефективна платформа за онлайн пазаруване.

Глава 1 Постановка на дипломната работа, цели и задачи

1.1. Въведение

Както вече беше споменато, онлайн търговията заема ключова роля в преобразяването на начина, по който бизнесът функционира и как потребителите изпълняват своите покупки. С разрастването на електронната комерсия, онлайн магазините и мобилните приложения са се превърнали в неотменими инструменти за търговия, които предоставят удобство, достъпност и бързина на потребителите. В днешния свят, където времето се явява ограничен ресурс, потребителите насочват вниманието си към решения, които им позволяват да извършват задачи ефективно и без усилие. В този контекст, традиционните търговски методи, които изискват физическа посещаемост и време за извършване на покупки, често биват изместени от онлайн търговията. Този преход предоставя на клиентите възможността да пазаруват продукти от различни категории, включително и цветя, с няколко клика на мишката или докосване на екрана.

С цел да се подобри и оптимизира процесът на онлайн търговия, представяме дипломната работа, която се фокусира върху разработването на система за управление на онлайн търговия, специфично на платформа за цветарски магазини. Чрез създаването на уеб магазин и мобилно приложение за обработка на поръчки, тази система предоставя възможност на множество цветарски магазини да предлагат своите продукти и услуги в една централизирана онлайн среда.

Системата за управление на онлайн търговия подпомага работата на служителите в цветарските магазини и техните клиенти. Тя предоставя информация в реално време за продуктите, налични в цветарските магазини, за нейните потребители, било то служители или клиенти. Това позволява лесен и бърз избор на желания артикул измежду множество от различни магазини. Системата е с потребителски интерфейс, който спомага за по-лесната работа с приложението. По този начин клиентите ще могат да разгледат всички продукти, да си изберат такива и да ги закупят и получат по най-удобния за тях начин. Системата позволява на собствениците на магазини да извършват онлайн продажби, без да е нужно да имат собствена платформа, да управляват обектите си по лесен и удобен начин, да добавят и премахват

служители, да следят поръчките. Приемането на поръчки и уточняване данните за доставка е много по-лесно и по-малко време отнемащо от стандартните метод използвани от цветарските магазини. Разплащането може да става дигитално и мигновено.

Чрез тази система освен подобряване на потребителското изживяване, се оптимизира работата на служителите, което им позволява да отделят време и да се съсредоточават върху други аспекти от своите задължения.

1.2. Цел

Целта на настоящата дипломна работа е да разработи интегрирана система за онлайн търговия, обхващаща уеб магазин и мобилно приложение, които да поддържат множество цветарски магазини. Тази система ще предоставя възможност на цветарските магазини да предлагат своите продукти в един централизиран портал, улеснявайки процеса на продажба както за тях, така и за техните клиенти.

Създаването на тази система има за цел да дигитализира и оптимизира търговските операции на цветарските магазини. С внедряването на системата, се стимулира бързината и ефективността на процесите на обработка на поръчки, позволяващи на клиентите да пазаруват от различни магазини в една обединена среда. Системата ще автоматизира множество аспекти на онлайн търговията, като обработка на плащания, следене на доставките и предоставяне на детайлна информация за продуктите.

Също така, разработването на тази система ще допринесе за повишаване на качеството на обслужване на клиентите. Тя ще предоставя удобство и гъвкавост при пазаруване, като позволява на потребителите да разглеждат разнообразни продукти от различни магазини, да сравняват цени и да извършват поръчки с лекота. Въвеждането на тази система ще създаде среда, в която магазините и техните клиенти могат да се възползват максимално от предимствата на онлайн търговията, подобрявайки своето изживяване и оптимизирайки бизнес процесите си.

1.3. Сравнение

В таблица 1.3.1 се извършва сравнителен анализ на няколко системи за онлайн търговия в областта на цветарските магазини. Анализирани са различни критерии, които отразяват функционалностите и възможностите на приложенията. Основните различия между разглежданите системи се отразяват в техните характеристики, които са изложени в съответната колона.

След подробно разглеждане на описаните системи става ясно, че нито една от тях не предоставя функционалност за многобройни магазини от различни вериги, както и възможност за продажба на различни видове продукти. В допълнение, множество от тях не разполагат с мобилни приложения за обработка на поръчки.

Обаче, нашата разработвана система за онлайн търговия е насочена към предоставянето на интегрирана платформа, която дава възможност на множество цветарски магазини да предлагат своите продукти в една обединена среда. Този подход отличава системата, като създава възможност за клиентите да пазаруват от различни вериги магазини и да избират от разнообразни продукти, всичко на едно място.

Също така, системата предлага удобна и ефективна работа чрез мобилно приложение за приемане на поръчки, което допринася за оптимизиране на процеса на обработка и изпълнение на поръчките. В допълнение, нашата система поддържа онлайн плащания, което осигурява удобство и сигурност на клиентите при завършването на транзакциите.

Обобщено, разработваната от нас система съчетава уникални характеристики, които позволяват на множество магазини да оперират в един обединен портал, предлагайки широка гама от продукти и опции на клиентите.

Таблица 1.3.1 Сравнителен анализ на онлайн магазини за продажба на букети

Система	Работят с брой магазини	Градове	Разходи за поддръжка	Мобилно приложение	Онлайн плащания	Лекота на употреба	Скалируемост
butiklilia.com	1	Варна	ДА	НЕ	Да	Средно	НЕ
buketite.net	1	София	ДА	НЕ	Да	Средно	НЕ
e-cvete.com	?	София, Пловдив, Варна, Бургас, Русе	ДА	НЕ	Да	Лесно	НЕ
top-flowers.com	2	София	ДА	НЕ	Да	Средно	НЕ
fairyflower.eu	1	София	ДА	НЕ	Да	Средно	НЕ
Bouquet.bg	Неограничен	Цяла България	Такса при продажба	ДА	Да	Лесно	ДА

Системите *butiklilia.com*, *buketite.net* и *fairyflower.eu* са доста идентични, те представляват сайтове с не лош дизайн и предлагат възможност за продажба на букети и други артикули онлайн. Системата, обект на настоящата дипломна работа, може да продават само букети, защото това е главната цел и функционалност на системата. Възможно е да се добавят и други артикули за в бъдеще.

Системата *top-flowers.com* е изключително подобна на по-горе посочените с разлика, че работи с два обекта на една и съща верига, което все още е ограничаващо спрямо нашата система. Досега споменатите сайтове правят впечатление с добрия си дизайн. В нашия случай дизайна е максимално опростен, за да е лесен за използване.

Системата *e-cvete.com* предлага работа с няколко обекта в различни градове, но те са от една верига и всички те предлагат идентични продукти, това е плюс спрямо другите, защото е много по-ефективно и евтино да се правят доставки в същия град, в който е магазина. Нашата система надгражда тази с това, че обектите не е задължително да са обвързани и всеки един от тях може да продава свой уникален артикул.

Както и повечето от системите *Bouquet.bg* поддържа онлайн плащания, но могат да се правят и поръчки с взимане от магазин и наложен платеж. Системата е много скалируема, защото всеки цветарски магазин може лесно да се регистрира и добавя продукти, които да продава. Освен това разполага и с мобилно приложение, което в реално време приема направените поръчки и уведомява служителите и собствениците с нотификации. Приложението дава удобството да се обработват поръчките в магазина, без да е нужен компютър или лаптоп, а това може да стане дори от личните устройства на служителите.

1.4. Поставяне на задачи за решаване

При разработването на системата за онлайн търговия са необходими ясни и структурирани задачи, които да организират процеса на работа. Тези задачи ще създадат основата за планиране и последователно изпълнение на дипломния проект.

1.4.1. Списък със задачи за изпълнение:

- ❖ Да се определи цялостната идея на дипломната работа:
 - Изследване на нуждите на цветарските магазини;
 - Определяне на целите на системата.
- ❖ Да се определят главните функционалности на системата;
- ❖ Да се определят програмните технологии и средства;
- ❖ Да се дефинират потребителските роли:
 - Идентифициране на потребителските групи и техните права;
 - Определение на функционалности за администратори, партньори, служители и клиенти.
- ❖ Да се избере архитектура на приложението;
- ❖ Да се определят функционалните и нефункционалните изисквания;
- ❖ Да се изготви базата данни и нейната структура;
- ❖ Да се проектира потребителски интерфейс:
 - Да се избере цялостната визия на приложението;
 - Да се проектират отделните дизайни за различните потребителски роли и техните панели;
 - Да се изготвят отделните страници, визуализиращи съответните функционалности;

- ❖ Да се изготви структурата на разработвания проект:
 - Да се реализира връзката с базата данни.
- ❖ Да се реализират отделните функционалности:
 - Да се реализира логиката;
 - Да се реализира връзката между front-end и back-end.
- ❖ Да се направи тестване на системата:
 - Да се проверят дали всички изисквания са изпълнени;
 - Да се провери дали всички функционалности работят коректно;
 - Да се провери дали има нужните съобщения за грешки, успешно завършен процес и известия;
 - Да се провери дали приложението се визуализира коректно на всички резолюции.

Глава 2 Проектиране на системата

2.1. Описание на приложението

Настоящата дипломна работа се фокусира върху разработването на система за онлайн търговия, наречена "bouquet.bg". Основната цел на тази система е да предостави на множество цветарски магазини възможността да продават своите продукти чрез обединена онлайн платформа. Системата включва както уеб магазин, така и мобилно приложение за обработка на поръчки. Тя се стреми да подобри ефективността и удобството на онлайн търговията за цветарите и техните клиенти.

Системата е съставена от сайт и мобилно приложение. Със сайта могат да работят всички потребители, без значение от ролята, но според нея те имат или нямат достъп до различни функционалности. Ролите в нашия случай са четири, те са: администратор, потребител – партньор, потребител - служител и потребител – клиент. Според тях потребителите виждат или не различни бутони и имат или нямат право да достъпват определени страници. Всеки потребител с по-високо ниво на достъп може да вижда и извършва операциите, които потребителите с по-ниско ниво на достъп могат. По този начин се осигурява едно приложение, с което да се работи независимо от ролята на потребителя, което улеснява и процеса на интеграция. С мобилното приложение е по-различно, с него могат да работят само потребители с ниво на достъп служител и нагоре, също така те трябва да имат права да оперират поне по един конкретен обект, за да могат да го използват .

Администраторите могат да оперират със всички възможни функционалности, без значение дали са собственици на даден обект или имат право да работят в него, нещо което не важи за останалите роли. Също така те имат и функционалности специфични само за администраторската роля, като например одобрение на партньори. По този начин администраторите имат пълните права и се избягва директна намеса в базата данни, и се елиминира опасността неквалифицирани лица да имат достъп до чувствителна информация.

Партньорите имат пълните права да управляват собствените си обекти, но не виждат чуждите, като обикновен потребител (клиент) и нямат права върху тях. Партньорите имат правото да добавят и премахват служители, но само за собствените си обекти. Тази операция представлява процедура, при която се въвежда уникален номер на потребител и ако той е с роля клиент, неговата роля се променя на служител. Освен това партньорите могат да добавят нови обекти и стоки към тях. Също така партньорите имат и портфейли, в които те могат да следят баланса си, а той зависи от продажбите на техните магазини. При всяка поръчка платена онлайн се задържа процент, а останалото се начислява в портфейла на собственика на магазина, за който се отнася поръчката.

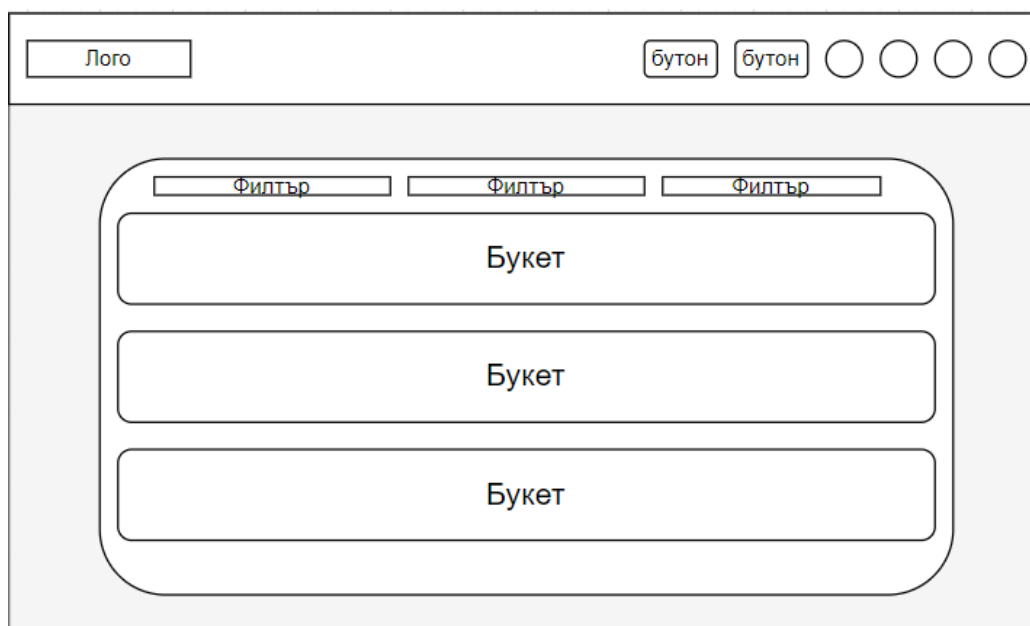
Служителите могат да правят всичко, което и клиентите, също така имат достъп и до функционалности манипулиращи обектите в които те са добавени като служители. Те могат да са служители в няколко обекта едновременно. Специфичните права които имат те са добавяне на продукти, приемане и приключване на продажби. Служителите, които са били добавени и след това премахнати от даден обект се връщат към роля клиент, ако нямат други работни места в системата.

Клиентите имат достъп само до сайта, там те могат да разглеждат стоките по различни категории, да намерят информация на магазините. Могат да добавят продукти и да пазаруват, като указват предпочитанията си за доставка. Могат да плащат онлайн с дебитна или кредитна карта и да следят статуса и историята на поръчки си.

2.2. Описание на потребителски интерфейс

Потребителският интерфейс на bouquet.bg представя модерен и изчистен дизайн в две теми тъмната и светла гама. Използван е като основен акцентиращ цвят зеленият в двете теми, а белият и бежовия като основни в светлата тема и тъмно сивия и тъмно зеления в тъмната, който придават изчистен и семпъл вид и позволяват на изображенията да изпъкват. Това придава на визията изискан, строг и професионален вид, който е предпочитан от потребителите. Контрастът се очаква от цветните изображения, за да създаде по-изчистен и универсален дизайн, приложим и подходящ за всеки цветарски магазин. За сайта и мобилното приложение дизайнът е различен, но запазва същата и подобна гама и типография. По този начин се осигурява визуално разделение между приложенията, но без да се нарушава целостта на системата и дизайнът остава консистентен.

Началната страница на сайта (Фиг. 2.2.1) е и основен екран. На нея потребителите виждат навигационното меню, което се вижда на всички екрани, списък с продукти и филтри по град (в който се намира магазина, който ги продава), цвят и сорт на цветята в букета. Тя е с изчистен и опростен дизайн. За фон се използва изображение с цветя за светлата тема, той се прилага на всички екрани.



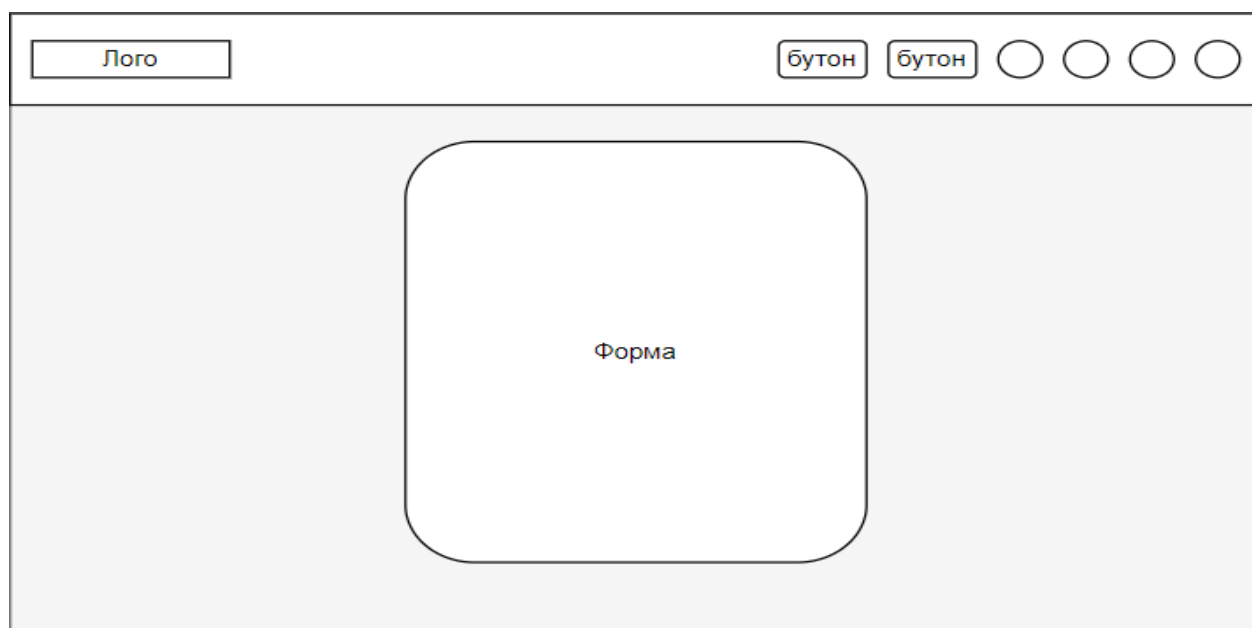
Фигура 2.2.1 Начална страница на bouquet.bg

Началната страница на мобилното (Фиг. 2.2.2) е логин формата за влизане в потребителския профил. Тя е с изчистен и опростен дизайн. Най отгоре се стои логото, а под него е поместена формата за попълване на имейл и парола за влизане в системата. По този начин вниманието на потребителя се насочва единствено върху формата за попълване. Отдолу има линк за регистрация, който препраща към сайта.



Фигура 2.2.2 Начална страница на мобилното приложение

Тъй като регистрацията е на две стъпки, имаме три страници за регистриране на нови потребители. Една обща, с форма за попълване на email и парола и две отделни, според това дали се регистрираме като потребител (с форма за попълване на данните на потребителя) или компания (с форма с допълнителни данни за компанията). Всички те, както и страниците за логин и забравена парола, са с еднаква структура, която представлява форма за попълване на данните, нужни за функцията на съответната страница (Фиг. 2.2.3).



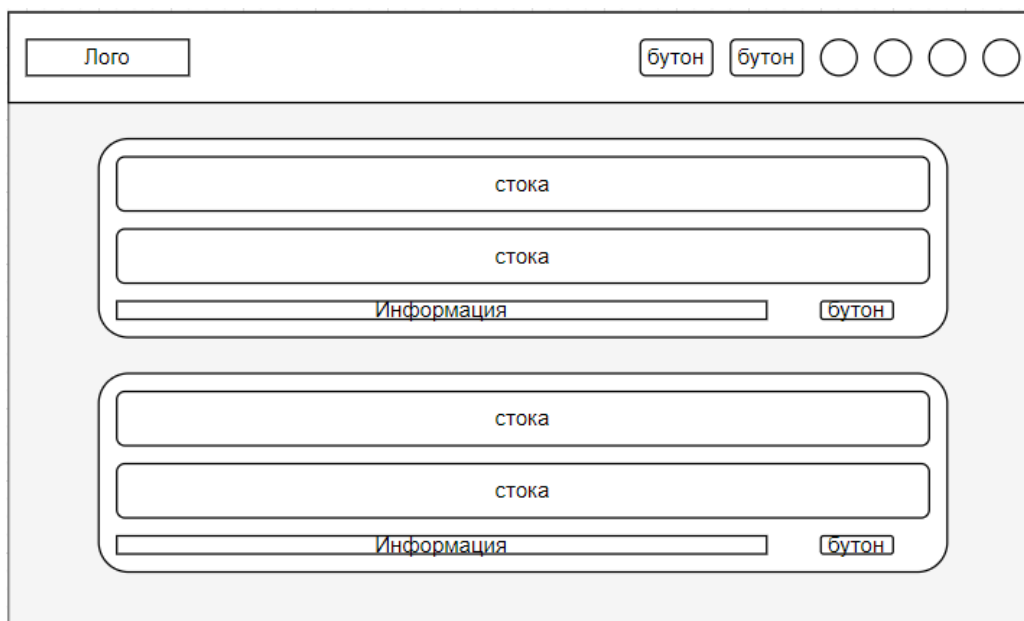
Фигура 2.2.3 Страници за регистрация, логин и забравена парола

Друга основна страница е тази с магазините тя прилича на основната страница, но с разликата, че на нея има карта, на която може да се намерят магазините, те също могат да бъдат селектирани върху картата. На страницата има филтър на градовете, който влияе на картата, както и бутона за филтриране на собствени обекти и добавяне на такива, ако потребителят има тези права (Фиг. 2.2.4).



Фигура 2.2.4 Страница за магазините

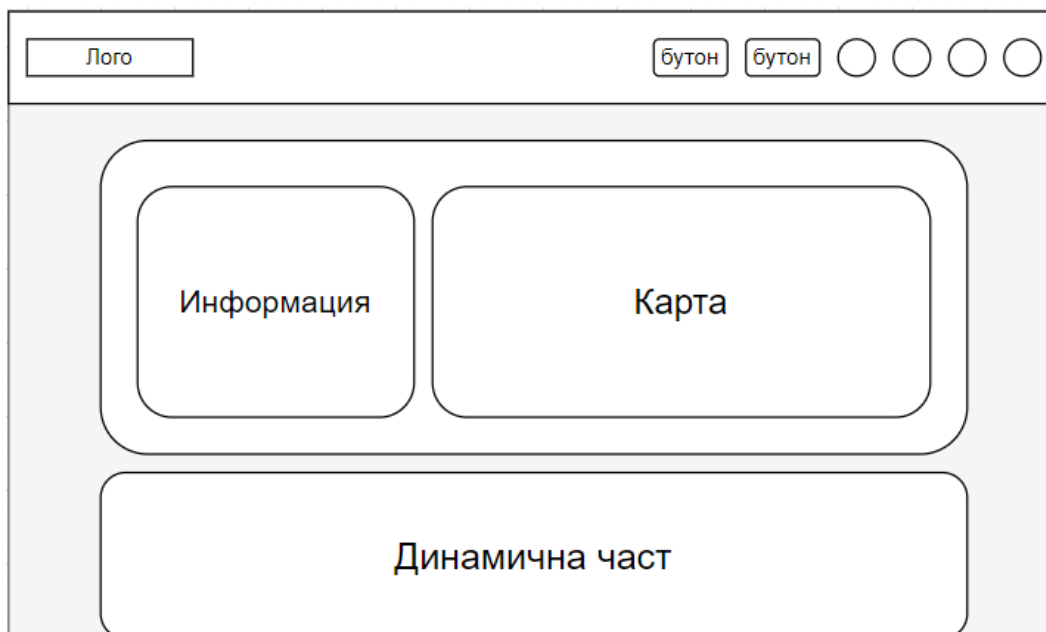
Страницата за кошницата представлява отделни блокове (списък) със стоки, разделени на отделни блокове, по обект от който са. Това се налага, защото не може да се направи поръчка с продукти от различни магазини, затова кошницата е разбита на толкова кошници колкото са различните обекти, от които са добавени стоки. В блокчетата може да открием списък с стоки, информация за тях, както и информация за цялата кошница, като сума, цена на доставка и други (Фиг. 2.2.5).



Фигура 2.2.5 Дизайн на страницата за кошницата

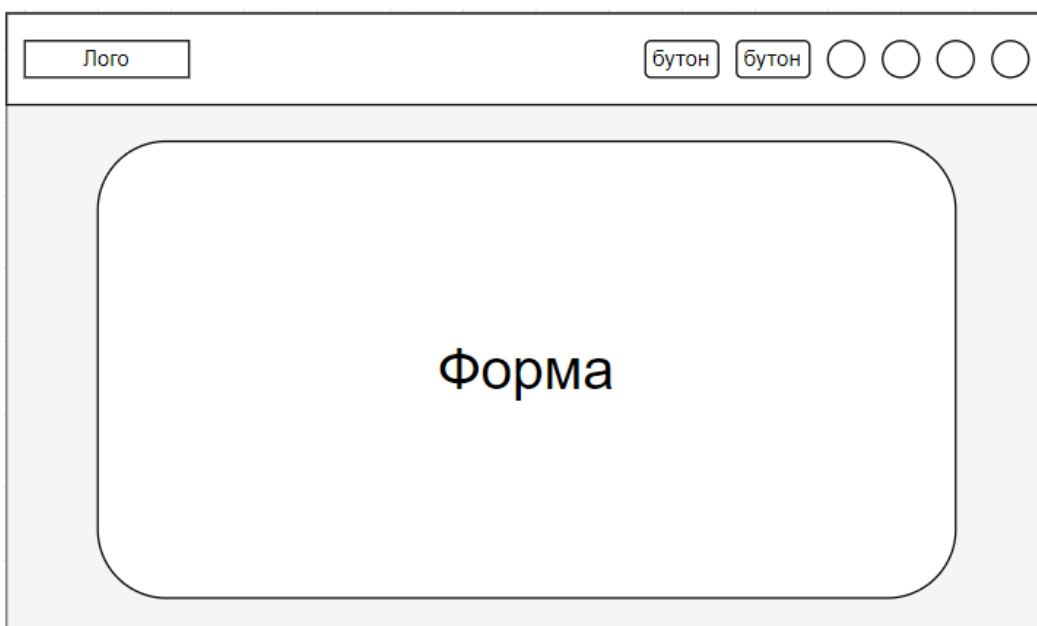
Страницата за конкретен обект (Фиг. 2.2.6), е разделена на две основни чати. Първата от тях само по себе си е разделена на още две. В ляво имаме снимка на обекта и информация за него, от дясната страна е картата с точка на нея, на точната локация на обекта.

Втората част е динамична, клиентите ще виждат списък с продуктите на магазина като на основния екран. Потребителите с повече права ще виждат бутони за функционалности, а списъка с продукти ще може да се сменя със списък с поръчки или списък с служители и форма за добавяне на служители. На тази страница както и другите със списъци в тях, ще може да се скролва .



Фигура 2.2.6 Страница за конкретен обект

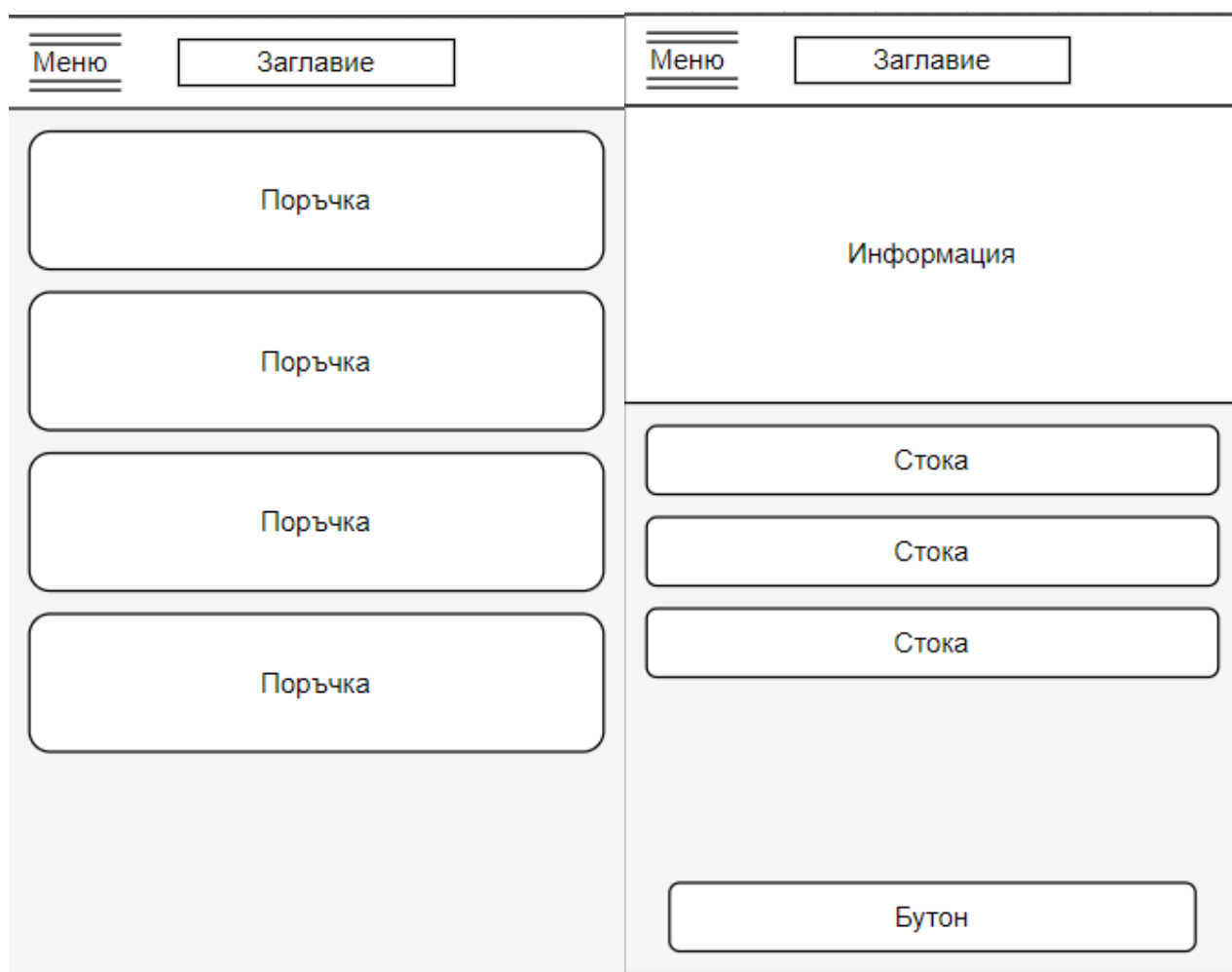
Страниците за добавяне на букет и приключване на операция са доста идентични и представляват голяма форма за попълване на всички необходими данни. Разликите между двете са, че при добавяне на букет ще има възможност за добавяне на изображения, а при завършване на поръчка ще има отметка за това дали поръчката е с доставка или не и според нея формата ще се променя динамично, като скрива или показва данните за доставка (Фиг. 2.2.7).



Фигура 2.2.7 Дизайн на страниците за добавяне на обект и приключване на поръчка

Страницата за обработване на поръчките в мобилното приложение представлява списък от отделни блокчета на поръчки с информация за нея: ид, сума, статус, описание и друго (Фиг. 2.2.8).

Страницата с детайли за поръчката в мобилното приложение е разделена на три части. Първата е за информация за поръчката. Втората е списък с продукти. И най-долу има бутон за промяна на статуса (Фиг. 2.2.9) .



Фигура 2.2.8 Дизайн на страницата за поръчки

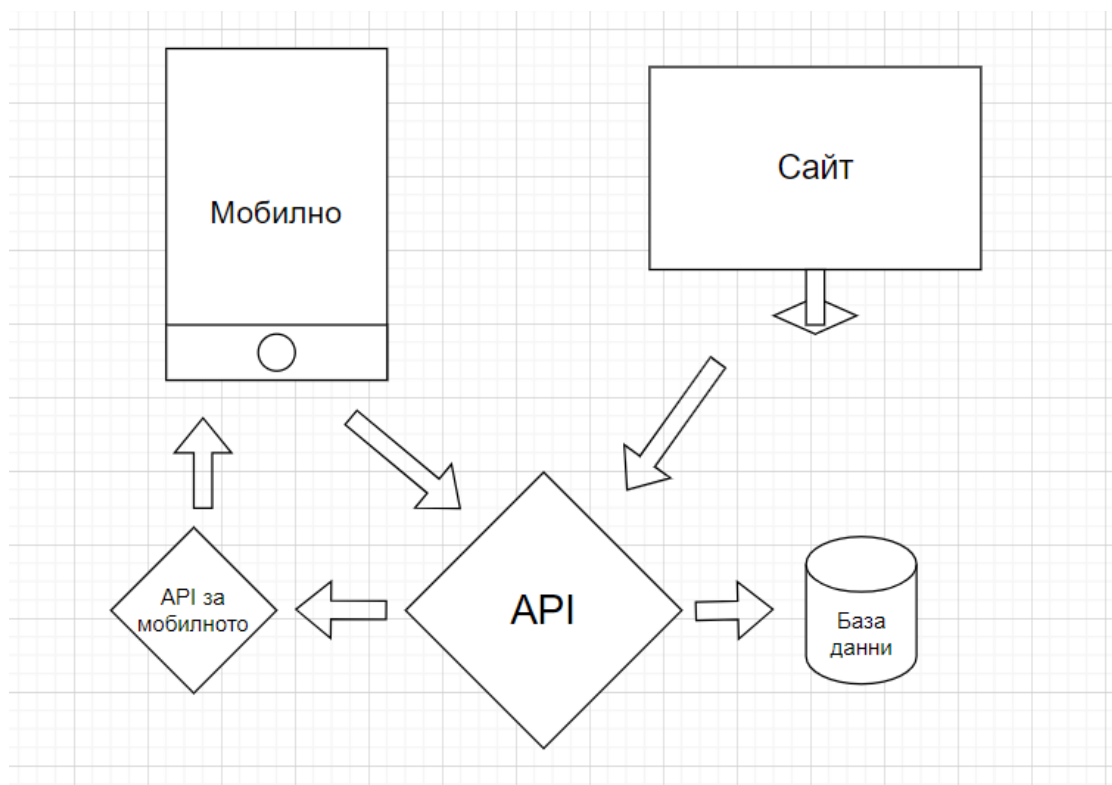
Фигура 2.2.9 Дизайн на страницата за поръчка

При наличие на нови функционалности и нужда от нови страници, те ще бъдат реализирани чрез същите композиции, елементи, последователност и цветова гама за съответния панел.

2.3. Описание на избор на архитектура

2.3.1. Цялостна архитектура

Системата се състои от четири отделни програми, които комуникират една с друга. Един от тях е сайтът (web client) той служи за представяне данните и функционалностите на потребителя по интуитивен и естетически начин. Мобилното приложение има същите функции, но е по-подходящо за целите за които се използва. Сайтът и мобилното приложение комуникират с третата програма, която е сървърна (REST API). Тя е основна и отговаря за цялата бизнес логика и базата данни. Четвърта също е сървърна, нейната задача е да известява конкретни мобилните приложения за ново постъпилите поръчки.



Фигура 2.3.1 Графично представяне на цялостната архитектура

2.3.2. Архитектура на сайта

За разработването на сайта нашият проект използва стандартната архитектурна модел на компоненти, като се базира на библиотеката React. Този модел на архитектура предоставя ефективен начин за структуриране и управление на потребителския интерфейс чрез множество независими компоненти.

Основни компоненти на компонентната архитектура с React:

Компоненти (Components): Всички части на потребителския интерфейс са структурирани в множество компоненти. Всяка компонента представлява изолирана част от интерфейса и може да включва HTML, CSS и JavaScript. Това позволява повторно използване, модулност и ясна организация.

Стейт (State): React предоставя механизми за управление на състоянието на приложението. Стейтът позволява динамичното обновление на компонентите спрямо промените в данните.

Пропс (Props): Пропс представлява механизъм за предаване на данни от един компонент към друг. Това осигурява взаимодействие и комуникация между компонентите.

Предимства на компонентната архитектура с React:

Модулност и повторно използване: Компонентите се разделят на малки, изолирани части, които могат да се използват повторно в различни части на приложението.

Организация и структурираност: Структурата на компонентите позволява ясна организация на потребителския интерфейс и лесно разширение на функционалността.

Лесно управление на състоянието: Стейтът и жизненият цикъл на React позволяват удобно управление на данните и автоматично обновление на интерфейса.

Ефективност: React използва виртуален DOM, което подобрява ефективността на обновлението на интерфейса.

Разработка на модули: Пропс и компонентната структура на React позволяват разработчиците да се фокусират върху отделни части на приложението.

С използването на компонентната архитектура с React, уеб клиентът на проекта осигурява добре организиран, модулен и ефективен начин за представяне и управление на данните и взаимодействие с потребителите.

2.3.3. Архитектура на мобилното приложение

За разработването на мобилното приложение е използвана архитектурата MVVM (Model-View-ViewModel). Тази архитектурна модел разделя компонентите на приложението на три основни елемента - модел, изглед и модел на изгледа (ViewModel). Този подход позволява ясно разграничение между логиката на приложението, представянето на данните и управлението на потребителския интерфейс.

MVVM моделът се адаптира ефективно за мобилното приложение, позволявайки гъвкаво и организирано управление на данните и потребителския интерфейс.

Основни компоненти на архитектурата MVVM:

Модел (Model): Моделът представлява бизнес логиката и данните на приложението. Тук се съхраняват и обработват информацията, която приложението използва. В случая на мобилното приложение за обработка на поръчки, моделът би съдържал информация за продуктите, потребителите, поръчките и други съответни данни.

Изглед (View): Изгледът представлява потребителския интерфейс на приложението. Това е това, което потребителите виждат и с което взаимодействат. Тук се групират елементите за визуализация като бутони, списъци, текстови полета и други.

Модел на изгледа (ViewModel): ViewModel свързва модела и изгледа, като осигурява необходимите данни и функционалности за визуализацията. Това е мястото, където се извършват преобразувания и обработка на данни, които се показват на потребителския интерфейс. В него се съхраняват и логиката за взаимодействие между изгледа и модела.

Предимства на архитектурата MVVM:

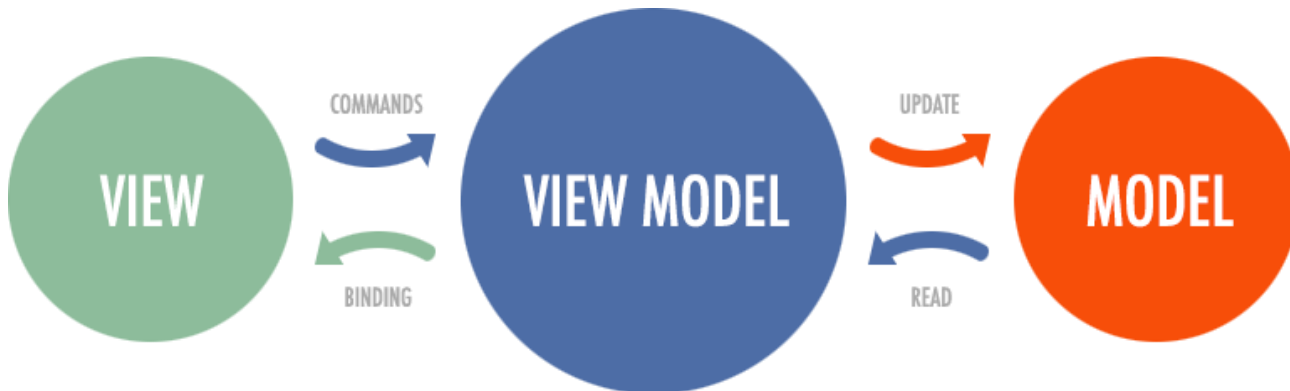
Разделение на отговорности: MVVM разделя ясно отговорностите между различните компоненти на приложението, което улеснява поддръжката и разширяването.

Тестваемост: Поради разделението на бизнес логиката и интерфейлната логика, компонентите се тестват по-лесно и независимо.

Гъвкавост: ViewModel позволява гъвкаво управление на визуализацията и взаимодействието с данните, без да нарушава модела.

Реактивност: MVVM е подходящ за ситуации, в които интерфейсът трябва да се обновява автоматично при промяна на данни (например, динамично обновление на списъци с продукти).

Този избор на архитектура осигурява структурираност, лесно поддържане и разширяемост на мобилното приложение за обработка на поръчки.



Фигура 2.3.2 Графично представяне на MVVM архитектурата

2.3.4. Архитектура на API-а

За реализирането на API (Application Programming Interface) в нашия проект сме използвали архитектурен подход, който включва използването на контролери и сървиси. Този подход позволява създаването на структуриран и модулен интерфейс, чрез който фронтенд клиентите взаимодействат с бизнес логиката на приложението.

Основни компоненти на архитектурата с контролери и сървиси:

Контролери (Controllers): Контролерите представляват прозорецът към външния свят и обработват HTTP заявките от клиентите. Те извличат информацията от заявките и я предават на сървисите за обработка. След получаване на резултата от сървисите, контролерите генерират подходящ HTTP отговор.

Сървиси (Services): Сървисите съдържат бизнес логиката на приложението и осъществяват операциите с данни. Те изпълняват действия като извличане, обновяване, добавяне и изтриване на информация. Сървисите са независими компоненти, които могат да бъдат използвани от различни контролери.

Предимства на архитектурата с контролери и сървиси:

Разделяне на отговорности: Архитектурата позволява разделение на задачите между контролерите и сървисите. Контролерите се фокусират върху взаимодействието с клиентите, докато сървисите се грижат за обработката на данните.

Модулност и преизползване: Сървисите са изградени модулно, което позволява преизползване на бизнес логиката в различни части на приложението.

Управление на данните: Сървисите централизират бизнес логиката, свързана с данните. Това улеснява поддържането на консистентни данни и прави бизнес логиката по-лесно за тестване.

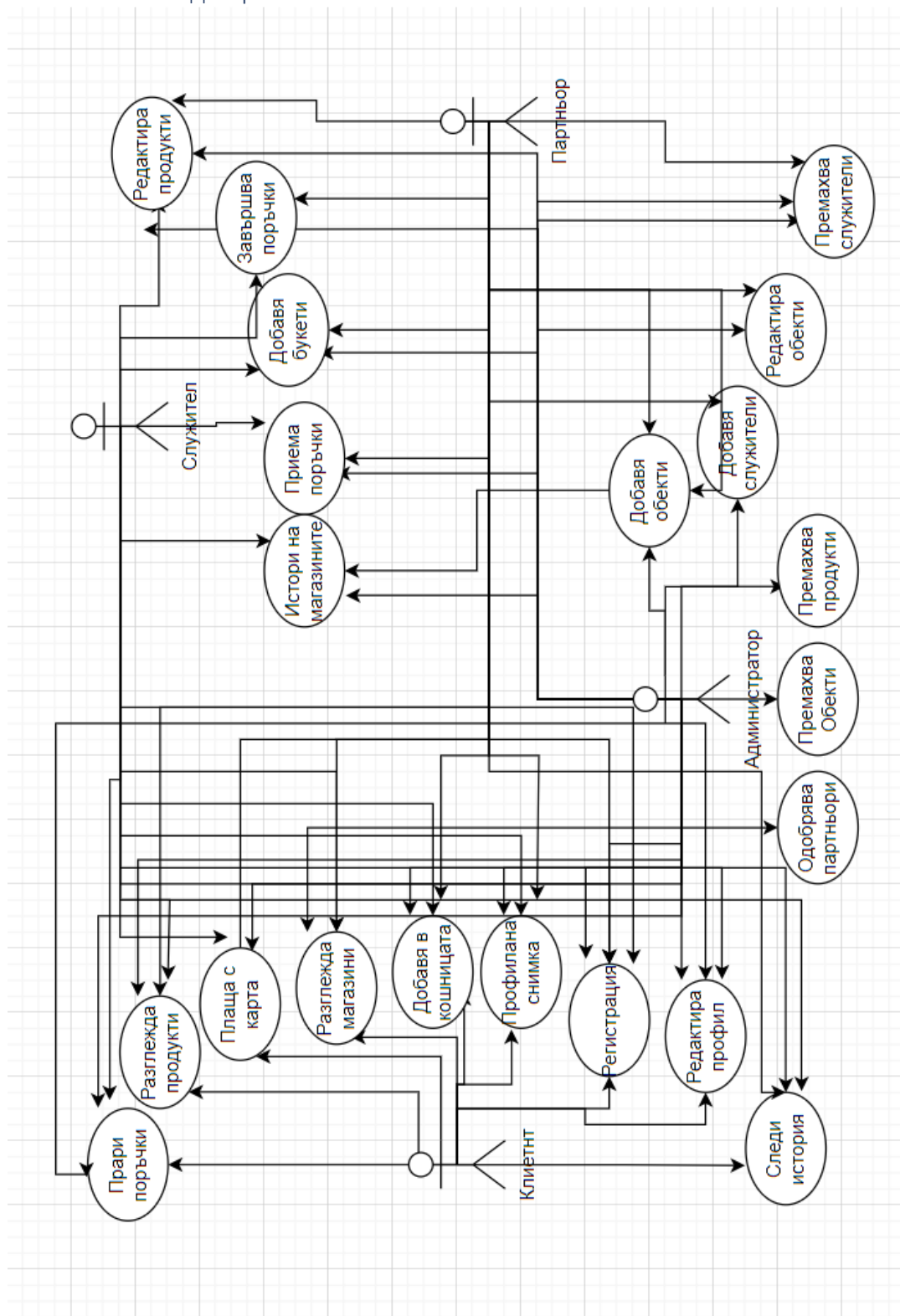
Разширяемост: Подходът позволява добавяне на нови функционалности чрез създаване на нови сървиси и контролери без промяна на вече съществуващия код.

Ясна структура: Разделението на задачите между контролери и сървиси дава ясна структура на приложението и улеснява навигацията и разработката.

С архитектурата с контролери и сървиси нашият API осигурява солидна основа за взаимодействие между клиентската страна и бизнес логиката на приложението, като поддържа модулност, ясна структура и ефективно управление на данните.

2.4. UML диаграми

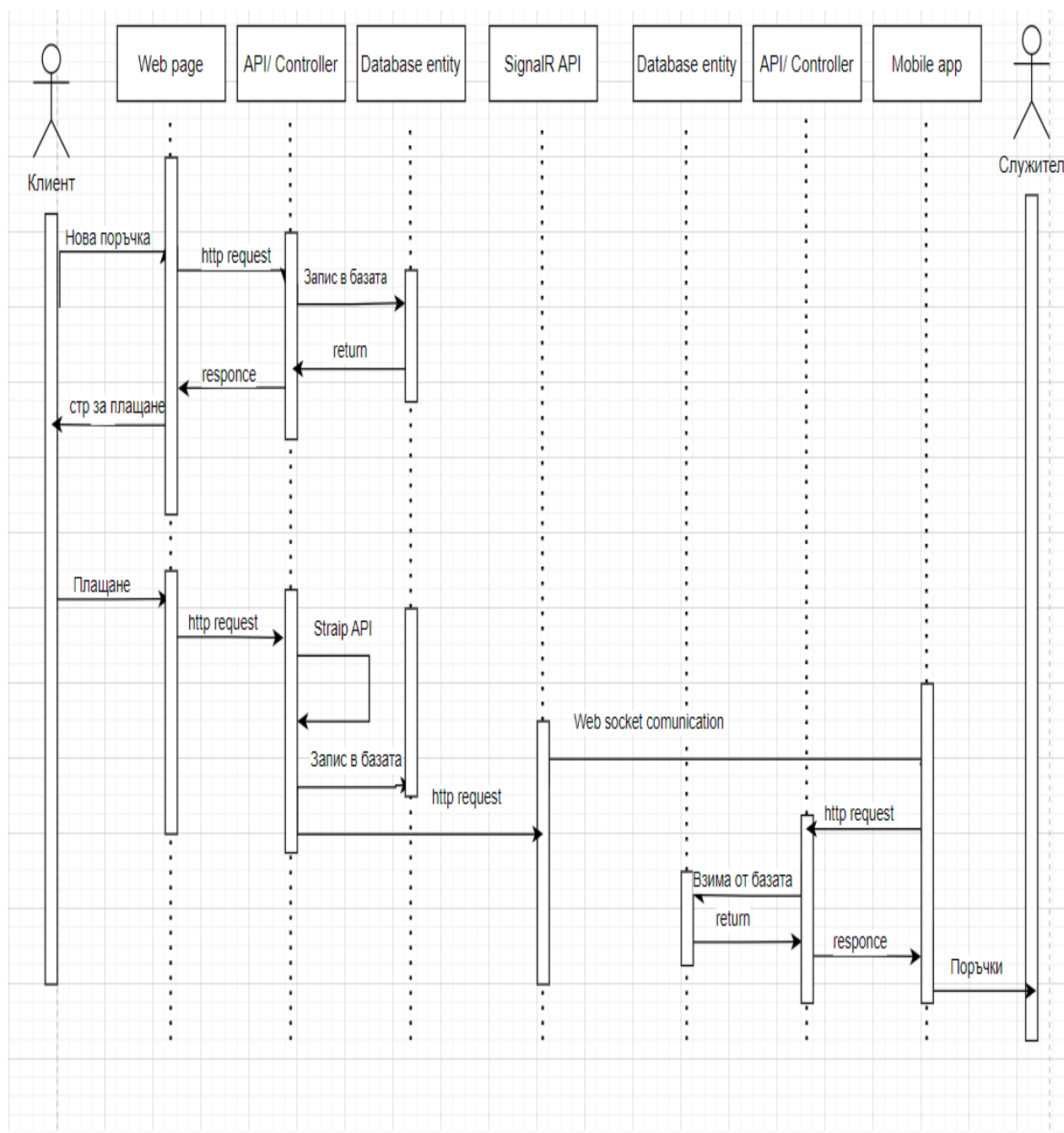
2.4.1. Use-case диаграма



Фигура 2.4.1 Use-case диаграма на системата bouquet.bg

Диаграмата (Фиг. 2.4.1) показва функционалностите на системата за онлайн търговия bouquet.bg и тяхното разпределение по различните потребителски роли. Чрез тази диаграма се разясняват функционалните изисквания на системата. Актьорите разиграват различни сценарии, чрез които се представя кои потребителски роли до кои функционалности имат достъп.

2.4.2. Sequence диаграма



Фигура 2.4.2 Sequence диаграма представяща поръчката

Диаграмата (Фиг. 2.4.2), представена по горе, изобразява функционалността за създаване на поръчка. Тя представя последователно различните етапи, през които се преминава, за да се изпълни изцяло създаването на нова поръчка и как тя стига до служителите. Показана е обработката на данни и от двете страни, участващи в действието – клиент и служител.

2.5. Описание на базата данни

Съхранението и управлението на данни в нашата система се реализират чрез системата за управление на бази данни (СУБД) Microsoft SQL Server (MSSQL). За управление и администриране на базата данни използваме SQL Server Management Studio (SSMS).

Използването на бази данни като средство за съхранение на информация предоставя няколко ключови предимства:

Намаляване на обема на данните: Базите данни позволяват ефективно управление на обема на данните чрез отстраняване на повторения и поддържане на интегритет, коректност и цялостност на информацията.

Интегритет и ограничения: СУБД предоставят механизми за прилагане на ограничения върху стойностите на данните, което гарантира валидността и съответствието им.

Независимост на софтуера от информацията: Съхранението на данните в база данни гарантира, че софтуерните приложения могат да обработват и манипулират данните независимо от конкретната им логика.

Използване на Microsoft SQL Server и SQL Server Management Studio:

За нашата система сме избрали Microsoft SQL Server като СУБД. Този избор се дължи на множеството предимства, които предоставя, включително скалируемост, сигурност и поддръжка на различни видове данни. За администрацията и управлението на базата данни използваме SQL Server Management Studio (SSMS). Този инструмент предоставя мощни възможности за управление, наблюдение и

администриране на базата данни, като позволява създаване на схеми, таблици, индекси, изпълнение на заявки и много други дейности. Изборът на Microsoft SQL Server и SQL Server Management Studio допринася за ефективността и сигурността на нашата система за управление на данни.

Bouquet.bg разполага с единна база данни. Името на базата данни е Bouquet.DB . Тя се състои от 40 таблици, съдържащи цялата информация на приложението: __EFMigrationsHistory, Agreements, AnonymousCustomers, BouquetColor, BouquetFlower, Bouquets, BouquetUserFlowerShop, Cards, CartBouquet, Citys, Colors, DeliveryDetails, Flowers, FlowerShops, OrderCards, Orders, Payments, Pictures, RoleClaims, Roles, ShopConfigs, Transactions, UserClaims, UserInfos, UserLogins, UserRoles, Users, UserTokens, Wallets, AggregatedCounter, Counter, Hash, Job, JobParameter, JobQueue, List, Schema, Server, Set,State. Някои от тях са системни, други са автоматично добавени, като тези на HangFire и тези за Identity. Ще разгледаме само някои от тях.

Таблицата Users съдържа информацията за потребителите. Има следните полета:

- id (първичен ключ): Уникален идентификационен номер за потребителя;
- UniqueNumber: Уникален номер за потребителя;
- ProfilePictureDataUrl: Стрингово поле, което съхранява URL адрес за профилна снимка на потребителя;
- IsDeleted: Флаг, който показва дали потребителят е изтрит;
- DeletedOn: Дата на изтриване на потребителя;
- IsActive: Флаг, който показва дали профилът на потребителя е активен;
- RefreshToken: Токен за опресняване на достъпа;
- RefreshTokenExpiryTime: Дата и час на изтичане на валидността на токена за опресняване;
- CreatedBy: Потребител, създал записа;
- CreatedOn: Дата на създаване на записа;
- LastModifiedBy: Потребител, който е направил последната промяна;
- LastModifiedOn: Дата на последна промяна;

- WalletID: Уникален идентификационен номер за портфейла на потребителя;
- CutomerId: Уникален идентификационен номер за клиента (ако е приложимо);

Таблицата FlowerShops съдържа информацията за обектите. Тя съдържа следните полета:

- Id: Уникален идентификационен номер за магазина;
- Latitude: Географска ширина на магазина;
- Longitude: Географска дължина на магазина;
- Name: Име на магазина;
- Address: Адрес на магазина;
- PictureDataUrl: URL адрес към изображение на магазина;
- CityID: Уникален идентификационен номер на града, в който се намира магазинът;
- OwnerID: Уникален идентификационен номер на собственика на магазина;
- ShopConfigID: Уникален идентификационен номер на конфигурацията на магазина;
- AgreementID: Уникален идентификационен номер на споразумението, свързано с магазина;
- Status: Статус на магазина;

Таблицата Orders съдържа информацията за поръчките. Тя съдържа следните полета:

- id - Уникален идентификационен номер за поръчката;
- Price: Цена на поръчката;
- Description: Описание на поръчката;
- FlowerShopID: Уникален идентификационен номер на магазина, от който е направена поръчката;
- UserID: Уникален идентификационен номер на потребителя, който е направил поръчката;
- AnonymousCustomerID: Уникален идентификационен номер на анонимния потребител, ако е такъв;

- DeliveryDetailsID: Уникален идентификацион номер на данните за доставката на поръчката;
- OrderCartID: Уникален идентификацион номер на количката, свързана с поръчката;
- Status: Статус на поръчката;

2.5.1. ER диаграма на базата данни



Фигура 2.5.1 ER диаграма на базата данни Bouquet.DB

Релациите между таблиците в базата са доста, затова ще дам по един пример от всеки един тип.

Една от връзките между таблиците Users и FlowerShops е M:M (Много към много). Това означава, че повече от един обекти от едната се асоциират с повече от един обекти от другата таблица и обратно. Конкретно в текущата база, това означава, че един потребител може да има няколко работни места (обекта), както и една обект може да има няколко служителя (потребители).

Връзката между таблиците Bouquets и Pictures е 1:M (Едно към много). Това означава, че един обект от Bouquets може да бъде асоцииран с повече от един обекти от Pictures. Един обект от Pictures се асоциира с точно един обект от Bouquets. Тоест една букет може да има много снимки, но една снимка може да бъде само на един букет.

Връзката между таблиците FlowerShops и ShopConfigs е 1:1 (Едно към едно). Това означава, че един обект от FlowerShops може да бъде асоцииран само с един обекти от ShopConfigs. Един обект от ShopConfigs се асоциира с точно един обект от FlowerShops. Тоест една магазин може да има само едни настройки и едни настройки може да се отнасят само за един магазин.

2.6. Описание на технологии и програмни средства

За разработването на системата за онлайн търговия са използване следните технологии: ASP .NET, React, Xamarin, SignalR и MSSQL.

2.6.1. Разработване на сайта

За разработването на сайта е използван React, Note.js и JavaScript.

JavaScript е програмен език за уеб разработка, който позволява създаването на интерактивни уеб страници и приложения. Той добавя динамичност и функционалности към уеб страниците, позволявайки взаимодействие с потребителите и манипулация на съдържанието на страницата.

React е съвременна JavaScript библиотека за разработка на потребителски интерфейси. Тя е разработена и поддържана от Facebook и има голяма популярност и използваност в съвременната уеб разработка. React позволява построяването на ефективни, мащабируеми и динамични интерфейси чрез компонентен подход.

Основни характеристики и предимства на React:

Компонентен подход: React придържа към компонентен подход, който позволява разделянето на интерфейса на малки, независими компоненти. Това прави кода по-преизползваем и по-лесно за поддръжка.

Виртуален DOM: React използва виртуален DOM (Document Object Model), който оптимизира процеса на актуализация на интерфейса. Вместо да променя реалния DOM директно, React създава виртуално дърво от елементи, което след това се актуализира с реалния DOM само с необходимите промени. Това подобрява бързодействието и ефективността на приложението.

Едностраничен поток на данни: Данните в React приложението се движат в една посока - от родителски компонент към деца. Това гарантира по-лесно следене на данните и избягване на нежелани конфликти.

Компонентни жизнени цикли: Всеки компонент в React има жизнен цикъл, който се състои от различни фази - създаване, актуализация и унищожаване. Това позволява на разработчиците да контролират поведението на компонентите в различни моменти.

JSX синтаксис: JSX е разширение на JavaScript, което позволява писането на HTML подобен код в JavaScript файлове. Това прави кода по-четим и по-лесен за разбиране.

Мащабируемост: React е проектиран така, че да бъде лесно мащабируем за големи и сложни приложения. Можете да създавате множество компоненти и ги съчетавате в по-големи структури.

Голяма общност и екосистема: React има огромна общност от разработчици и множество библиотеки и инструменти, които подпомагат разработката на приложения. Това включва например React Router за маршрутизация, Redux за управление на състоянието и много други.

React е мощна технология, която предоставя средства за разработка на модерни и динамични уеб приложения. Неговата гъвкавост и обширна общност го правят предпочитан избор за разработчици в сферата на уеб технологиите.

Node.js е платформа за изпълнение на JavaScript код извън браузъра. В контекста на React, Node.js играе важна роля като среда за разработка и сървърно изпълнение на приложения. Вотът е кратко описание защо Node.js е необходим за React, защото играе роля на:

Сървърна среда: React приложенията се изпълняват в браузъра на клиентската страна. Обаче, за по-сложни приложения може да бъде необходимо сървърно изпълнение, например за генериране на HTML на сървъра преди доставянето му на клиента. Node.js предоставя сървърна среда, която позволява на React приложенията да бъдат изпълнявани на сървъра.

Инструменти и пакети: Node.js се доставя с npm (Node Package Manager), който е репозиторий за хиляди JavaScript пакети и инструменти. Това прави Node.js отличен избор за управление на зависимости и интегриране на инструменти в разработката на React приложения.

Създаване на сървърни приложения: С Node.js можете лесно да създадете сървърни приложения, като например API за взаимодействие между клиентската и сървърната част на React приложението.

Среда за разработка: Node.js предоставя среда, в която може да се изпълняват различни сценарии, като сборка на приложения, автоматизирани тестове, разработка на инструменти за управление на кода и др.

Скорост и ефективност: Node.js е известен със своята висока скорост на изпълнение и ниска консумация на ресурси. Това го прави подходящ за разработка и изпълнение на съвременни уеб приложения, включително такива, базирани на React.

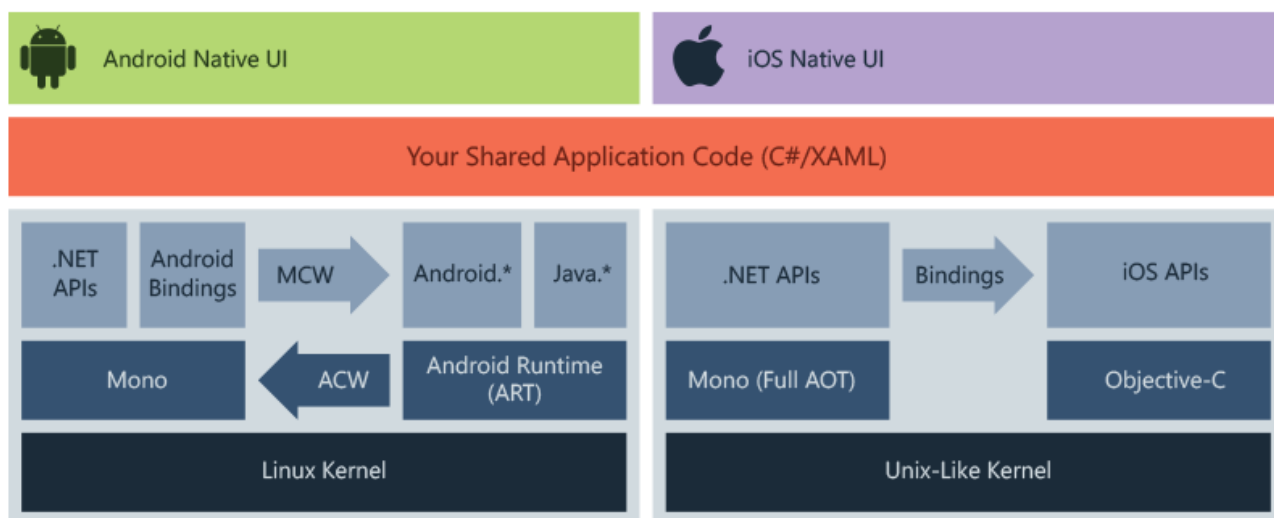
В заключение, Node.js е необходим за React, тъй като предоставя средата и инструментите, които подпомагат разработката, сървърното изпълнение и оптимизацията на приложенията, базирани на тази библиотека..

2.6.2. Разработване на мобилното приложение

За разработването на мобилното приложение са използвана технологията за на Microsoft, а Xamarin.

Xamarin е платформа с отворен код за изграждане на модерни и производителни приложения за iOS, Android и Windows с .NET. Xamarin е абстракционен слой, който управлява комуникацията на споделен код с основния код на платформата, тоест бизнес логиката се пише само веднъж и може да се преизползва за iOS, Android и Windows. Xamarin работи в управлявана среда, която предоставя удобства като разпределяне и освобождаването на памет.

Xamarin позволява на разработчиците да споделят средно 90% от своите приложения между платформи. Този модел позволява на разработчиците да напишат цялата си бизнес логика на един език (или да използват повторно съществуващ код на приложение), но да постигнат собствена производителност, външен вид и усещане на всяка платформа, както е и в нашия случай.



Фигура 2.6.1 Диаграма как работи Xamarin

Диаграмата показва цялостната архитектура на междуплатформено Xamarin приложение. Xamarin позволява да създава естествен потребителски интерфейс на всяка платформа и да се пише бизнес логика на C#, която се споделя между платформи. В повечето случаи 80% от кода на приложението може да се споделя с помощта на Xamarin.

Xamarin е изграден върху .NET, който автоматично обработва задачи като разпределяне на памет, събиране на отпадъци и оперативна съвместимост с базовите платформи.

2.6.3. Разработване на API-а

За разработването на API приложението се използва фреймуоркът ASP.NET API.

.NET е платформа за софтуерно приложение, разработена от Microsoft. Тя предоставя средствата и инфраструктурата за създаване, развитие и изпълнение на разнообразни видове приложения - от десктоп до уеб и мобилни. В основата на .NET стои концепцията за управляем код (managed code), която поддържа автоматично управление на паметта и висока сигурност. .NET се състои от няколко компонента:

Common Language Runtime (CLR): CLR е основният изпълнител на .NET, който управлява изпълнението на програмите. Той предоставя автоматично управление на паметта (сбирателя на боклука) и гарантира сигурността на кода чрез провеждане на валидации и изолация.

Base Class Library (BCL): BCL е библиотеката с основни класове и типове, които предоставят готови функционалности, като работа с файлове, мрежи, данни, текст и др. Това помага на разработчиците да изградят приложения по-бързо, като използват вече създадени компоненти.

Languages: .NET поддържа множество програмни езици, като C# (него използваме за разработката на тази дипломна работа), VB.NET, F#, C++/CLI и други. C# е един от най-широко използваните езици в .NET и се разработва специално за платформата.

Development Tools: Microsoft Visual Studio е основната интегрирана среда за разработка (IDE) за .NET. Тя предоставя редактор на код, дизайнер на потребителски интерфейс, дебъгер и други инструменти за разработка.

ASP.NET: ASP.NET е платформа за създаване на уеб приложения и услуги. Тя предоставя средства за изграждане на динамични уеб страници, извличане на данни от бази данни, управление на сесии и други уеб функционалности. Тази точка ще бъде разисквана подробно по-късно.

.NET Core и .NET Framework: .NET Core е версия на .NET, която е отворен код и се фокусира върху макарото-служебни и уеб приложения, които могат да бъдат използвани на различни операционни системи. .NET Framework е версия на .NET, която е по-подходяща за десктоп, уеб и сървърни приложения, но е ограничена до Windows операционни системи.

.NET платформата има голям принос към разработката на съвременни приложения, като предоставя мощни инструменти за разработка, гъвкавост в избора на език и възможност за създаване на разнообразни видове приложения в различни области.

ASP.NET Web API е фреймуърк, който позволява разработка на уеб услуги (API - Application Programming Interface) чрез използване на платформата ASP.NET. Тези API могат да бъдат достъпвани от различни клиенти, като уеб приложения, мобилни приложения, устройства IoT (Internet of Things) и други. Ето някои ключови аспекти на ASP.NET Web API:

RESTful архитектура: ASP.NET Web API насърчава разработката на API, които следват REST (Representational State Transfer) принципите. Това позволява лесна интеграция и обмяна на данни между клиентите и сървъра.

HTTP протокол: Фреймуъркът използва HTTP методи като GET, POST, PUT и DELETE за взаимодействие с ресурсите на сървъра. Това дава възможност за създаване на CRUD (Create, Read, Update, Delete) операции за данните.

Маршрутизация: ASP.NET Web API използва маршрутизация, която асоциира HTTP заявките с конкретни методи в контролерите. Това позволява динамично извличане на данни и извършване на действия според заявките.

Сериализация: Фреймуъркът автоматично сериализира и десериализира данни между обекти в паметта и формати като JSON или XML, които могат да бъдат използвани за обмяна на информация.

Поддръжка на различни формати: ASP.NET Web API поддържа различни формати на съобщенията, като JSON, XML и други. Това позволява клиентите да избират предпочитания формат за обмяна на данни.

Сигурност и автентикация: Фреймуъркът предоставя вградена поддръжка за сигурност и автентикация, което позволява защита на API-то и управление на достъпа до данни.

Тестване и документация: ASP.NET Web API предоставя инструменти за тестване и документация на API-то. Това прави процеса на разработка, тестване и използване на API-та по-лесен и удобен.

Интеграция със съществуващи ASP.NET приложения: ASP.NET Web API може да бъде интегриран в съществуващи ASP.NET приложения или да бъде използван като отделен проект за създаване на API.

Това са само някои от ключовите аспекти на ASP.NET Web API. Този фреймуърк предоставя мощни инструменти за разработка на уеб услуги и API, които са важен елемент за създаването на съвременни приложения.

2.6.4. Разработване на API-а за известяване на мобилните устройства

За разработването на API-а за известяване на мобилното приложение използва същата технология като предишния API, но с разликата че използва също и SignalR.

SignalR е библиотека, която позволява реално време (real-time) комуникация между уеб приложенията и сървър. Тя използва различни технологии и транспортни протоколи, за да осигури двупосочна комуникация в реално време.

С SignalR, сървърът може да изпраща данни и съобщения към клиентите, а клиентите могат да извикват методи на сървър. Това позволява на приложенията да се актуализират и реагират на събития в реално време, без да се налага клиентите постоянно да заявяват сървър за нови данни.

SignalR използва различни методи за комуникация в зависимост от наличността на поддържани транспортни протоколи, като WebSockets, Server-Sent Events (SSE), Long Polling и други. Това позволява на SignalR да бъде гъвкава и да поддържа комуникация дори в случай на ограничения на клиентската или сървърната страна.

Със сигурността и надеждността на връзката, SignalR управлява автоматично възстановяването на връзката след временни смущения и гарантира доставката на съобщенията. Това прави SignalR подходящ за приложения, които изискват реално време обмяна на данни, като чат системи, споделяне на данни и игри.

2.6.5. Програмни среди

За разработка на цялата система са използвани две среди Visual Studio и Visual Studio Code (VS Code).

Visual Studio: Това е интегрирана среда за разработка (IDE) от Microsoft, която предоставя множество инструменти и функционалности за създаване на разнообразни приложения, включително уеб, десктоп, мобилни и облачни приложения. Visual Studio предоставя интуитивен интерфейс, богата поддръжка на езици за програмиране и интеграция с различни инструменти и услуги.

Visual Studio Code (VS Code): Това е лека и мощна програмна среда, предназначена за разработка на разнообразни приложения. Въпреки че се нарича "Code", тя предоставя богат набор от функции, разширения и инструменти за работа с различни езици за програмиране, уеб технологии и среди. VS Code е известна със своята гъвкавост, бързина и поддръжка на разширения, което позволява на разработчиците да я настроят според своите нужди.

Тези програмни среди предоставят поддръжка за разработка на различни видове приложения, като Visual Studio е използвана за създаване на мобилното приложения и API-те, докато Visual Studio Code е предпочитана за разработка на уеб приложения, нея съм използвал за създаване на сайта. Тя също допълнително може да бъде настроена за разнообразни сценарии чрез разширения.

2.7. Сигурност

Системата за онлайн търговия bouquet.bg разполага с чувствителна информация за своите потребители, като частична информация за дебитни и кредитни карти, лични данни и други. По тази причина е необходимо данните да бъдат защитени по най-добрия възможен начин. За да няма изтичане на данни съм използвал удостоверяване и упълномощаване които са вградени в ASP.NET, които са широко използвани ,и са се доказали като надежна защита.

Authentication с JwtBearer: JwtBearer (JSON Web Token Bearer) е механизъм за аутентикация, който използва JSON Web Tokens (JWT) за предоставяне на сигурни и уникални токени за потребителите след успешна аутентикация. Тези токени се подписват и криптират и могат да съдържат информация за потребителя и неговите разрешения. Приложението генерира и подписва JWT при успешна аутентикация на потребителите, а те след това се предават от клиента (например мобилното приложение или уебсайт) при всяка заявка, за да се докаже тяхната идентичност. Сървърът може да провери валидността на токена и да идентифицира потребителя чрез него.

Authorization: След като потребителите са успешно аутентикирани, се прилага механизъм за разрешения (authorization), който контролира какви действия могат да се изпълняват от даден потребител. Това осигурява защита на ресурсите на приложението, като се гарантира, че само упълномощени потребители могат да изпълняват определени операции. В случай на API, разрешенията се контролират на база на ролята или други критерии, зададени в JWT токена. Това позволява да се осигури гъвкавост и безопасност на приложението, като се предотвратява неоторизиран достъп до чувствителни данни и функционалности.

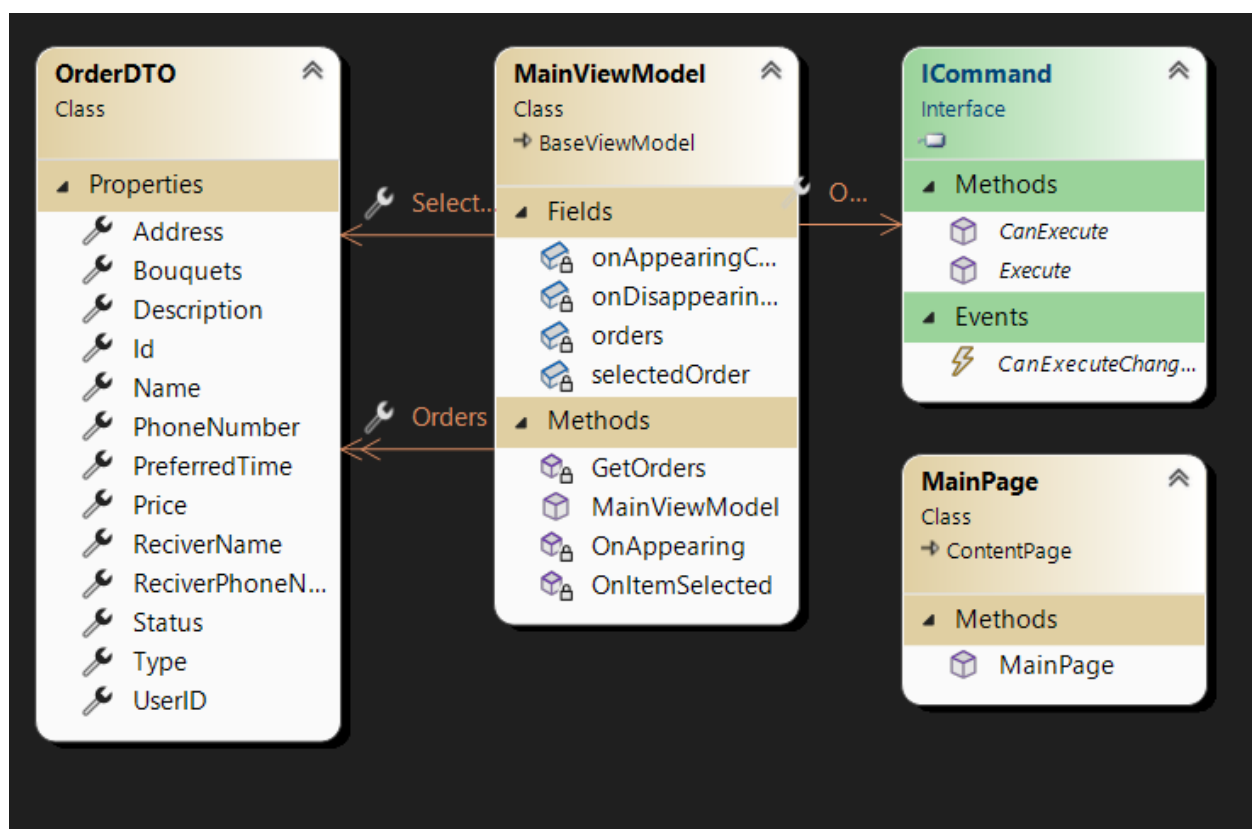
Използването на Authentication с JwtBearer и Authorization съчетава аутентикацията с установяване на идентичността на потребителите и разрешенията за достъп до ресурсите. Това създава по-сигурно и защитено приложение, като гарантира, че само упълномощени потребители могат да взаимодействат с него.

Глава 3 Програмна реализация

В настоящата глава ще се разгледат класовете, използвани за реализацията на системата, методите и връзките между тях.

3.1. Класови диаграми

Фигура 3.1.1 представя класовата диаграма на главния екран на моблното приложение, следваща MVVM архитектурата.



Фигура 3.1.1 Класова диаграма на главния екран на мобилното приложение

Фигура 3.1.2 представя класовата диаграма на моделите, използвани в bouquet.bg.



Фигура 3.1.2 Класова диаграма на моделите

3.2. Реализация на автентификация и авторизация

Добавянето на идентичност и JWT аутентикация в приложението Bouquet е извършено по следния начин:

AddIdentity: Този метод се използва за добавяне на идентичност в приложението. Той настройва стандартната идентичност в .NET Core и връзва потребителите и ролята с контекста на базата данни.

AddJWTAuthentication: Този метод се използва за добавяне на JWT аутентикация. Той конфигурира JWT аутентикацията, включително валидационните параметри за токена, включително издателя (issuer), аудиторията (audience), времето на валидност и др. Също така, добавя се JWT авторизацията към сервисите на приложението, така че контролерите могат да използват атрибути за авторизация.

След добавянето на идентичност и JWT аутентикацията, приложението е готово да приема и валидира JWT токени, които могат да бъдат използвани за аутентикация и авторизация на потребителите.

Следните контролери и класове имат за цел да управляват и поддържат аутентикация и авторизация в приложението Bouquet.bg:

AuthenticationController

Login([FromBody] LoginModel loginModel) - Този метод се използва за вход в приложението. Потребителските данни се подават чрез loginModel. При успешен вход се генерира JWT токен и връща към клиента.

Register([FromBody] RegisterInfo registerModel) - Този метод се използва за регистрация на нов потребител. Данните за регистрацията се подават чрез registerModel. При успешна регистрация се създава потребителски профил и се изпраща имейл за потвърждение.

RefreshToken(JWTTokenModel tokenModel) - Този метод се използва за обновяване на JWT токена. При успешна актуализация на токена, новият токен се връща към клиента.

`ConfirmEmail([FromQuery] string userId, [FromQuery] string code)` - Този метод се използва за потвърждение на имейла на потребителя след регистрация. При успешно потвърждение, потребителят се пренасочва към страницата за вход.

PermissionsController

`GetPermissions()` - Този метод се използва за извличане на правата, които потребителят има. Проверява се валидността на JWT токена и след това се връщат правата на потребителя.

`AddPermission([FromBody] AddPermissionRequest request)` - Този метод се използва за добавяне на ново право към роля. Заявката за добавяне на право се подава чрез request. Проверява се дали потребителят има права да извърши тази операция и след това се добавя правото.

`RemovePermission([FromBody] RemovePermissionRequest request)` - Този метод се използва за премахване на право от роля. Заявката за премахване на право се подава чрез request. Проверява се дали потребителят има права да извърши тази операция и след това се премахва правото.

RoleClaimsHandler

Този клас служи за управление на авторизационни правила и се използва от PermissionsController. Той проверява дали потребителят, който изисква дадено право, има права да го направи. Ако потребителят има правото, RoleClaimsHandler разрешава достъпа, в противен случай - го отказва.

Тези контролери и класове обаче не само гарантират, че потребителите имат достъп до приложението и неговите функционалности, но и предоставят начин за управление на ролите и разрешенията в приложението. Това ги прави съществена част от сигурността и функционалността на Bouquet.bg.

3.3. Реализация на работа с продукти

Следните контролери и класове имат за цел да управляват и поддържат работата с продукти в системата Bouquet.bg:

BouquetController:

Този контролер служи като част от API на вашето приложение и приема HTTP заявки свързани с букетите.

Действието **GetBouquets** позволява на клиентите да получават списък с букети. Може да филтрират букетите по град и магазин.

Действието **AddBouquet** позволява на клиентите да добавят нов букет към системата.

Действието **EditBouquet** позволява на клиентите да редактира съществуващ букет в системата.

Действието **UploadBouquetPictures** дава възможност за качване на снимки към съществуващ букет.

BouquetService:

Тази сървиз предоставя бизнес логика за операциите с букети.

GetBouquetsAsync методът връща списък с букети, които са налични в определен град или магазин. Това може да включва данни за цветовете, цветовете и други атрибути на букетите.

AddBouquetsAsync методът се използва за добавяне на нов букет към системата. Той приема информация за новия букет, включително снимки, цветовете и цветовете, и го записва в базата данни.

EditBouquetAsync методът се използва за редакция на букет към. Той приема информация за редактирания букет, и го променя в базата данни.

UploadPictureAsync методът качва снимки към съществуващ букет. Той генерира уникално име за снимката, я записва на файлова система и обновява букета с новата снимка.

Home Component:

Този компонент представя началната страница на приложението, където потребителите могат да разглеждат списък с налични букети.

Потребителите могат да филтрират букетите по град, цветове и цветя, за да видят само тези, които съответстват на техните предпочитания.

Методът `fetchBouquets` се извиква, за да зареди букетите от сървъра при първоначалното зареждане на страницата или след промяна в избрания град.

Bouquets Component:

Този компонент показва индивидуални букети на потребителите.

Всяка картина на букета се показва във въртящ се слайдшоу (carousel), което позволява на потребителите да виждат различни снимки на букета.

Компонентът показва основни данни за букета, като име, описание, брой цветя, височина и цена.

Потребителите могат да добавят букети към количката си за покупка или да изпратят заявка за поръчка на букета.

Ако няма налични букети, се показва съобщение, че няма букети налични.

3.4. Реализация на работа с обекти

Следните контролери и класове имат за цел да управляват и поддържат работата с продукти в системата Bouquet.bg:

FlowerShopController:

Този контролер е отговорен за обработката на HTTP заявки, свързани с магазините за цветя.

`GetObjects` методът връща информация за магазините за цветя, като може да приеме опционален параметър `shopID`, който филтрира резултата до конкретен магазин.

`GetOwnedObjects` и `GetWorkPlacesObjects` методите връщат магазините, на които потребителят има определени права (например, право да работи в тях или право да управлява поръчки). Методите използват атрибута `[Authorize]`, за да изискват аутентикация преди достъп до тези ресурси.

AddObject и **AddWorker** методите са отговорни за добавянето на нови магазини и служители. Те изискват специфични разрешения (потребителски роли) за изпълнение.

UploadObjectPicture методът позволява качването на снимки за конкретен магазин.

FlowerShopService:

Тази сървиз съдържа бизнес логиката, свързана с магазините за цветя.

GetShopsAsync, **GetShopAsync**, **GetOwnedShopsAsync** и **GetWorkPlacesAsync** методите извличат информация за магазините на основата на различни критерии и връщат ги в удобен за употреба формат.

GetWorkersAsync методът връща списък със служителите на даден магазин.

GetPictureUrlAsync методът връща URL адрес за снимката на даден магазин.

AddShopAsync и **AddWorkerAsync** методите добавят нови магазини и служители, като следват бизнес правилата и проверки за разрешения.

UploadPictureAsync методът качва снимка за даден магазин.

RemoveWorkerAsync методът премахва служител от магазин и извършва съответни проверки.

Shops Component:

Този компонент представя информация за магазини с цветя и предоставя възможности за филтриране и навигация.

Показва списък с магазини с цветя.

Позволява на потребителите да филтрират магазините по град и конкретен магазин.

Визуализира броя на магазините, както и информация за града и магазина.

Позволява на аутентифицираните потребители с определени права да видят бутони за управление на магазините, включително бутон "Добави магазин".

Използва картата, за да покаже разположението на магазините.

Shop Component:

Този компонент представя подробности за един магазин с цветя и включва различни раздели за информация и функционалности.

Показва основна информация за магазина, като име, адрес и работно време.

Показва снимка на магазина.

Предоставя възможности за навигация между различни раздели, включително букети, работници и поръчки.

Позволява на аутентифицираните потребители с определени права да добавят нови букети към магазина.

Позволява на собствениците на магазина да управляват работниците на магазина.

Позволява на потребителите да видят списък с поръчки, свързани с магазина.

AddShop Component:

Този компонент представя формуляр за добавяне на нов магазин с цвята и включва важни полета за въвеждане на информация.

Позволява на потребителите да изберат град, където се намира магазинът.

Предоставя полета за въвеждане на име, адрес, цени и работно време на магазина.

Позволява на потребителите да качат снимка на магазина.

Използва карта за избор на географските координати на магазина.

Валидира въведените данни и предоставя съобщения за грешки при невалидни данни.

Позволява на потребителите да изпратят формуляра за добавяне на магазина.

3.5. Реализация на поръчка и плащания

Следните контролери и класове и компоненти имат за цел да осигуряват функционалността за правене на поръчки и извършване на онлайн плащания в системата Bouquet.bg:

OrderController (Контролер за поръчки):

Този контролер се използва за управление на поръчките в приложението.

GetOrders: Този метод позволява на потребителите да вземат списък с поръчките. Потребителите могат да филтрират поръчките по магазин или потребител.

UpdateStatus: Този метод позволява на администраторите да променят статуса на поръчките.

OrderService (Сървис за поръчки):

Този сървис се използва за извличане и обработка на поръчки.

GetOrdersAsync: Този метод извлича списък с поръчки на база на предоставени параметри като shopID (ако искате да вземете поръчките за определен магазин) или userID (ако искате да вземете поръчките на определен потребител).

UpdateStatus: Този метод променя статуса на поръчка, като приема поръчка по ID и нов статус.

PaymentsController (Контролер за плащания):

Този контролер се използва за управление на плащанията в приложението.

CreatePayment: Този метод позволява на потребителите да създават плащания със съществуващи карти. При успешно плащане, методът генерира уведомления и управлява плащането в портфейла на потребителя.

CreatePaymentWithNewCard: Този метод позволява на потребителите да създават плащания с нови карти. При успешно плащане, методът генерира уведомления и управлява плащането в портфейла на потребителя.

PaymentsService (Сървис за плащания):

Този сървис се използва за обработка на плащанията.

CreatePaymentWithExistingCardAsync: Този метод използва съществуваща карта и създава плащане. При успешно създадено плащане, методът обработва плащането и обновява статуса на поръчката.

CreatePaymentWithNewCardAsync: Този метод създава плащане с нова карта. Преди създаването на плащането, този метод добавя новата карта към потребителския акаунт. След успешно създадено плащане, методът генерира уведомления и управлява плащането в портфейла на потребителя.

Компонентът Cart съдържа списък с списъци от продукти разделени според обекта, от който са добавени. На него може да се намери информация за разбитите поръчки по обекти, съответно обща сума, цена доставка и друго. Списъка от продукти представлява CartBouquets компонента, и за всеки един от тях има бутон, който пренасочва към страницата за поръчка.

Компонентът Order съдържа форма за въвеждане на информация за поръчката, като град, адрес, предпочитан час за доставка, име на получателя, телефонен номер и описание. Също така, има опция за избор на доставка или получаване от магазина. Използва се useForm от react-hook-form за управление на формата и валидация.

Компонентът CartBouquets е отговорен за визуализирането на букетите в кошницата и позволява на потребителите да увеличават или намаляват количеството на букетите в кошницата. Също така, изчислява общата цена на букетите. Използва се библиотеката react-responsive-carousel за показване на снимки на букетите.

Компонентът AddNewPaymentCard съдържа форма за въвеждане на информация за нова плащателна карта. Използва се react-hook-form за управление на формата и валидация на входните данни. Формата включва полета за въвеждане на следните данни: номер на кредитната карта, име на държателя на картата, CVV (защитен код), месец и година на валидност на картата. Потребителите могат да изберат месец и година от падащи списъци. При успешно добавяне на картата, този компонент извиква addCard функцията за създаване на нова плащателна карта и връща потребителя към началната страница или към страницата с поръчката, ако такава е предоставена.

Компонентът PaymentCardsPage се използва за управление на списък със съществуващи разплащателни карти на потребителя. В случай, че потребителят няма добавени карти и има предоставена поръчка, страницата автоматично го препраща към страницата за добавяне на нова карта за плащане. Списъкът с разплащателни карти съдържа информация за последните 4 цифри от номера на картата. Предоставя опция за изтриване на съществуващи карти. Потребителите могат да изберат определена карта и да направят плащане с нея чрез бутона "Плати". При избор на карта и плащане, този компонент извиква pay функцията за плащане с избраната карта.

3.6. Реализация на комуникация между мобилното приложение и сървъра

Сървърната част (MobileHub):

SignalR хъб на сървъра, който се използва за взаимодействие с мобилните клиенти.

В **OnConnectedAsync** метода се установява връзка между клиента и хъба, като се асоциира клиентски идентификатор (например, email) с връзката.

В **OnDisconnectedAsync** метода се премахва клиента от списъка на свързаните клиенти при разкачване.

Хъбът съдържа и други методи, които могат да бъдат извиквани от мобилните клиенти.

Мобилното приложение (SignalRService):

Android сървис (Service), която служи за управление на SignalR връзката и уведомленията на мобилния клиент.

След логин на потребителя се свързва със сървъра чрез SignalR връзка, като използва зададения email като клиентски идентификатор.

При получаване на съобщение от сървъра чрез SignalR, методът **HandleIncomingMessage** създава уведомление (notification) и го изпраща до устройството.

Сървисът съдържа и методи за стартиране и спиране на SignalR връзката, както и за управление на връзката при промяна в интернет свързаността (**ConnectivityChanged**).

Също така, при стартиране на сървиса, тя става фонова (foreground), което позволява на приложението да работи във фонов режим и да получава уведомления дори когато не е активно.

Обобщено, съчетава се функционалността на SignalR за реално времева комуникация със способността на мобилното приложение за управление на уведомления и работа във фонов режим. Това позволява на мобилните клиенти да получават реално време уведомления от сървъра, дори когато приложението не е активно.

3.7. Реализация на обработката на поръчки в мобилното приложение

MainViewModel:

Този клас представлява модел на главния екран на приложението, който показва списък с поръчки.

Списъкът с поръчки се зарежда чрез метода **GetOrders**, който изглежда изпраща заявка към сървъра и обработва отговора.

При избор на поръчка от списъка, се извиква методът **OnItemSelected**, който препраща потребителя към страницата за детайли на поръчката (OrderDetailPage).

OrderDetailViewModel:

Този клас представлява модел на страницата за детайли на поръчка (OrderDetailPage).

При стартиране на страницата се извиква методът **GetOrder**, който изглежда обработва параметъра *item* и инициализира модела на поръчката (Order) със съответните данни.

При натискане на бутона "Промени статус", се извиква методът **ChangeStatus**, който изглежда изпраща заявка към сървъра за промяна на статуса на поръчката.

MainPage и OrderDetailPage:

Те представляват XAML файлове съдържат дизайна на главния екран и страницата за детайли на поръчка.

В тях се използват данни от горните ViewModel класове за показване на информацията на потребителя и управление на действията.

Така структурно е реализирано управлението на поръчки в мобилното приложение. Използват команди, за да реагира на действията на потребителя (като избор на поръчка или промяна на статус), и изпраща заявки към сървъра за обновление на данните за поръчките. Това позволява на потребителите на приложението да виждат и управляват поръчки в обекта в който работи.

Глава 4 Ръководство за ползване

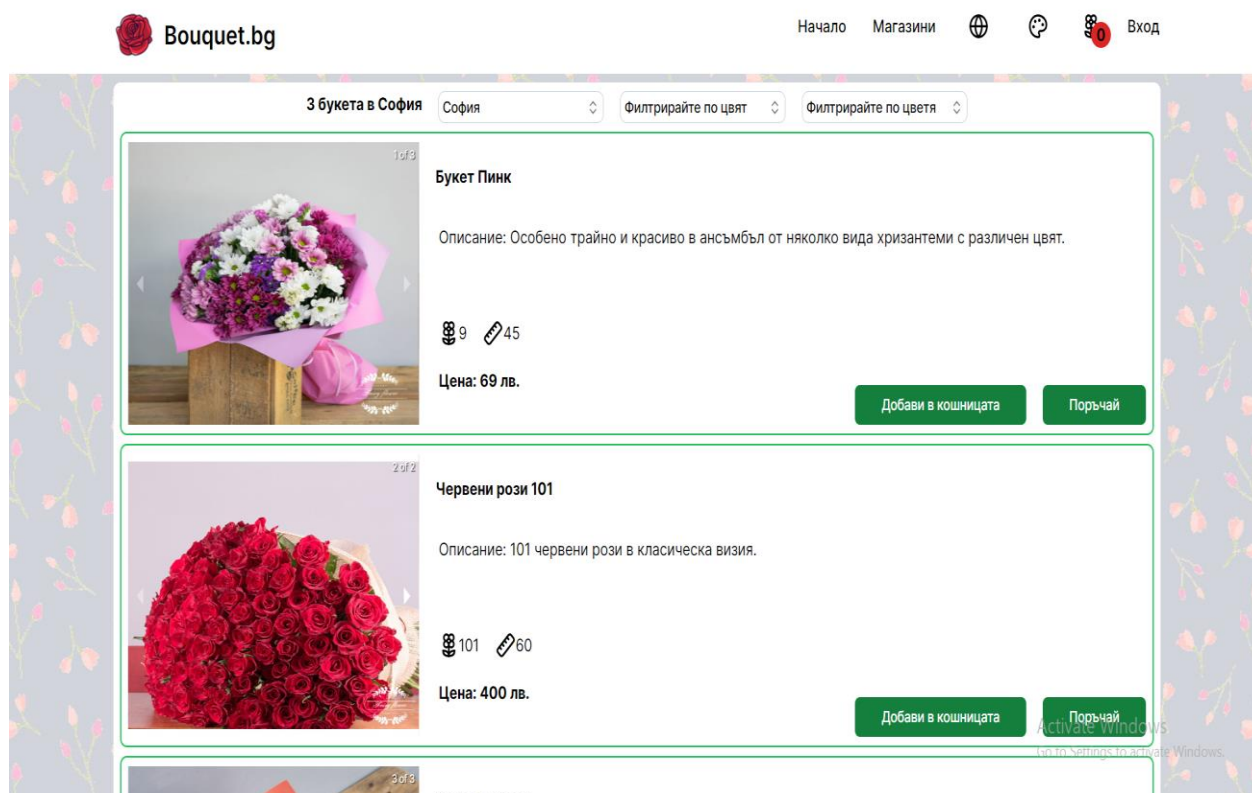
4.1. Софтуерни и хардуерни изисквания

Създаваната система представлява уеб и мобилно приложение приложение, достъпа до уеб приложението се осъществява чрез мрежа като интернет. Това позволява на потребителите да използват системата, независимо от операционната си система, нейната версия, софтуерът, с който разполагат или вида на машината и нейните характеристики стига да притежава интернет браузер. Използвани са последните устойчиви версии на технологиите, така че да се поддържат от всички браузери, което осигурява безпроблемна работа с приложението. Мобилното приложение от своя страна трябва да бъде инсталирано на Андроид или ios устройство, но това е наложашо, заради спецификата му, например функционалността за известяване, а и то ще се използва само от служители и наложашата инсталация не голям проблем. Минималната изискваща се версия за мобилното приложение за Андроид е Android 9.0, а за ios е Xcode 14.1. Това са минималните изискващи се версии за качване на приложението в Google Play и Apple App store към момента на изготвяне на тази дипломна работа.

Разработена по този начин, системата става общо достъпна за клиентите и позволява да бъде използвана от широк кръг от потребители. Това е и главната причина да бъде избрана системата да се разработи като уеб приложение, за да бъде лесно достъпна от всички потребители без специфични изисквания и условности. А за служителите има отделно мобилното приложение, което им дава удобство, практичност и възможност за повече функционалности, които не са специфични за уеб приложенията.

4.2. Визуализация на потребителски интерфейс и ръководство

Началната страница (Фигура 4.2.1) на уеб приложението на системата за онлайн търговия bouquet.bg представлява екран със списък от букети.

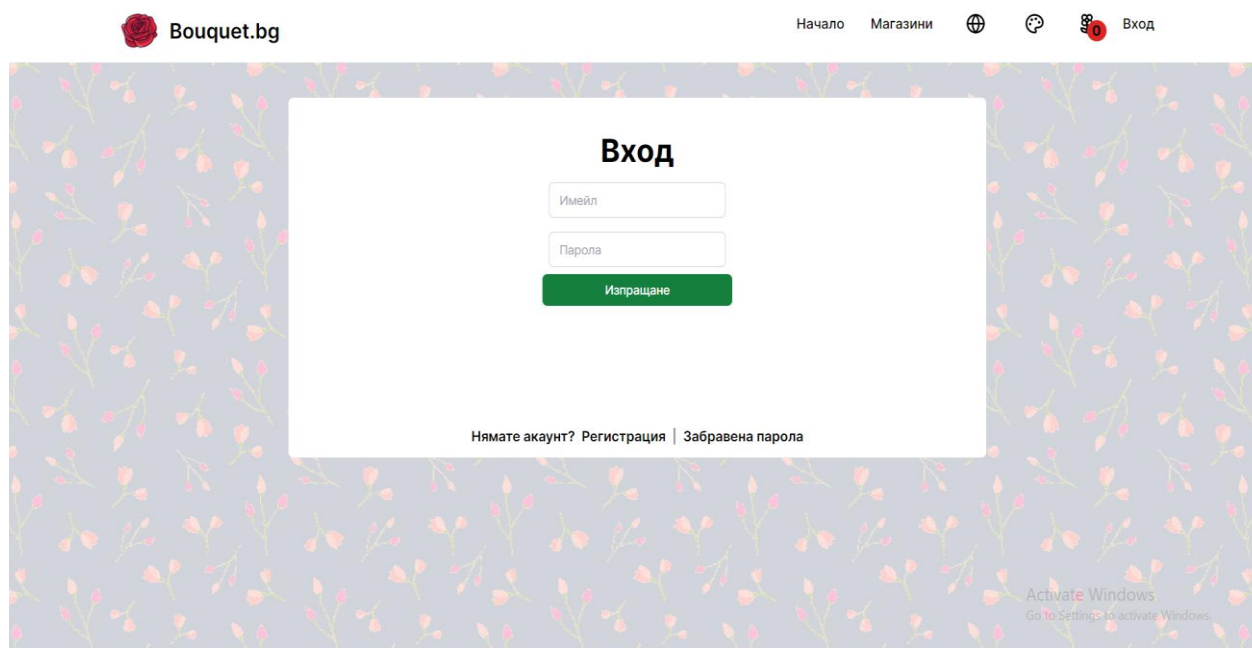


Фигура 4.2.1 Начална страница на системата bouquet.bg

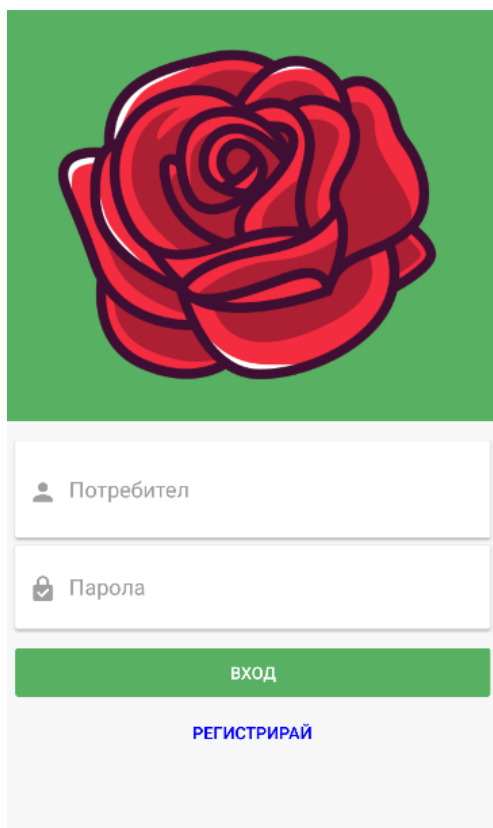
На всяка страница в горната част винаги стои навигационното меню. В него може да намерим следното: логот и името на сайта, при клик върху тях сайта зарежда началния екран. В дясната част на менюто може да намерим бутон „Начало“, който също ни препраща към главния екран, бутон „Магазини“ с него се отваря страницата с магазините, които работят със системата. След това има две иконки, които при клик се разпъват и всъщност представляват селектори. Първият е за език, тъй като сайтът поддържа два езика – български и английски. Вторият е за тема, защото сайтът има две теми – тъман и светла, но в падащото меню има и трета опция, а именно системна, която взема системната настройка на потребителя. След тях има още една иконка с цифра на нея, това е кошницата, а цифрата показва броя на букетите в нея. При клик върху нея се отваря страницата за кошницата. Най-вдясно се намира бутон за вход на потребител или снимката на потребителя, ако той вече се е вписъл. При клик върху нея се разпъва падащо меню с два бутона – „Профил“ и „Изход“, съответно за отваряне на страницата на профила или за изход от системата.

На началната страница виждаме списък с букети. В над списъка можем да видим информация за това колко продукта са намерени за избрания град, до нея има три филтъра, които представляват падащи менюта. Първият е именно за града, не може да не е избран град, ако системата е получила достъп до местоположението на потребителя, избрания град е най-близкия до местоположението на потребителя, ако не то избрания град е София. Само един град може да бъде избран в даден момент, не може да се избират няколко града едновременно. Вторият е филтъра по цвят, при него може да се селектира повече от един цвят. Последният е филтъра по сорта на цветята, при него също може да се селектират няколко сорта едновременно, тъй като може да има букети с няколко различни цветя. При промяна на селекцията, на който и да е от филтрите списъка със стоки се обновява. Всяки букет в списъка е разделен от другите визуално. Той е визуализиран, като вляво се виждат неговите снимки. Те се презаменят една по една през определено време, потребителя също може да ги сменя със стрелките в двата края, ако държи курсора върху снимката, то тя няма да се замени. Вдясно от снимката се намира информация за букета: име, описани, височина, брой цветя и цена. В долната част има два бутона, единият е за добавяне на букета в количката и един за директна поръчка.

Страниците за вход (Фигура 4.2.2) на уеб и (Фигура 4.2.3) мобилното приложения на системата за онлайн търговия bouquet.bg представляват форми.



Фигура 4.2.2 Страница за вход в системата bouquet.bg (уеб)

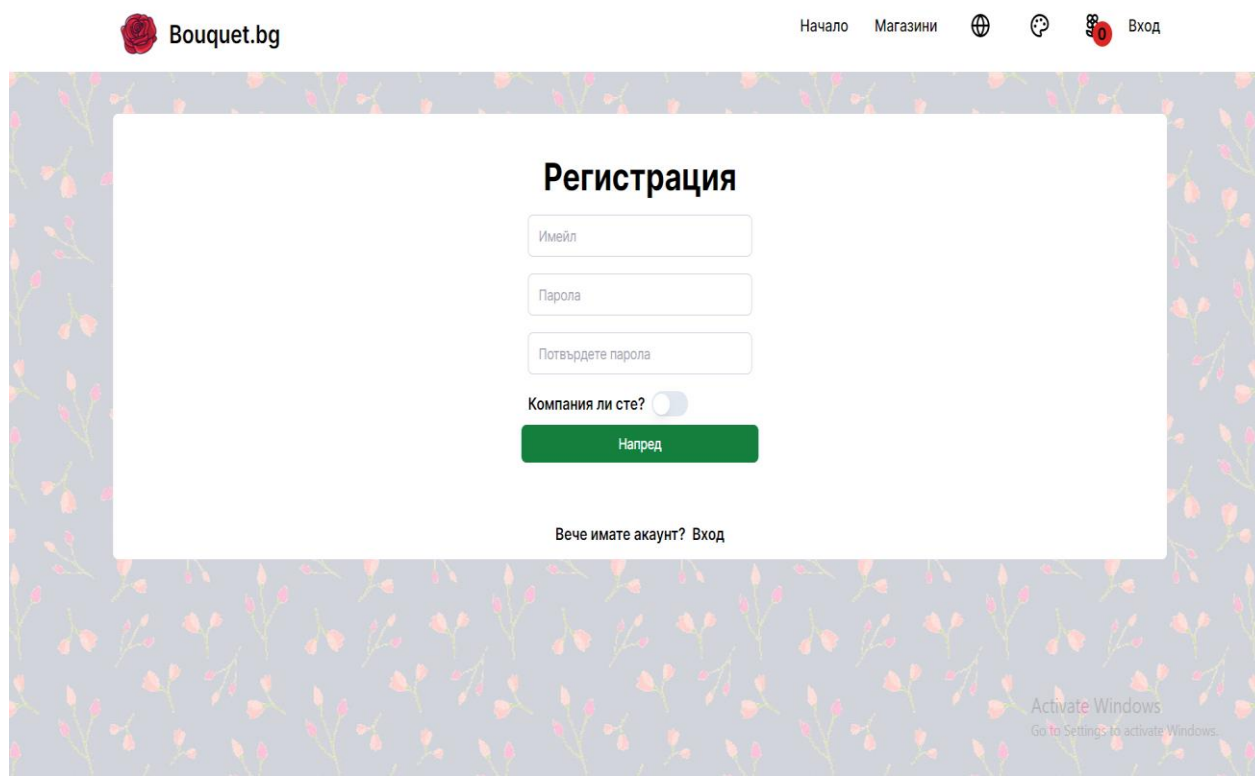


Фигура 4.2.3 Страница за вход в системата bouquet.bg (мобилно)

Във формите се въвеждат потребителските данни, нужни за влизане в системата – имейл адрес и парола, след което се натиска бутона вход. При опит за влизане в системата, без да са попълнени нужните полета, се изписват предупредителни съобщения, указващи, че полетата са задължителни. При неправилно въведени данни се изкача съобщение за грешка в долната част на екрана. След натискане на бутона за влизане в системата и успешен вход потребителят бива прехвърлен към началния екран.

При кликане върху бутона за регистрация в мобилното приложение, потребителя се прехвърля към страницата за регистрация на уеб приложението. Мобилното приложение умишлено няма регистрация, защото нов акаунт няма правата да използва мобилното приложение, то е само за потребители с роля служители или такава с по-високо ниво на достъп. Имено за това не може да се извършват регистрации през мобилното приложение.


Ако потребителят все още не е регистриран в базата и системата трябва да натисне линка под формата за влизане. Чрез него потребителят бива прехвърлен на първата страницата за регистрация (Фигура 4.2.4).

The image shows the registration page of the Bouquet.bg website. At the top, there is a navigation bar with the Bouquet.bg logo on the left and links for 'Начало' (Home), 'Магазини' (Stores), a globe icon, a speech bubble icon, a shopping cart icon, and 'Вход' (Login). The main content area has a light blue background with a repeating pattern of small pink flowers. In the center, there is a white rectangular box containing the registration form. The form is titled 'Регистрация' (Registration) in bold black text. It includes three input fields: 'Имейл' (Email), 'Парола' (Password), and 'Потвърдете парола' (Confirm password). Below these fields is a toggle switch for 'Компания ли сте?' (Are you a company?). At the bottom of the form is a green button labeled 'Напред' (Next). Below the form box, there is a link that says 'Вече имате акаунт? Вход' (Already have an account? Login). In the bottom right corner of the page, there is a small 'Activate Windows' watermark.




Фигура 4.2.4 Първа страница за регистрация на потребители

Както вече знаем регистрацията в системата е на две стъпки, първата страницата за създаване на акаунт съдържа форма, в която трябва да се въведат общите данни на потребителя. Нужната данни тук са: имейл адрес и парола и дали потребителя се регистрира като компания. С цел валидиране е поставено второ поле за потвърждаване на паролата. При непълнени полета се изписват предупредителни съобщения, указващи, че полетата са задължителни. При опит за регистрация с вече съществуващ имейл в базата данни, се изписва съобщение за грешка.

Втората стъпка е различна според това дали потребителя е избрал да се регистрира като компания или фирма. Фигура 4.2.5 представлява регистрацията на обикновен потребител, а Фигура 4.2.6 регистрация на компания.

 Bouquet.bg

НачалоМагазини

 Вход


Регистрация на частно лице

Регистрация




Вече имате акаунт? [Вход](#)

Activate Windows
Go to Settings to activate Windows.

Фигура 4.2.5 Страница за регистрация на часно лице

 Bouquet.bg

НачалоМагазини

 Вход

Регистрация на фирма

Регистрация

Вече имате акаунт? [Вход](#)

Activate Windows
Go to Settings to activate Windows.

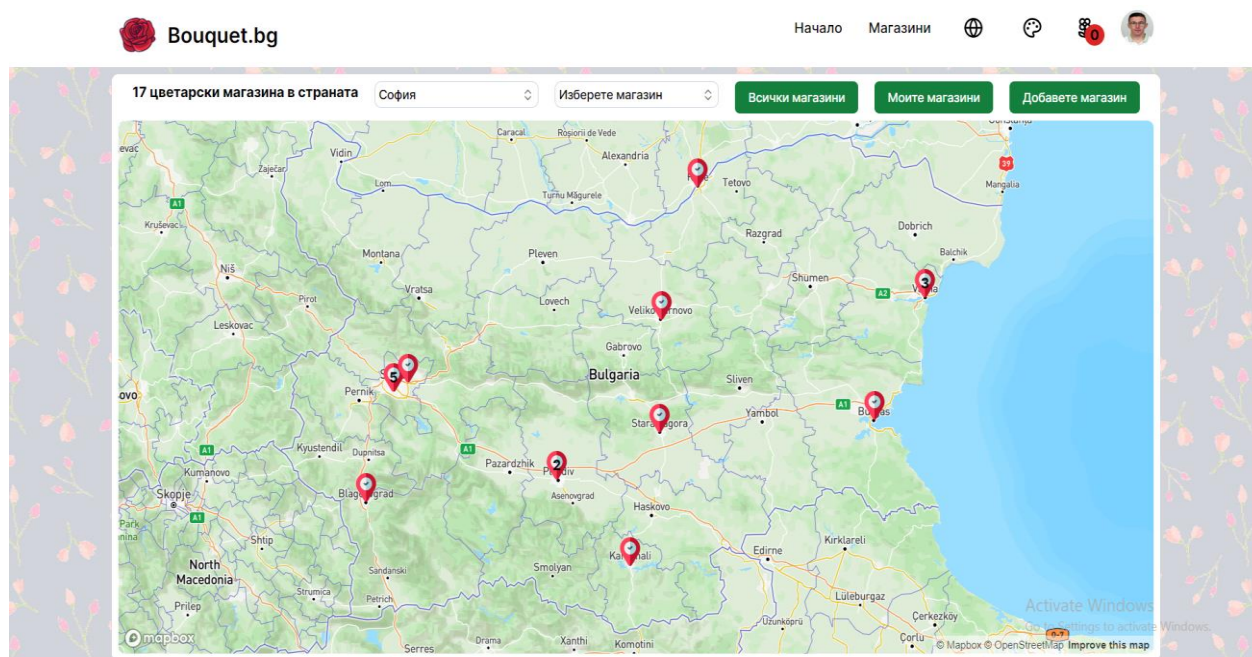
Фигура 4.2.6 Страница за регистрация на компания

Всяка от двете страницата за съдържа форма, в която трябва да се въведат съответните данните на потребителя. Нужната информация зависи от това какъв потребител регистрираме. При непълнени полета се изписват предупредителни съобщения, указващи, че полетата са задължителни. При опит за регистрация с вече съществуващ имейл в базата данни, се изписва съобщение за грешка. На всеки от трите страници в долния ъгъл има препратка към страницата за вход.

След вярно попълнени данни, потребителят натиска бутона „Регистрирай се“, регистрацията му се завършва и бива прехвърлен отново към потребителския вход, където се изписва съобщение за успешна регистрация.

Със своята регистрация потребителя създава свой акаунт, но той все още не е активиран. За да активира акаунта си той трябва да потвърди през линка, който му е изпратен по email и едва след това може да използва акаунта си. Потребителите, които се регистрират като компания получават клиентска роля след регистрация. За да придобие един акаунт партньорска роля, то той трябва да бъде одобрен от администратор. Това се случва от друг екран в който се виждат всички акаунта регистрирани като компании и чакащи одобрение от администратор.

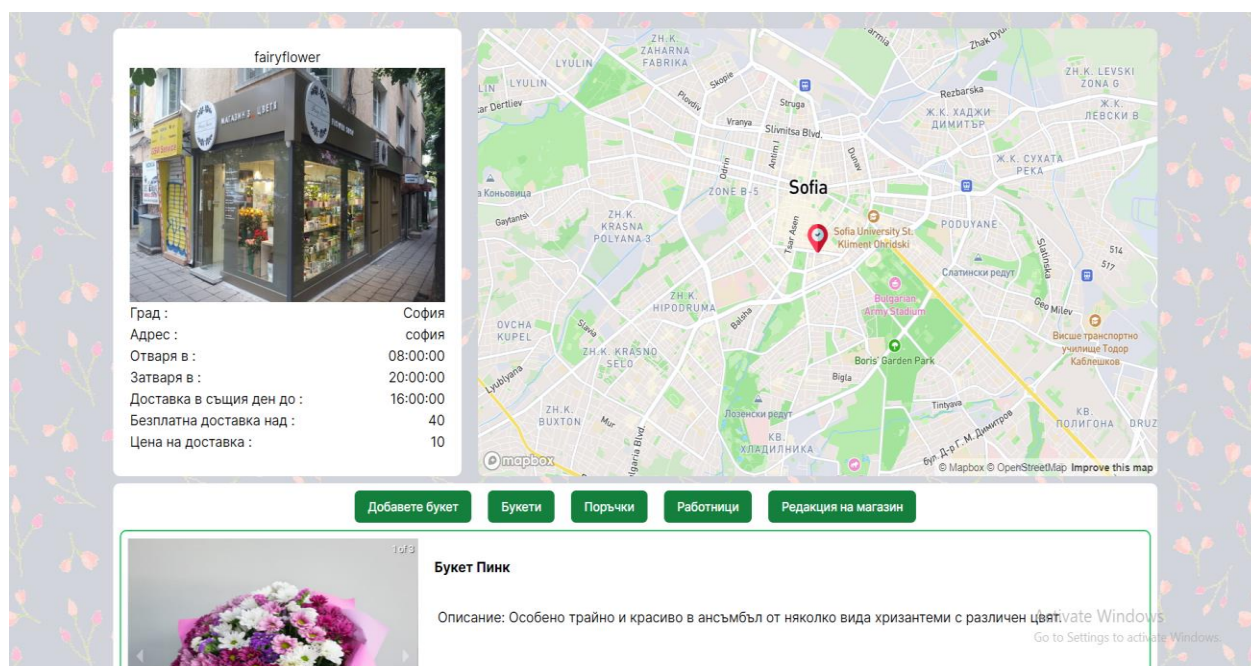
Втората главна страница на уеб приложението (Фигура 4.2.7) е страницата с магазините.



Фигура 4.2.7 Страница с всички магазини

На тази страница има карта с точки по нея, това са всички магазини регистрирани в системата. Над нея има информация колко са те в страната и два филтъра. Първия е за град, при избор на някой град, на картата се показва само избрания град. Вторият филтър е по конкретен магазин, при селекция на магазин на картата се показва само избрания магазин. На тази страница потребителите с роля партньор или такава с по-високи права на достъп виждат и три бутона. Един който показва всички обекти, такъв който показва само обектите на конкретния потребител и един за добавяне на обект.

При клик върху някоя от точките се показва прозорче със снимката на обекта, неговото име и два бутона, един за навигиране до обекта и един за отваряне на страницата за конкретния обект.



Фигура 4.2.8 Страница на магазин

На страницата за даден магазин се виждат три отделни блока. В първия имаме информация за магазина – име, снимка, град, адрес, работно време и други. Във втория имаме карта, на която се вижда само този обект. В третия, по подобие на главния екран се виждат списък с букети, но този път само тези на избрания магазин. Потребителите с роля сужител виждат триа бутона, за добавяне на букет, всички букети и поръчки. Бутон за добавяне на букет отваря нова страница, а останалите динамично

презаменят съдържанието в третия блок. Потребителите с роля партньор или такава с по-високо ниво на достъп, виждат още два бутона – работници и редакция на магазин.

Бouquet.bg

Начало Магазини

Добави букет

Добавете снимка

Цветя: Филтрирайте по цветя Цветове: Филтрирайте по цветя

Име: Име Цена: Цена

Брой цветя: Брой цветя Височина: Височина

Описание: Описание Изпращане

Фигура 4.2.9 Страница за добавяне на букет

Страницата за добавяне на букет (Фигура 4.2.9) представлява форма, в която има бутон за добавяне на снимки, два селектора, за цветове и цветя, с които може да се селектира повече от един елемент. След това има полета за нужната за букета информация.

Бouquet.bg

Начало Магазини

buketite.net

Кутия "Намигване"

15 30

Цена: 67 лв. Брой: - 2 + Общо: 134 лв.

Безплатна доставка над: 30 лв. Общо: 134 лв. Общо с доставка: 134 лв. Поръчай

fairyflower

Букет Пинк

9 45

Цена: 69 лв. Брой: - 1 + Общо: 69 лв.

Червени рози 101

101 60

Цена: 400 лв. Брой: - 1 + Общо: 400 лв.

Букет Коралов

Фигура 4.2.10 Страница за кошницата

Страницата за кошницата (Фиг. 4.2.10) представлява отделни блокове, списъци със стоки, разделени на отделни блокове, по обекта от който са добавени. Всяка стока в списъка има информация, брой, който може да се променя, единична и обща цена. Всеки блок представлява отделна поръчка и информация за нея, като сума, цена на доставка и други.

Поръчка

☒ С доставка

Град: София

Адрес: Адрес

Предпочитан час за доставка: Предпочитан час за доставка

Получател: Красимир

Номер на получателя: 0893458999

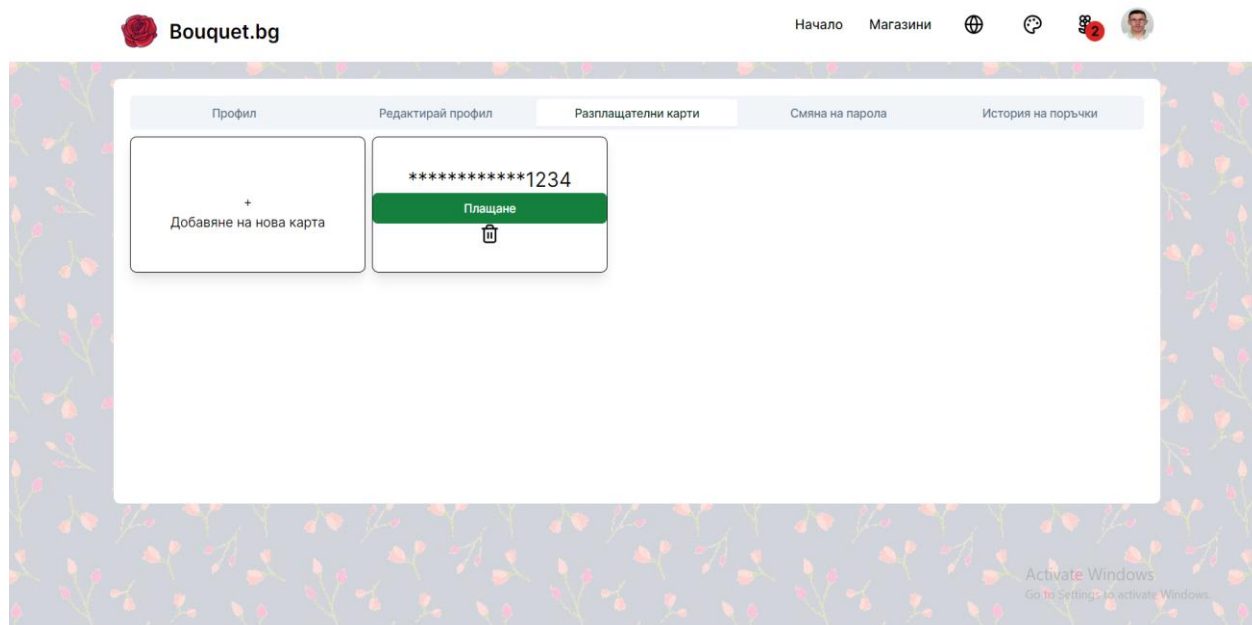
Описание: Описание

Цена на доставка: 0 Общо: 529

Поръчай

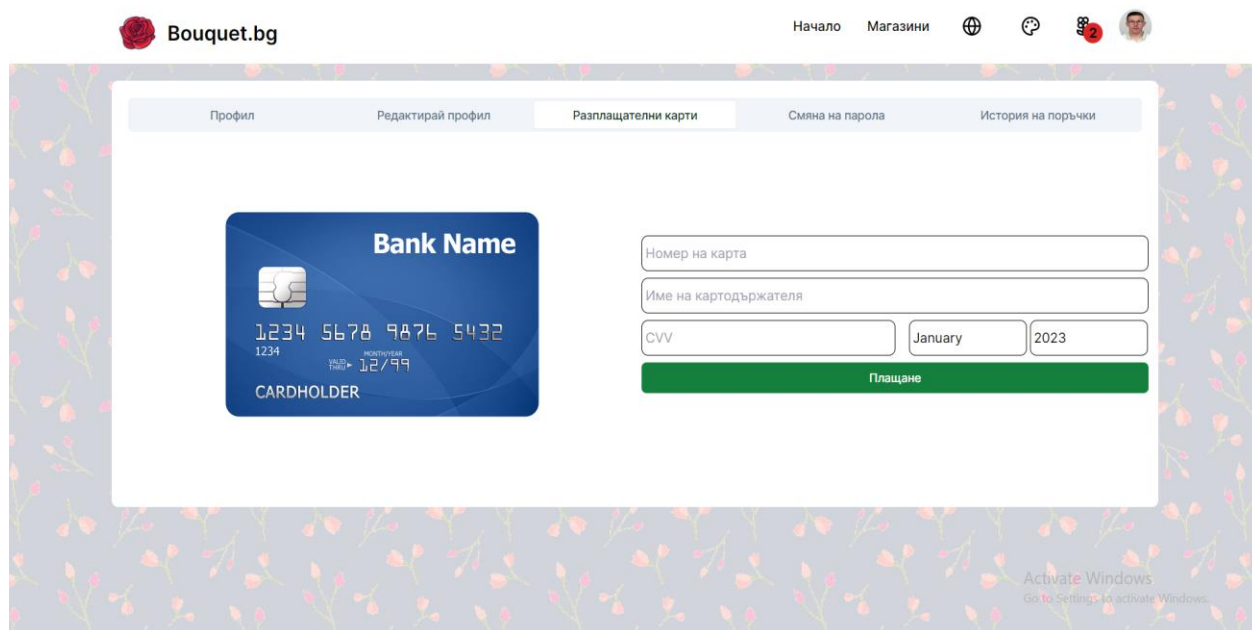
Фигура 4.2.11 Страница за поръчка

Страницата за поръчката (Фиг. 4.2.11) има форма за въвеждане на данните за доставка, ако има такава. В горната част на екрана има бутон с две положения, включен и изключен, съответно с или без доставка. Ако бутона е изключен, полетата за въвеждане изчезват. В долната част на екрана има информация цената на доставката и общата сума. Под нея има бутон за поръчка, ако поръчката е без доставка, тя се пренасочва към служителите, но ако поръчката е с доставка, потребителя бива пренасочен към плащане и поръчката няма да стигне до служителите, докато не бъде платена.

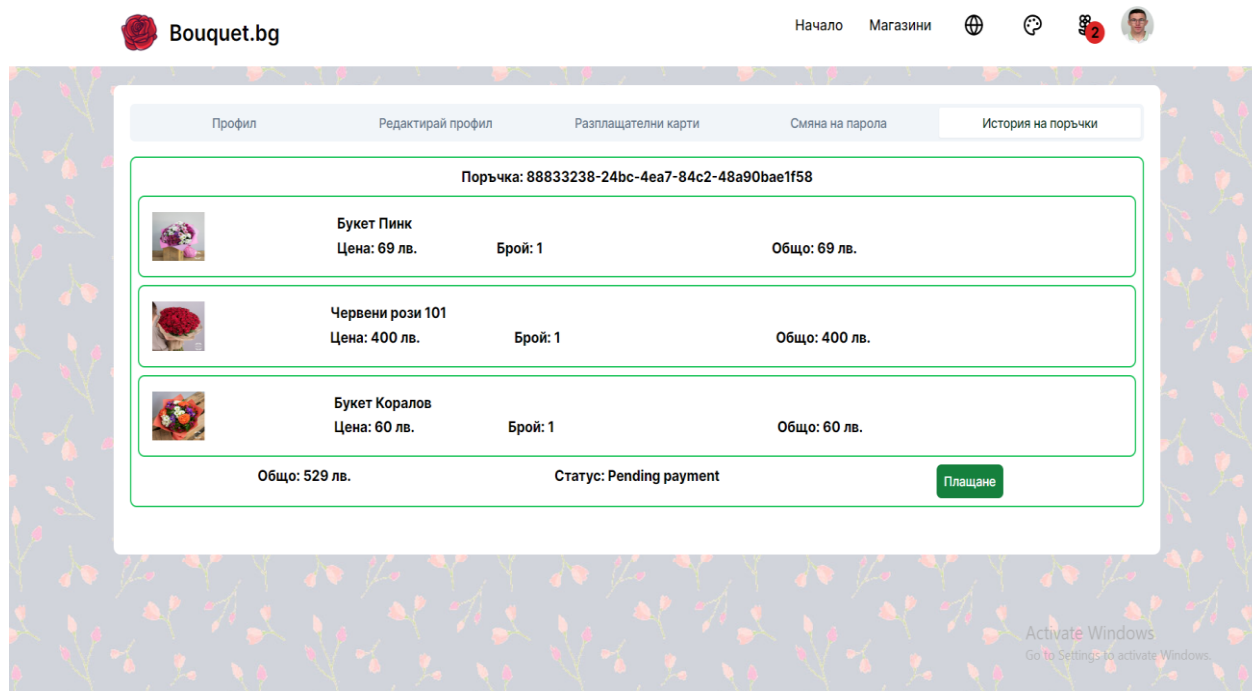


Фигура 4.2.12 Страница с разплащателни карти

След поръчка с доставка потребителя е препратен към плащане, ако той вече има добавена карта вижда страницата с разплащателни карти (Фиг. 4.2.12), в противен случай ще вижда страницата за добавяне на нова карта (Фиг. 4.2.13). На страницата с разплащателни карти има списък от вече добавените карти, с които потребителя може да заплати поръчката. На страницата за добавяне на карта има форма с полетата необходими за данните за добавяне на нова карта. След плащане с новата карта тя бива запазена, за следващи плащания.



Фигура 4.2.13 Страница за добавяне на нова карта

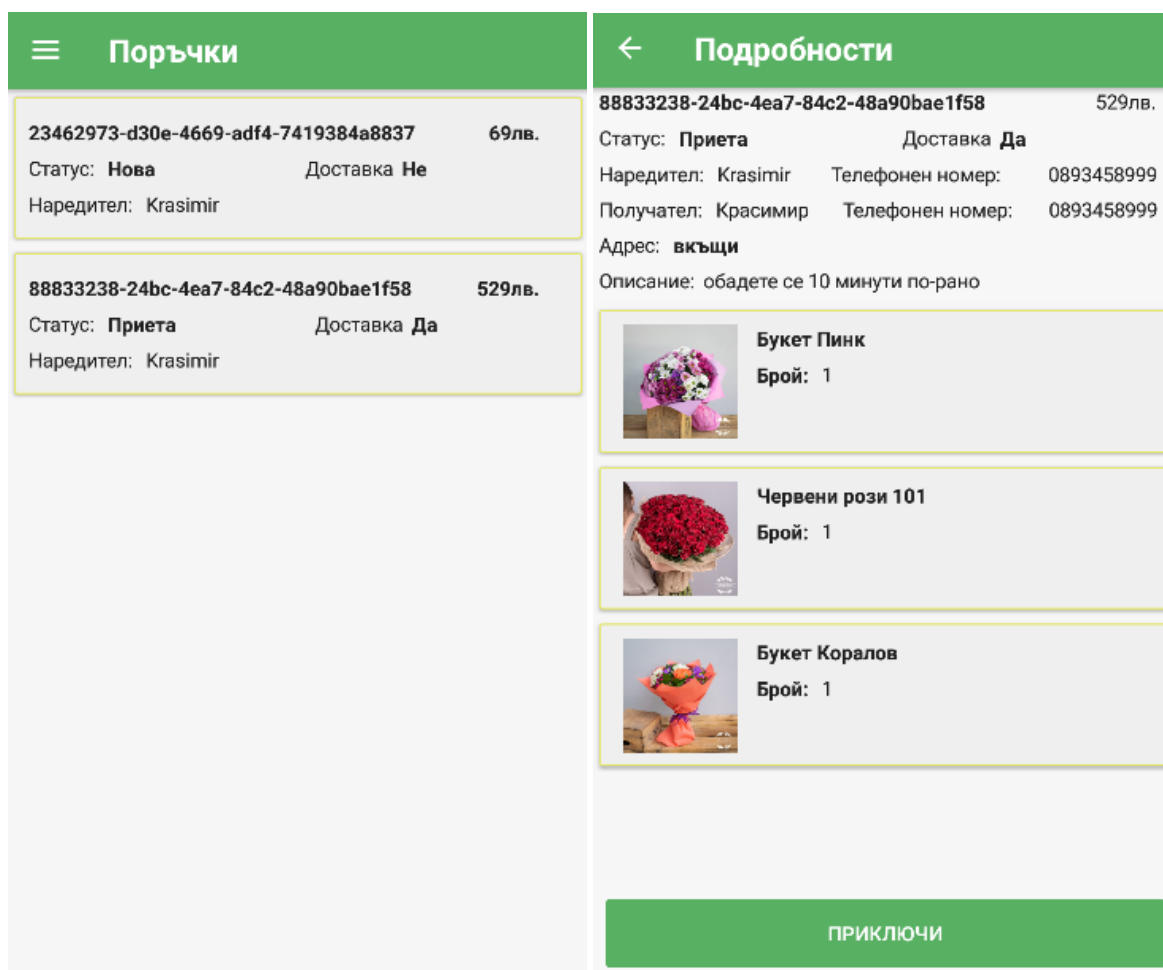


Фигура 4.2.14 Страница история на поръчки

При отваряне на страницата на профила се зарежда екран с пет бутона в горната част. Това са пет различни страници – профил, редакция на профил, разплащателни карти, смена на парола и история на поръчки.

На фигура 4.2.14 е показана страницата за история на поръчки. В нея потребителите могат да виждат какъв е статуса на поръчката. В примера на фигурата поръчката не е била платена по някаква причина, от тази страница това може да се направи и в по-късен момент от време.

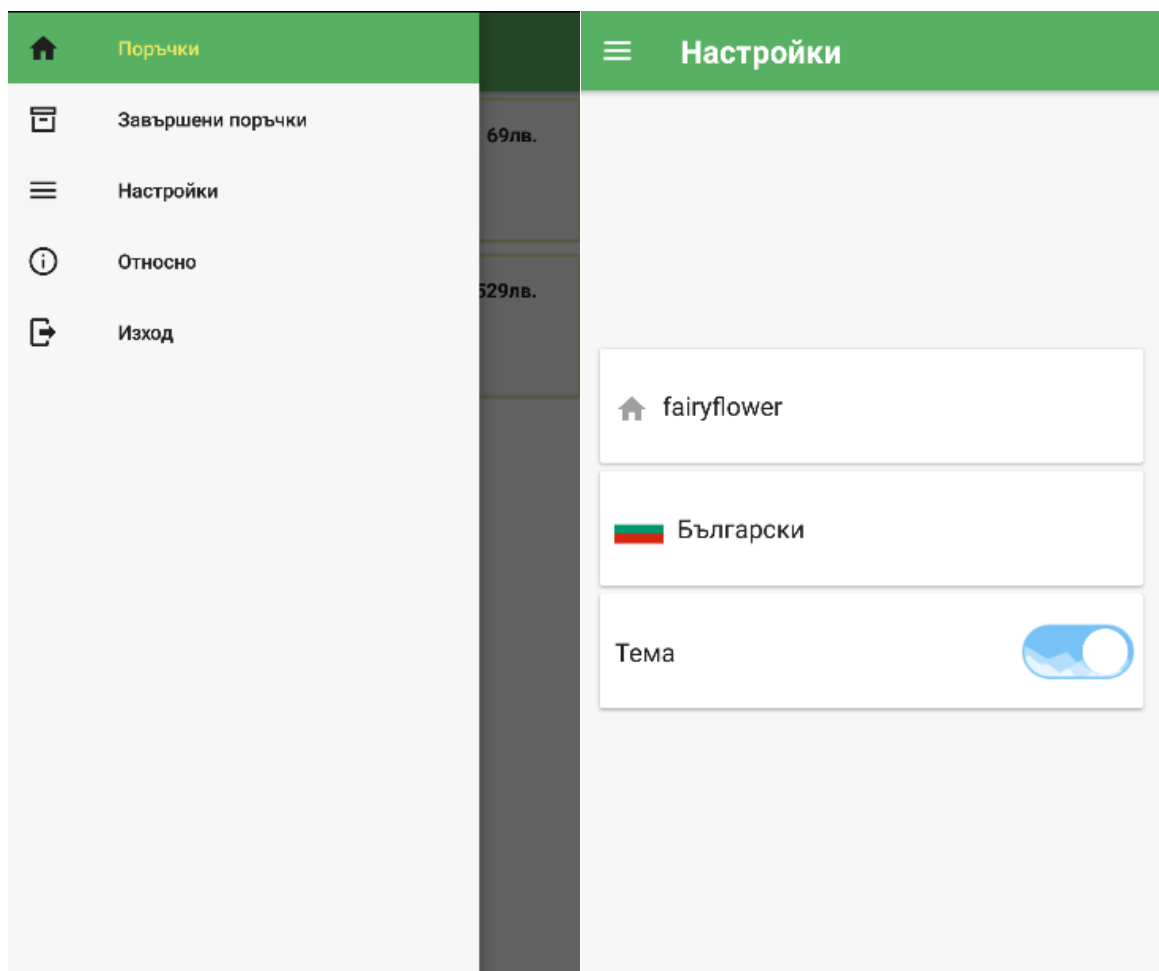
Главната страница на мобилното приложение (Фигура 4.2.15) представлява списък с поръчки с най-важната информация за тях. Върху всяка поръчка може да се кликне, което ще отвори страницата за поръчка на мобилното приложение. Тази страница може да се скролира, ако на нея има повече поръчки. Също така на главната страница се показват само поръчките, които все още не са приключени. За приключените поръчки има отделна страница, която изглежда по абсолютно същия начин.



Фигура 4.2.15 Начална страница на мобилното приложение

Фигура 4.2.16 Страница за поръчка

Страницата за конкретна поръчка в мобилното приложение (Фигура 4.2.16) съдържа всичките необходими данни за поръчката – номер, цена, статъс, данни на наредителя и на получателя, адрес за доставка, вижда се само ако има доставка, описание и списък с поръчаните букети и техния брой. Списъка заема винаги едно и също пространство от екрана, тоест не избутва и крие останалите компоненти, защото може да се скролира само в неговата си собствена секция. В долната част на екрана има бутон, който може да е различен в зависимост от статуса на поръчката. Той може да е „Приеми“ или „Приклучи“ поръчката и съответно променя статуса на поръчката.



Фигура 4.2.17 Меню на мобилното приложение

Фигура 4.2.18 Настройки на мобилното приложение

Менюто на мобилното приложение (Фигура 4.2.17) е много просто. В него има пет опции – поръчки, приключени поръчки, настройки, относно и изход. В страницата относно има информация за приложението, като версия, предназначение и друго.

Страницата с настройките (Фигура 4.2.18) на мобилното приложение има три настройки. За обект, това е обекта в който работим в момента, може да се избира измежду всички обекти, в които има право да работи дадения потребител. Следващата настройка е за език. Мобилното приложение по подобие на уеб приложението поддържа два езика, български и английски. Последната настройка е за темата, мобилното също има две теми, светла и тъмна.

4.3. Тестване

В настоящата дипломна работа обект на извършените тестове е единствено софтуерът на разработваната система. Целта на тестванията е да се провери коректната работа на създадената система, както и дали изпълнява всички изисквания.

Всички извършени тестове са контролирани с ръчни средства. Разиграни са различни сценарии и случаи за да е сугорно, че системата работи, както се очаква и осигурява всички предвидени функционалности. В таблица 4.3.1 са изброени всички направени тестове и техните резултати. Тестовите които са приложими едновременно за мобилното и уеб приложението са упоменати само веднъж, но са извършени върху двете приложения.

Таблица 4.3.1 Матрица на функционалните изисквания

<i>Req №</i>	<i>Requirement</i>	<i>TC Id</i>	<i>Test Case</i>	<i>Status</i>	<i>Bugs</i>
1	Системата да предоставя вход и изход на потребителите си	TC01	Проверка дали при въвеждане на коректни данни потребителят влиза успешно в системата	Преминал успешно	-
		TC02	Проверка дали при въвеждане на невалидни данни се изписва съответното съобщение	Преминал успешно	-
		TC03	Проверка дали при натискане на бутона „Изход“ потребителят излиза от системата и се прекратява сесията му	Преминал успешно	-
2	Системата да показва и дава достъп или не до различни страници или бутони, според правата на ролята	TC04	Проверка дали клиентите нямат достъп до определени функционалности	Преминал успешно	-
		TC05	Проверка дали служителите и партньорите има достъп до своите функционалности	Преминал успешно	-
		TC06	Проверка дали администраторът има достъп до всички функционалности	Преминал успешно	-
3	Системата да позволява търсене в нейните ресурси	TC07	Търсене на букети и визуализиране на резултат	Преминал успешно	-
		TC08	Търсене на магазини и визуализиране на коректен резултат	Преминал успешно	-
4	Системата да разпределя магазините по градове	TC09	Проверка дали се визуализират градовете	Преминал успешно	-
		TC10	Проверка дали при избор на град се визуализират съответните магазини	Преминал успешно	-

<i>Req №</i>	<i>Requirement</i>	<i>TC Id</i>	<i>Test Case</i>	<i>Status</i>	<i>Bugs</i>
5	Клиентите да могат да правят и разплащат поръчки	TC11	Проверяване правенето на поръчка	Преминал успешно	-
		TC12	Проверяване разплащането на конкретна поръчка	Преминал успешно	-
6	Служителите да могат да обработват поръчките на клиентите	TC13	Проверка дали служителите виждат текущите поръчки на клиента	Преминал успешно	-
		TC14	Проверка дали служителят може да приеме поръчка	Преминал успешно	-
		TC15	Проверка дали служителят може да приключи конкретна поръчка	Преминал успешно	-
7	Партньорите да могат да добавят нови служители	TC16	Проверка дали при въвеждане на нов служител той е променен в базата данни	Преминал успешно	-
8	Администраторите да могат да одобряват партньори	TC17	Проверка одобряване на партньори	Преминал успешно	-
9	Администраторите да могат да редактират магазини, дори те да не са негова собстреност	TC18	Проверка дали се визуализира бутона за редактиране и дали има достъп до страницата	Преминал успешно	-
		TC19	Проверка дали промените се отразяват в базата данни	Преминал успешно	-

<i>Req №</i>	<i>Requirement</i>	<i>TC Id</i>	<i>Test Case</i>	<i>Status</i>	<i>Bugs</i>
10	Служители и партньори да могат да добавят нови букети	TC20	Проверка дали при въвеждане на невалидни данни се изписват съобщения за грешка	Преминал успешно	-
		TC21	Проверка дали с въвеждане на данните се създава съответния запис в базата данни	Преминал успешно	-
11	Партнюрите да могат да добавят нови магазини	TC22	Проверка дали се създава обект в базата данни	Преминал успешно	-
12	Служителите да получават нотификация на мобилното устройство	TC23	Проверка дали служителите получават нотификация при работещо приложение	Преминал успешно	-
		TC24	Проверка дали служителите получават нотификация при изключено приложение	Преминал успешно	-
13	В мобилното приложение не трябва да могат да се вписват клиенти или други роли без работно място	TC25	Проверка дали имаме достъп с валиден потребител	Преминал успешно	-
		TC26	Проверка дали имаме ще ни бъде отказан достъп с невалиден потребител	Преминал успешно	-

Заклучение

Целта на настоящата дипломна работа бе да се създаде система за онлайн търговия, изградена от уеб и мобилно приложение и тя е изпълнена.

Системата за онлайн търговия, bouquet.bg, бе разработена успешно и предоставя на своите потребители всички нужни функционалности за гладка и безпроблемна работа, осигурявайки им удобство и приятно изживяване. Тази система предоставя възможност на цветарските магазини да предлагат своите продукти в един централизиран портал. Разработеното уеб приложение, позволява да бъде използвана от всякакъв тип устройства, без значение от локацията на потребителя.

Освен това системата bouquet.bg е единствената от проучените в *Глава 1*, която позволява работа с множество магазини. Това я превръща в централизиран портал и е най-голямото ѝ предимство пред останалите.

Системата за онлайн търговия подобрява ефективността на служителите и потребителите, също така насърчава конкурентоспособността на магазините, като позволява на потребителите да разглеждат и сравняват продукти от различни доставчици, което им спестява време и лутане между сайтовете на различните магазини, подобрява тяхното изживяване при пазаруване. Тя предоставя лесен достъп до информация за продуктите и предлага удобен начин за извършване на поръчки. Чрез използването на приложението се намалява разходите при онлайн търговията за поддържането за самостоятелен уеб сайт и увеличава продуктивност на служителите по време на работа, поради лесния начин за обработка на поръчки. Не на последно място използването на подобна система прави цветарските магазини, които я използват по-забележими и достъпни за клиентите .

Приложението, разработено чрез настоящата дипломна работа, е създадено по начин, който позволява по-нататъшно развитие и разширение на неговите възможности.

Добавянето на нови функционалности може да бъде реализирано безпроблемно, без да се налагат промени по вече създадените структура и архитектура, единствено чрез създаване на нови компоненти.

Функционалности, които биха могли да се разработят и добавят към системата bouquet.bg, с цел подобряването ѝ, са:

- ❖ допълнителни функционалности в мобилното приложение – добавяне на функционалности като добавяне и редактиране на продукт, редакция на обект, редакция на профил и други;
- ❖ следене на инвентара – системата да може да следи и оправлява наличности и да предлага само налични продукти, на този етап това става с намеса на сужител, кото редактира букет;
- ❖ проследяване на доставка – мобилното приложение да се използва при доставка, да се интегрира карта с навигация, за улеснение на доставчиците;
- ❖ известяване на клиентите – при промяна в статута на поръчката, клиента да бива известяван за това, на този етап клиентите могат да следят поръчките си, но не биват известявани за промяна;

Тези функционалности биха разширили възможностите на системата по начин, който допълнително ще оптимизира работата и времето на служителите и ще автоматизира по-голямата част от процесите.

Системата за онлайн търговия, bouquet.bg, съчетава нужните функционалности за ефикасна, бърза и лесна работа. Помага на потребителите при търсенето, избирането, поръчката и заплащането при покупка на букети, а на служителите с приемане и обработването на поръчките, добавянето на нови букети и обекти и служители и цялостното управление на цветарски магазини. Внедряването и използването на системата модернизира търговията на цветарските магазини.

Източници

- 1) Microsoft's .NET официален уебсайт - <https://dotnet.microsoft.com>
- 2) ASP.NET Core официална документация - <https://docs.microsoft.com/en-us/aspnet/core>
- 3) Официална документация на Microsoft SQL Server - <https://docs.microsoft.com/en-us/sql>
- 4) Официален уебсайт на Node.js - <https://nodejs.org>
- 5) Официален уебсайт на React - <https://reactjs.org>
- 6) Официален уебсайт на Xamarin - <https://dotnet.microsoft.com/apps/xamarin>
- 7) Официална документация на SignalR - <https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0>
- 8) Марк Дж. Прайс, "C# 9 and .NET 5 – Modern Cross-Platform Development", 2020г.
- 9) Филип Джапикс и Кевин Гросниклаус, "Building Web APIs with ASP.NET Core", 2018г.
- 10) Рандолф Уест, Уилям Асаф, Свен Аелтерман и Минди Кърнът, "SQL Server 2019 Administration Inside Out", 2019г.
- 11) Алекс Банкс и Ив Порселло, "Learning React", 2017г.
- 12) Марио Касчаро, "Node.js Design Patterns", 2014г.
- 13) Марк Ласоф и Стивън Ф. Даниел, "Xamarin.Forms Essentials" , 2019г.

Приложение код

Линк към изходния код в Github: <https://github.com/k-mitkov/Bouquet.bg>

1. Добавяне на нов магазин

Controller

```
/// <summary>
/// Добавяне на обект
/// </summary>
/// <returns></returns>
[Authorize(Policy = RoleClaim.AddFlowerShop)]
[HttpPost]
public async Task<IActionResult> AddObject([FromBody] AddFlowerShopRequest shopRequest)
{
    var email = Request.GetEmailFromAccessToken(_tokenHelper);

    var result = await _flowerShopService.AddShopAsync(shopRequest, email);

    if (result.Status == StatusEnum.Failure)
        return BadRequest(result);

    return Ok(result);
}

/// <summary>
/// Обновява снимката на обекта
/// </summary>
/// <param name="formData"></param>
/// <returns></returns>
[Authorize(Policy = RoleClaim.AddFlowerShop)]
[HttpPost("upload-shop-picture")]
public async Task<IActionResult> UploadObjectPicture([FromForm] IFormCollection formData, [FromQuery] string shopID)
{
    if (formData == null || formData.Files.Count == 0)
        return BadRequest();

    IFormFile file = formData.Files.GetFile("file");

    if (file == null || file.Length == 0)
        return BadRequest();

    var result = await _flowerShopService.UploadPictureAsync(file, shopID);

    if (result.Status == StatusEnum.Failure)
        return StatusCode(StatusCodes.Status500InternalServerError, result);

    return Ok(result);
}
```

Service

```
/// <summary>
/// Добавяне на обект
/// </summary>
/// <returns></returns>
public async Task<Response> AddShopAsync(AddFlowerShopRequest shopRequest, string email)
{
    try
    {
        var user = await _dbContext.Users.FirstOrDefaultAsync(u => u.Email!.ToLower() == email.ToLower());

        if (user == null)
            return new Response { Status = StatusEnum.Failure, Message = "User not found" };

        var flowerShop = _mapper.Map<Database.Entities.FlowerShop>(shopRequest);
        var shopConfig = _mapper.Map<ShopConfig>(shopRequest);

        flowerShop.AgreementID = (await _dbContext.Agreements.FirstOrDefault(a =>
a.Name.Equals("Standart")))!.Id;
        flowerShop.OwnerID = user.Id;
        flowerShop.ShopConfig = shopConfig;

        await _dbContext.FlowerShops.AddAsync(flowerShop);
        shopConfig.FlowerShopID = flowerShop.Id;
        await _dbContext.ShopConfigs.AddAsync(shopConfig);

        await _dbContext.SaveChangesAsync();

        return new Response<string> { Status = StatusEnum.Success, Data = flowerShop.Id };
    }
    catch (Exception ex)
    {
        return new Response { Status = StatusEnum.Failure, Message = ex.Message };
    }
}

/// <summary>
/// Качва снимка за обекта
/// </summary>
/// <param name="formData"></param>
/// <param name="objectID"></param>
/// <returns></returns>
public async Task<Response> UploadPictureAsync(IFormFile file, string objectID)
{
    try
    {
        var shop = await _dbContext.FlowerShops.FirstOrDefaultAsync(o => o.Id == objectID);

        if (shop == null)
            return new Response { Status = StatusEnum.Failure, Message = "Shops not found" };

        string directoryPath = "Files/Shop-Pictures";

        if (!Directory.Exists(directoryPath))
            Directory.CreateDirectory(directoryPath);

        var extension = Path.GetExtension(file.FileName);

        string uniqueFileName = $"{objectID}-{file.FileName}{extension}";
        string filePath = Path.Combine(directoryPath, uniqueFileName);

        var filesToDelete = Directory.GetFiles(directoryPath).Where(f => f.Contains(objectID));
```

```

    foreach (var fileToDelete in filesToDelete)
    {
        File.Delete(fileToDelete);
    }

    using (var stream = new FileStream(filePath, FileMode.Create))
    {
        await file.CopyToAsync(stream);
    }

    shop.PictureDataUrl = filePath;
    _dbContext.FlowerShops.Update(shop);
    await _dbContext.SaveChangesAsync();

    return new Response<string> { Status = StatusEnum.Success, Data = filePath };
}
catch (Exception ex)
{
    return new Response { Status = StatusEnum.Failure, Message = "The picture is not saved." };
}
}

```

Component

```

const AddShop: React.FC = () => {
    const { t } = useTranslation();
    const { cities } = useCities();
    const [selectedCity, setSelectedCity] = useState<City | undefined>(undefined);
    const { toast } = useToast()
    const { control, handleSubmit, formState: { errors }, setValue } = useForm<AddShopType>();
    const navigate = useNavigate();
    const [formData, setFormData] = useState<FormData>();
    const [previewUrl, setPreviewUrl] = useState<string[]>([]);
    const { city, isCityLoaded } = useCity();

    const onSubmit = async (data: AddShopType) => {
        try {
            const result = await addShop(data);
            uploadPicture(result.data);
            navigate('/');
        } catch (error) {
            showError(error, toast, t);
        }
    };

    const uploadPicture = async (shopID: string) => {
        if (formData == null) {
            return;
        }
        try {
            const urlWithQueryParam = '/FlowerShop/upload-shop-picture?shopID=' + shopID;
            await axiosAuthClient.post(urlWithQueryParam, formData, { headers: { "Content-Type":

```

```

"multipart/form-data" } }));
    }
    catch (error) {
        showError(error, toast, t);
    }
}

const handleShopAdded = (newShop: FlowerShop) => {
    const { latitude, longitude } = newShop;
    setValue('latitude', latitude);
    setValue('longitude', longitude);
};

async function onPictureChanged(e: ChangeEvent<HTMLInputElement>) {
    e.preventDefault();

    if (e.target.files == null)
        return;

    let newformData = new FormData();

    newformData!.append('file', e.target.files[0]);

    setFormData(newformData);

    setPreviewUrl(() =>
        Array.from(e.target.files ?? []).map((f) =>
            window.URL.createObjectURL(f)
        ));
}

function handleCitySelect(cityID: string) {
    if (cityID != undefined) {
        const foundCity = cities.find((c) => c.id === cityID)

        if (foundCity != undefined) setSelectedCity(foundCity)
    }
}

return (
    <div className="flex overflow-hidden min-h-[700px] justify-center mt-5" >
        <div className='grid grid-cols-1 md:grid-cols-3 rounded-lg space-x-5 p-5 bg-secondary-
background dark:bg-secondary-background-dark'>
            <div className="flex flex-col
min-h-[500px] h-screen md:h-auto text-black dark:text-gray-300">
                <div className='space-y-2'>
                    <h2 className='text-center'>{t('Add shop')}</h2>
                    <form className='space-y-5' onSubmit={handleSubmit(onSubmit)}>

```

```

<div className='grid grid-cols-2 place-items-start items-center space-x-2'>
  <label className='text-left'>{t('City')}</label>
  <Controller
    control={control}
    name='cityID'
    rules={validationRules.object}
    render={({ field: { onChange, onBlur } }) => (
      <CitySelect
        defaultCity={city}
        cities={cities}
        onChange={(event) => {
          onChange(event);
          handleCitySelect(event as string);
        }}
        onBlur={onBlur}
        className='col-start-2'></CitySelect>
      )>
    {errors.cityID && <span className="error-message col-span-
2">{t(errors.cityID.message || '')}</span>}
  </div>
  <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
    <label className='text-left'>{t('Name')}</label>
    <Controller control={control}
      name='name'
      rules={validationRules.firstName}
      render={({ field: { onChange, onBlur } }) => (
        <Input onChange={onChange} onBlur={onBlur} placeholder={t('Name')} />
      )>
    {errors.name && <span className="error-message col-span-
2">{t(errors.name.message || '')}</span>}
  </div>
  <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
    <label className='text-left'>{t('Address')}</label>
    <Controller control={control}
      name='address'
      rules={validationRules.address}
      render={({ field: { onChange, onBlur } }) => (
        <Input onChange={onChange} onBlur={onBlur} placeholder={t('Address')} />
      )>
    {errors.address && <span className="error-message col-span-
2">{t(errors.address.message || '')}</span>}
  </div>
  <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
    <label className='text-left'>{t('Delivery price')}</label>
    <Controller control={control}
      name='price'
      rules={validationRules.price}
      render={({ field: { onChange, onBlur } }) => (

```



```

                <Input onChange={onChange} onBlur={onBlur} placeholder={t('Delivery
price'))} />
            )) />
            {errors.price && <span className="error-message col-span-
2">{t(errors.price.message || '')}</span>}
        </div>
        <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
            <label className='text-left'>{t('Free shipping over')}</label>
            <Controller control={control}
                name='freeDeliveryAt'
                rules={validationRules.price}
                render={({ field: { onChange, onBlur } }) => (
                    <Input onChange={onChange} onBlur={onBlur} placeholder={t('Free shipping
over'))} />
                )} />
            {errors.price && <span className="error-message col-span-
2">{t(errors.price.message || '')}</span>}
        </div>
        <div className='grid grid-rows-2 place-items-start items-center'>
            <label className='text-left'>{t('Work time')}</label>
            <div className='grid grid-cols-2 space-x-2'>
                <Controller control={control}
                    name='openAt'
                    rules={validationRules.hour}
                    render={({ field: { onChange, onBlur } }) => (
                        <Input onChange={onChange} onBlur={onBlur} placeholder={t('Open at')} />
                    )} />
                <Controller control={control}
                    name='closeAt'
                    rules={validationRules.hour}
                    render={({ field: { onChange, onBlur } }) => (
                        <Input onChange={onChange} onBlur={onBlur} placeholder={t('Close at')} />
                    )} />
                {errors.openAt && <span className="error-message col-span-
2">{t(errors.openAt.message || '')}</span> || errors.closeAt && <span className="error-message
col-span-2">{t(errors.closeAt.message || '')}</span>}
            </div>
        </div>
        <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
            <label className='text-left'>{t('Same day delivery till')}</label>
            <Controller control={control}
                name='sameDayTillHour'
                rules={validationRules.hour}
                render={({ field: { onChange, onBlur } }) => (
                    <Input onChange={onChange} onBlur={onBlur} placeholder={t('Same day
delivery till')} />
                )} />
            {errors.sameDayTillHour && <span className="error-message col-span-

```

```

2">{t(errors.sameDayTillHour.message || "")}</span>
</div>
<div className='grid grid-cols-2 place-items-start items-center space-x-2'>
  <div className='flex flex-row space-x-2'>
    {previewUrl.map((url) => (
      <img className='h-auto w-full' src={url} />
    ))}
  </div>
  <label htmlFor="files" className="w-full col-start-2 border border-slate-300
rounded-lg hover:bg-black hover:text-white cursor-pointer text-center">{formData == undefined
|| formData?.getAll.length == 0 ? t('Select image') : t('Change image')}</label>
  <input id="files" className='hidden' type="file" accept='.png, .jpg'
    onChange={(e) => onPictureChanged(e)} />
</div>
<div className='grid grid-cols-2 place-items-start items-center'>
  <div className='hidden'>
    <label className='text-left'>{t('Latitude')}</label>
    <Controller control={control}
      name='latitude'
      rules={validationRules.latitude}
      render={({ field: { onChange, onBlur } }) => (
        <Input onChange={onChange} disabled name='latitude' onBlur={onBlur}
placeholder={t('Latitude')} />
      )} />
    {errors.latitude && <span className="error-message col-span-
2">{t(errors.latitude.message || "")}</span>
  </div>
  <div className='hidden'>
    <label className='text-left'>{t('Longitude')}</label>
    <Controller control={control}
      name='longitude'
      rules={validationRules.longitude}
      render={({ field: { onChange, onBlur } }) => (
        <Input onChange={onChange} disabled name='longitude' onBlur={onBlur}
placeholder={t('Longitude')} />
      )} />
    {errors.longitude && <span className="error-message col-span-
2">{t(errors.longitude.message || "")}</span>
  </div>
  {errors.latitude && <span className="error-message col-span-
2">{t(errors.latitude.message || "")}</span> || errors.longitude && <span className="error-
message col-span-2">{t(errors.longitude.message || "")}</span>}
  </div>
  <Button className='w-full' type="submit">{t('Submit')}</Button>
</form>
</div>
</div>

```

```

        <div className="col-start-2 col-span-2 hidden md:block">
            <Map flowerShops={[]} height='100%' width='100%' allowPointSelection
onObjectAdded={handleShopAdded} selectedCity={selectedCity} />
        </div>
    </div>
</div >

);
};

```

export default AddShop

2. Визуализиране на всички букети

```

Controller

/// <summary>
/// Взима всички букети
/// </summary>
/// <returns></returns>
[AllowAnonymous]
[HttpGet]
public async Task<IActionResult> GetBouquets([FromQuery] string? cityID,[FromQuery] string? shopID)
{
    var result = await _bouquetService.GetBouquetsAsync(cityID, shopID);

    if (result.Status == StatusEnum.Failure)
        return NotFound(result);

    return Ok(result);
}

Service

/// <summary>
/// Взима всички букети в даден град
/// </summary>
/// <returns></returns>
public async Task<Response> GetBouquetsAsync(string cityID, string shopID)
{
    var bouquets = await _dbContext.Bouquets.Include(b => b.Pictures).Include(b => b.Colors).Include(b =>
b.Flowers).Include(b => b.FlowerShop).Where(b => b.Status == 0 && (b.FlowerShopID == shopID || b.FlowerShop.CityID
== cityID)).ToListAsync();

    if (bouquets == null || !bouquets.Any())
    {
        return new Response { Status = StatusEnum.Failure, Message = "Bouquets not found" };
    }

    return new Response<IEnumerable<BouquetDTO>> { Status = StatusEnum.Success, Data =
_mapper.Map<IEnumerable<BouquetDTO>>(bouquets) };
}

Component

return (
    <>

```

```

    <div className='flex flex-col w-[100%] items-center justify-center space-y-2 my-2 p-2 bg-secondary-
background dark:bg-secondary-backgroud-dark rounded-lg'>
      <div className='flex flex-col md:flex-row space-x-5 justify-center text-black dark:text-gray-300'>
        <span className='text-start font-bold'>{(bouquets != undefined ? bouquets.length : '?') + ' ' +
t('bouquets in ') + t(selectedCity?.name || city.name)}</span>
        <CitySelect
          defaultCity={city}
          cities={cities}
          onChange={(event) => {
            handleCitySelect(event as string);
          }}
          className='md:z-10'></CitySelect>
        <ColorSelect
          colors={colors}
          onChange={(event) => {
            handleColorsSelect(event as Color[]);
          }}
          className='md:z-10'></ColorSelect>
        <FlowerSelect
          flowers={flowers}
          onChange={(event) => {
            handleFlowersSelect(event as Flower[]);
          }}
          className='md:z-10'></FlowerSelect>
      </div>
      <Bouquets bouquets={filteredBouquets}></Bouquets>
    </div>
  </>
);

```

3. Нова поръчка

```
Controller

/// <summary>
/// Добавяне на обект
/// </summary>
/// <returns></returns>
[HttpPost]
public async Task<ActionResult> MakeOder([FromBody] MakeOrderRequest orderRequest)
{
    var email = Request.GetEmailFromAccessToken(_tokenHelper);

    Services.Models.Responses.Response result = null;

    if (string.IsNullOrEmpty(email))
    {
        //result = await _orderService.MakeAnonymousOrder(orderRequest);
        return Unauthorized();
    }
    else
    {
        result = await _orderService.MakeOrder(orderRequest, email);
    }

    if (result.Status == StatusEnum.Failure)
        return BadRequest(result);

    if (!orderRequest.HasDelivery)
    {
        await _notificationHelper.SentNotification(orderRequest.ShopId);
    }

    return Ok(result);
}

Service

public async Task<Response> MakeOrder(MakeOrderRequest orderRequest, string email)
{
    try
    {
        if (orderRequest == null || orderRequest.Bouquets == null || !orderRequest.Bouquets.Any())
        {
            return new Response { Status = StatusEnum.Failure, Message = "Invalid input" };
        }

        var user = await _dbContext.Users.Include(u => u.UserInfo).FirstOrDefaultAsync(u => u.Email!.ToLower() ==
email.ToLower());

        var order = new Database.Entities.Order();

        order.UserID = orderRequest.UserId;
        order.FlowerShopID = orderRequest.ShopId;
        order.Description = orderRequest.Description;
        order.Price = orderRequest.Price;

        var cart = new OrderCart();
        var bouquetdInCart = new List<CartBouquet>();

        foreach (var bouquet in orderRequest.Bouquets)
        {
            var bouquetFromDB = await _dbContext.Bouquets.FirstOrDefaultAsync(b => b.Id == bouquet.Bouquet.Id);

            if (bouquetFromDB != null)
```

```

    {
        var cartBouquet = new CartBouquet();
        cartBouquet.BouquetID = bouquetFromDB.Id;
        cartBouquet.Bouquet = bouquetFromDB;
        cartBouquet.Count = bouquet.Count;
        bouquetdInCart.Add(cartBouquet);
    }
}

var deliveryDetails = new DeliveryDetails();

if (orderRequest.HasDelivery)
{
    order.Status = (int)OrderStatusEnum.New;

    deliveryDetails.Address = orderRequest.Address;
    deliveryDetails.Type = (int)DeliveryDetailsType.ToAddress;
}
else
{
    order.Status = (int)OrderStatusEnum.NotPaid;

    deliveryDetails.Address = "";
    deliveryDetails.Type = (int)DeliveryDetailsType.FromStore;
}

deliveryDetails.PhoneNumber = orderRequest.ReciverPhoneNumber ?? user.PhoneNumber ?? "";
deliveryDetails.ReciverName = orderRequest.ReciverName ?? user.UserInfo.FirstName ?? "";
deliveryDetails.UserID = orderRequest.UserId;

_dbContext.DeliveryDetails.Add(deliveryDetails);

order.DeliveryDetailsID = deliveryDetails.Id;

cart.Bouquets = bouquetdInCart;
order.OrderCart = cart;
cart.Order = order;

_dbContext.Orders.Add(order);

order.OrderCart.OrderID = order.Id;

_dbContext.OrderCarts.Add(cart);
_dbContext.SaveChanges();

return new Response<string> { Status = StatusEnum.Success, Data = order.Id };
}
catch (Exception ex)
{
    return new Response { Status = StatusEnum.Failure, Message = ex.Message };
}
}

```

Component

```

return (
    <div className="flex overflow-hidden min-h-[500px] justify-center mt-5" >
        <div className='grid grid-cols-1 w-[100%] rounded-lg space-x-5 p-5 bg-secondary-
background dark:bg-secondary-backgroud-dark'>

```

```

<div className="flex flex-col min-h-[300px] h-screen md:h-auto text-black dark:text-gray-
300">

  <div className='space-y-2'>

    <h2 className='text-center'>{t('Order')}</h2>

    <label className="relative inline-flex items-center cursor-pointer ml-5">

      <input type="checkbox" className="sr-only peer" checked={hasDelivery}
onChange={() => setHasDelivery(!hasDelivery)} />

      <div className="w-11 h-6 bg-gray-200 rounded-full peer peer-focus:ring-4 peer-
focus:ring-blue-300 dark:peer-focus:ring-blue-800 dark:bg-gray-700 peer-checked:after:translate-x-
full peer-checked:after:border-white after:content-[''] after:absolute after:top-0.5 after:left-[2px]
after:bg-white after:border-gray-300 after:border after:rounded-full after:h-5 after:w-5
after:transition-all dark:border-gray-600 peer-checked:bg-blue-600"></div>

      <span className="ml-3 text-sm font-medium text-gray-900 dark:text-gray-
300">{hasDelivery ? t('With delivery') : t('Get it from the store')}</span>

    </label>

    <form className='space-y-5 space-x-5 grid grid-cols-1 md:grid-cols-2'
onSubmit={handleSubmit(onSubmit)}>

      {hasDelivery ? (

        <>

          <div className='m-5 space-y-5'>

            <div className='grid grid-cols-2 place-items-start items-center space-x-2'>

              <label className='text-left'>{t('City')}</label>

              <label className='text-left'>{t('Sofia')}</label>

            </div>

            <div className='grid grid-cols-2 place-items-start items-center space-x-2'>

              <label className='text-left'>{t('Address')}</label>

              <Controller control={control}

                name='address'

                rules={validationRules.address}

                render={({ field: { onChange, onBlur } }) => (

                  <Input onChange={onChange} onBlur={onBlur}

placeholder={t('Address')} />

                )} />

            </div>

          </div>

        </>

      ) : (

        <div className='m-5 space-y-5'>

          <div className='grid grid-cols-2 place-items-start items-center space-x-2'>

            <label className='text-left'>{t('City')}</label>

            <label className='text-left'>{t('Sofia')}</label>

          </div>

          <div className='grid grid-cols-2 place-items-start items-center space-x-2'>

            <label className='text-left'>{t('Address')}</label>

            <Controller control={control}

              name='address'

              rules={validationRules.address}

              render={({ field: { onChange, onBlur } }) => (

                <Input onChange={onChange} onBlur={onBlur}

placeholder={t('Address')} />

              )} />

          </div>

        </div>

      )}

    </form>

  </div>

```

```

        {errors.address && <span className="error-message col-span-
2">{t(errors.address.message || "")}</span>}
    </div>

    <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
        <label className='text-left'>{t('Preferred delivery hour')}</label>
        <Controller control={control}
            name='preferredTime'
            rules={validationRules.hour}
            render={({ field: { onChange, onBlur } }) => (
                <Input onChange={onChange} onBlur={onBlur}
placeholder={t('Preferred delivery hour')} />
            )} />
        {errors.preferredTime && <span className="error-message col-span-
2">{t(errors.preferredTime.message || "")}</span>}
    </div>
</div>

<div className='m-5 space-y-5'>
    <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
        <label className='text-left'>{t('Reciver name')}</label>
        <Controller control={control}
            name='reciverName'
            rules={validationRules.firstName}
            render={({ field: { onChange, onBlur } }) => (
                <Input onChange={onChange} onBlur={onBlur} placeholder={t('Reciver
name')} />
            )} />
        {errors.reciverName && <span className="error-message col-span-
2">{t(errors.reciverName.message || "")}</span>}
    </div>
    <div className='grid grid-cols-2 place-items-start items-center space-x-2'>
        <label className='text-left'>{t('Reciver phone number')}</label>
        <Controller control={control}

```



```

        name='reciverPhoneNumber'

        rules={{ ...validationRules.phoneNumber, validate: (value) =>
validationRules.phoneNumber.validate(validator.isMobilePhone(value, 'bg-BG')) }}

        render={({ field: { onChange, onBlur } }) => (
            <Input onChange={onChange} onBlur={onBlur} placeholder={t('Reciver
phone number')} />

        )} />

        {errors.reciverPhoneNumber && <span className="error-message col-
span-2">{t(errors.reciverPhoneNumber.message || "")}</span>}

    </div>

    <div className='grid grid-cols-2 place-items-start items-center space-x-2'>

        <label className='text-left'>{t('Description')}</label>

        <Controller control={control}

            name='description'

            rules={validationRules.description}

            render={({ field: { onChange, onBlur } }) => (

                <Input onChange={onChange} onBlur={onBlur}

placeholder={t('Description')} />

            )} />

            {errors.description && <span className="error-message col-span-
2">{t(errors.description.message || "")}</span>}

        </div>

    </div>

    </>

    ) : (

        <></>

    )

}

    <div className='space-x-5'>

        {hasDelivery ? (

            <span className='text-start font-bold'>{t('Delivery fee')} + ": " +
order.deliveryFee}</span>

```

```

    ): (
        <></>
    )
}

<span className='text-start font-bold'>{t('Total') + ": " + amount}</span>

<Button className='m-5 w-full' type="submit">{t('MakeOrder')}</Button>

</div>

</form>

</div>

</div>

</div >

);

```