



ВАРНЕНСКИ СВОБОДЕН УНИВЕРСИТЕТ "ЧЕРНОРИЗЕЦ ХРАБЪР"  
ФАКУЛТЕТ „СОЦИАЛНИ, СТОПНАСКИ И КОМПЮТЪРНИ НАУКИ“

---

Катедра "Компютърни науки"

ДИПЛОМНА РАБОТА

## **ХОТЕЛСКА СИСТЕМА ЗА НАСТАНЯВАНЕ ЗА УНИВЕРСИТЕТСКИ НУЖДИ**

**University Accommodation System**

Дипломна работа по придобиване на ОКС бакалавър"

Дипломант: Георги Любомиров Блажев

Специалност:

Информатика и Компютърни Науки

Фак. № 193010043

Научен Ръководител:

проф. д-р инж. Т. Бакърджиева

## **Съдържание**

<b>СПИСЪК НА ИЗПОЛЗВАНИТЕ ТЕРМИНИ И СЪКРАЩЕНИЯ</b>	<b>4</b>
<b>УВОД</b>	<b>6</b>
Актуалност на избраната тема	7
Предмет на изследване	8
Обект на изследване	8
Цел	8
Задачи	8
Методология	9
Очаквани ползи	9
<b>1. ГЛАВА ПЪРВА - ТЕОРЕТИЧНА ЧАСТ</b>	<b>9</b>
1.1. Въведение	9
1.2. Кратка история на уеб приложенията	10
1.3. Развитие на технологиите	11
1.4. Съществуващи решения	12
1.5. Анализ на стъпките за действие	12
1.5.1. Дефиниране на изискванията	13
1.5.2. Извършване на анализ	13
1.5.3. Проектиране на архитектура	13
1.5.4. План за управление	14
1.5.5. Проектиране на внедряването	14
1.5.6. Изпълнение на решението	15
1.5.7. Поддръжка	15
1.6. Методология на организация	15
1.6.1. Разработка разделена на итерации	16
1.6.2. Обратна връзка и гъвкавост	16
1.6.3. Сътрудничество	16
1.6.4. Планиране	16
1.6.5. Тестване	17
1.7. Резултати от анализ	17
1.7.1. Автоматизация на процесите	17
1.7.2. Анализ и отчетност	18

1.7.3. Организация	18
<b>2. ГЛАВА ВТОРА - РАЗРАБОТВАНЕ НА ХОТЕЛСКА СИСТЕМА ЗА НАСТАНЯВАНЕ ЗА УНИВЕРСИТЕТСКИ НУЖДИ</b>	<b>19</b>
2.1. Архитектура	19
Архитектура на клиента	21
Архитектура на сървъра	21
2.2. Сигурност	23
Автентикация	23
Оторизация	25
2.3. Технологии	26
Клиент	26
2.3.1.1. Angular	26
2.3.1.2. HTML5	27
2.3.1.3. SCSS	27
2.3.1.4. TypeScript	27
2.3.1.5. Visual Studio Code	27
Сървър	28
2.3.1.6. .NET 7	28
2.3.1.7. C# 11	28
2.3.1.8. ASP.NET Core	29
2.3.1.9. Entity Framework Core	29
База Данни	30
Хостинг	31
2.3.1.11. Клиент	31
2.3.1.12. Сървър	31
2.4. Функционални възможности	31
Начален екран	33
Потребители	38
Гости	41
Отчети	43
Разписание	45
Настройки	46

<b>ЗАКЛЮЧЕНИЕ И ПРЕПОРЪКИ</b>	<b>49</b>
<b>ИЗПОЛЗВАНИ ИЗТОЧНИЦИ</b>	<b>50</b>
<b>БЪДЕЩО РАЗВИТИЕ НА ПРОЕКТА</b>	<b>51</b>
1. Множество висши учебни заведения	51
2. Автоматизация на услугите	51
3. Лоялностни програми	51
<b>ПРИЛОЖЕНИЕ</b>	<b>52</b>
1. Програмен код за вход в системата	52
1.1. Клиентска част	52
1.1.1. HTML	52
1.1.2. SCSS	53
1.1.3. TypeScript	54
1.1.3.1. Код на компонента	54
1.1.3.2. Код на оторизиращата заявка	55
1.1.3.3. Код на прехващач на заявки	57
1.1.3.4. Код на пазача на заявки	59
1.1.3.5. Код на модула за оторизация	59
1.1.3.6. Код на пренасочването към модула за оторизация	60
1.2. Сървърна част	61
1.2.1. Модел за потребители	61
1.2.2. Интерфейс на бизнес логика за оторизация	62
1.2.3. Клас на бизнес логика за оторизация	62
1.2.4. Интерфейс на логика за JWT	64
1.2.5. Клас на бизнес логика за издаване на JWT	64
1.2.6. Интерфейс на хранилище за потребители	66
1.2.7. Клас на бизнес логика за хранилище на потребители	67
1.2.8. Интерфейс на бизнес логика за потребители	68
1.2.9. Клас на бизнес логика за потребители	69
1.2.10. Контролер за оторизация	72

# СПИСЪК НА ИЗПОЛЗВАНИТЕ ТЕРМИНИ И СЪКРАЩЕНИЯ

---

**Agile** - Методология за разработване на софтуер

**BA** - Бизнес Анализ

**SDLC** - Жизнен цикъл на разработка на софтуер

**Stakeholder** - Роля дефинирана в **Agile** манифеста, означава човек извън **Agile** екипа, които са замесени в разработката на продукта, или притежават ценна информация за успешната му разработка

**Sprint** - Спринт: определен период, обикновено продължаващ от една до две седмици, в който са дефинирани задачите за даденото време

**Клиент** - Консуматор на уеб приложение, като фронт енд клиент

**FE** - Front end, интерфейса на приложението, с което крайният потребител изпраща заявки към сървъра

**BE** - Back end, сървърът, който съдържа бизнес логиката и функционалността на приложението

**API** - Application programming interface - интерфейс, който позволява на две приложения да комуникират помежду си

**JSON** - Javascript Object Notation - формат, чрез който се осъществява връзката между програмни интерфейси

**SQL** - Structured Query Language - Език за правене на заявки към база данни

**RDBMS** - Система за управление на релационни бази данни

**ORM** - Object Relational Mapper - инструмент за преобразуване на обекти от база данни към обекти в кода

**Routing стратегия** - Стратегия на разделяне на пътищата във **FE** приложение с цел да се придобие ясно разделение между страниците

**Lazy loading** - Мързеливо зареждане, т.е. нещо да бъде заредено само когато е извикано, а не при първоначалното пускане на програмата

**HTTP** - HyperText Transfer Protocol - интернет протокола за обмяна на ресурси, който уеб приложенията използват

**XML** - Extensible markup language - маркиращ език, който може да се използва както за изграждане на структура на документи, така и за изграждане на структура на отговори от програмни интерфейси

**URI** - Unique resource identifier - уникален идентификатор на ресурс във формата на линк, например: api/users/123

**JWT** - **JSON WEB TOKEN** - стандарт за пренасяне на информация в криптирана форма, чието съдържание е в JSON формат.

**Bearer token** - Стратегия на автентикация, при която се изпраща **JWT** в **Authorization Header** на заявката заедно с думата “Bearer” преди него

**Waterfall** - Методология за създаване на софтуер, която изисква предварителен анализ и често се ползва като олицетворение на липса на гъвкавост

## УВОД

---

В България вече почти няма институция или сфера, които да не са засегнати, било то и в различни степени, от един от най - отявлените отпечатащи на нашето време - дигитализацията. Множество процеси, които хората са свикнали да извършват физически, присъствено и на ръка, вече биват дигитализирани и автоматизирани. Широката достъпност на електронните услуги носи масив от ползи, както на крайните потребители, така и на институциите, които ги реализират:

1. Улеснен достъп до услуги
2. Ускорено изпълнение на услуги
3. Отпадане на нуждата от физическо присъствие на лица в институциите
4. Синхронизация и чистота на данните
5. Проследимост и прозрачност
6. Повишена продуктивност

Уеб приложенията са един от пособите, които позволяват ускореното развитието към електронните услуги. Те дават достижимост до дигитални инструменти в рамките на секунди. Внедряването им е дълъг и сложен процес, който включва поредица от критични етапи, като дизайн, бизнес анализ, сигурност, бюджети, намиране на изпълнители на поръчката, както и поддръжка след въвеждане на електронната услуга в експлоатация от предвидените лица. Въпреки нелекото изпълнение на миграцията към дигитални приспособления и приложения, успешното преминаване към

такива спомага на институции и бизнеси за оптимизиране и улесняването на начините на работа и повишаване на продуктивността.

### **Актуалност на избраната тема**

В процеса на модернизиране, темата за дигитализация чрез уеб приложения е от изключително значение за развитието на институциите, технологиите и иновациите. В областта на образованието този процес става все по - ключов. Висшите учебни заведения, изправени пред предизвикателствата на дигиталната епоха и условия на световна пандемия, бяха принудени да се съсредоточат към обща дигитализация както на занятията си, така и на административните процеси, за да могат да поддържат високо ниво на образование. В този контекст, поради нуждата от постоянна свързаност между студенти и университети, уеб приложенията са най - логичното софтуерно решение за реализиране на тези дейности. Освен за дистанционно обучение, не по - маловажни са и вътрешните приложения на университета, които отговарят за настаняването на студенти в студентски общежития и помещения. Създаването на система, или модернизирането на стара такава, за управление на вътрешните процеси на университети, в частност настаняването на гости в помещенията на висшите учебни заведения, обещава оптимизиран процес за същевременното разпределение на стаите, заетостта им, отчети и издаване на фактури. Всичко това дава гъвкавост в достъпа, възможност за актуализация и интеграция с други системи. Същевременно, те предоставят платформа за анализ на данни, оптимизиране на ресурсите и по-ефективно решаване на проблеми. Именно



поради липса на все още наложен стандарт за решение за настаняване за университетски нужди, темата за такава система е актуална.

### **Предмет на изследване**

Предмет на настоящата дипломна работа е разработване на хотелска система за настаняване за университетски нужди в местни висши учебни заведения.

### **Обект на изследване**

Обект на изследването е Варненски Свободен Университет “Черноризец Храбър”. ВСУ е най - големият частен университет в България. Седалището му се намира в гр. Варна. С огромната си студентска база, висшето учебно заведение има нужните условия за направата на анализ за университетските нужди за настаняване.

### **Цел**

Цел на изследването е разработването и внедряването на интегрирана уеб система за университетско настаняване, която отговаря на съвременните изисквания за дигитална трансформация и предоставя оптимални решения за административните и образователни нужди на висши учебни заведения, като се фокусира върху ефективното управление на студентски помещения и настаняване.

### **Задачи**

- Да се дефинират нуждите и проблемите, които трябва да бъдат решени

- Да се направи анализ на изискванията, нужни за успешната реализация на хотелска система за настаняване за университетски нужди
- Да се дефинират технологии, дизайн и архитектура на системата
- Разработката на система за настаняване за университетски нужди

## **Методология**

Методиката за реализацията на изследването ще следва **Agile** принципи, които носят предимствата на гъвкавост, постоянна връзка с всички замесени страни (разработчици, отговорници по качество, анализатори, мениджъри на проекти), динамична разработка, позволяваща приспособяване при промяна на изисквания, разбивка на работата на периоди, прозрачност на задачите и изписването им върху табло.

## **Очаквани ползи**

Очаквана полза от реализацията на приложението е разработването на университетска система като уеб приложение, което да бъде готово за използване след минимална конфигурация от страна на разработчиците, за да може специализираните лица да могат да управляват частите предназначени за отдаване на гости.

# **1. ГЛАВА ПЪРВА - ТЕОРЕТИЧНА ЧАСТ**

---

## **1.1. Въведение**

**Уеб приложенията** са вид софтуерни приложения, които се достъпват през интернет, обикновено чрез уеб браузър. Това е възможно

с помощта на **HTTP (HyperText Transfer Protocol)** протокола, който е основата зад обмяната на информацията в уеб пространството, като той работи на принципа: **заявка - отговор**, т.е. клиент прави заявка, а сървърът му връща отговор. Приложенията дават на потребителите достъп до множество различни услуги в различни области като социални мрежи, електронна търговия, банкиране и финанси, административни услуги и още други. Широка поредица от похвати и стандарти се грижат за масовата интеграция на онлайн решенията. Иновациите в технологиите на разработване, облачните услуги и по - достъпната и евтина за обикновените разработчици инфраструктура, установените стандарти и протоколи за сигурност и милиардите потребители на интернет пространството създават благоприятна среда за бум на производството и консумирането на уеб софтуерни решения и внедряването им в ежедневието ни.

## **1.2. Кратка история на уеб приложенията**

Преди масовата експлоатация на уеб приложенията и резкия скок в развитието на интернет, програмите с по - сериозна функционалност са били инсталирани локално, обикновено на настолен компютър, без задължителна нужда от връзка с интернет и пренасяни с външни носители като дискове или дискети, докато уеб страниците са били статични документи без почти никаква интерактивност. При използване, потребителите правят заявка чрез клиентско приложение и получават отговор от сървър под формата на страница. В зората на уеб пространството, това е било сполучлив механизъм. През средата на 90-те години компанията “Netscape” пуска “JavaScript” - програмен

език, който позволява динамичност в страниците от страна на клиента, а в последствие уеб приложенията придобиват все по - голяма разпознаваемост като концепция, като “Java” е един от първите програмни езици, които се адаптират към разработване на уеб софтуер.

### **1.3. Развитие на технологиите**

Бързото развитие на интернет технологиите води до нов етап в софтуерното разработване, който бива наричан Web 2.0. Под шапката му седят уеб приложенията от “второ поколение”, придобили разпознаваемост в началото и средата на 2000-те години като социални мрежи, форуми, платформи за споделяне на медия и общуване. Някои от приложенията зародени през този етап са “YouTube”, “Facebook” и “Twitter”. Тези приложения се смятат за революционни, защото освен изключително широкото потребление, те са и основен фактор в по - нататъшното развитие на уеб технологиите. Пример за това създадените тогава рамки за разработване като “React” на “Facebook” или “Angular” на “Google”. Първоначално разработени от фирмите за вътрешно ползване, те биват пуснати като отворен код за използване от разработчици по целия свят, като потреблението им не стихва вече близо десетилетие. В днешно време технологиите продължават да се развиват с невероятна бързина и въпреки огромния пазар и набор от съществуващи решения, иновациите все още не стихват. Софтуерното инженерство става все по - достъпно и това води освен до невиджан досега пазар за професионалисти от тази сфера, но и до бум на предприемачество и частна дейност, тъй като дава възможност на хора

от различни области да се преквалифицират и да навлязат в различни етапи от живота си в света на технологиите.

#### **1.4. Съществуващи решения**

При проучване не бяха открити съществуващи публично известни решения, които да предоставят възможност за отдаване на хотелски части и общежития за настаняване. Технологичният институт на ВСУ “Черноризец Храбър” представи тяхна вътрешна система, която служи за основа за разработка. Хотелската система за настаняване за университетски нужди подобрява решението, служещо за основа, с подобрен и разбираем потребителски интерфейс, оптимизирани заявки към базата данни, въведени добри практики при писане на кода, подобрена сигурност и възможност за внедряване с минимално настройване.

#### **1.5. Анализ на стъпките за действие**

Реализацията и внедренията на софтуерни системи е отговорна и трудна задача, в която участват както разработчиците, така и институцията, която ще използва приложението след това. Важно е лицата, които разработват софтуера, да са запознати с инфраструктурата на висшето учебно заведение, в което ще бъде внедрена системата, за да могат правилно да конфигурират самите приложения. Въпреки това е възможно да възникне нужда от миграции на данни или пренаписване на стари системи на по - нови технологии, било то поради несъвместимост между нови и стари технологии или рискове за сигурността на приложението, базата данни, кода или цялата организация. За да се счита едно внедряване на софтуерна система за усъществувано

трябва да има ясна представа какво означава внедряване за всяка една институция поотделно. Въпреки това съществуват принципи за интеграция, които дават ориентир и основа.

#### **1.5.1. Дефиниране на изискванията**

За да може да бъде от полза едно приложение, неговите разработчици трябва да знаят какъв проблем трябва да бъде решен. За да се случи това, институциите, които ще използват софтуера, трябва да работят отблизо с лицата, които ще го пишат, за да е подсигурено, че внедрената система изпълнява всички нужни изисквания. Това включва ясно очертаване на действия, бизнес процеси и специфични за областта на работа особености. Всичко това е нужно, за да може приложението да бъде лесно за използване и с интуитивен интерфейс.

#### **1.5.2. Извършване на анализ**

След като изискванията са ясно очертани, разработчиците следва да направят анализ, който да доведе до проектиране на софтуер, който решава всички изложени проблеми и спазва изискванията, наложени от бизнеса. Важно е двете страни да достигат до компромисни варианти и както бизнесът насочва разработчиците към изисквания и регулации, така и инженерите да насочват бизнеса към адекватни софтуерни решения. Обикновено има човек седящ между двете страни - **ВА (Бизнес Анализатор)**, като той може да принадлежи на която и да е от двете страни.

#### **1.5.3. Проектиране на архитектура**

След одобрението на анализа, трябва да бъде създадена правилна

софтуерна архитектура, която да може да изпълни изискванията. Тя трябва предварително да предвижда различни сценарии за това как различните системи ще общуват помежду си и ще боравят с информацията. За показване на готови планове за архитектури се използват диаграми, които да визуализират тези връзки, за да могат да бъдат разбрани от всички замесени във внедряването на софтуера екипи.

#### **1.5.4. План за управление**

След одобрение на архитектурата, технологичният институт на висшето учебно заведение трябва да се събере и прецени възможни рискове, крайни срокове и да се предвидят всички възможни изходи. Важно е екипите да са подготвени за потенциални пречки, тъй като винаги изскачат проблеми в етапите на разработка, особено ако разработчиците се сблъскат с остарели системи и отнеме повече време от предвиденото, за да бъдат отстранени проблемите.

#### **1.5.5. Проектиране на внедряването**

Същинското внедряване е най - трудната и отнемаща време задача, тъй като включва в себе си различни подетапи:

- Процесно очертание
- Тестване на системата
- Методология
- Логистика

За да е успешна тази стъпка, трябва и предните стъпки да бъдат изпълнени, да не бъдат оставяни неизяснени подробности и въпроси, доколкото е възможно. При успешна системо внедряване, системите започват

да работят безаварийно и, най - важно, без загуба на данни при прехвърлянето им.

#### **1.5.6. Изпълнение на решението**

След успешна разработка и тестване на софтуера, той е готов да бъде въведен в експлоатация. На този етап може да започне обучението на персонала върху новата система.

#### **1.5.7. Поддръжка**

Важно е след успешна интеграция да се продължи наблюдение над въведената в експлоатация система както от персонала, който ще го използва, така и от разработчиците. Трябва да има оставен прозорец за връзка между двете страни, в случай, че възникват нови грешки, както и за обратна връзка.

### **1.6. Методология на организация**

При процес на внедряване на софтуерното решение за настаняване, се използва методологията **Agile**, която да бъде съчетана с **SDLC (Software Development Life Cycle)** - рамката, която дава дефинициите за същинската разработка на софтуерните решения. **Agile**, както се подразбира от името си (в превод: Гъвкав), е методология, която е подготвена за новости по всяко време на софтуерната разработка. При създаването си през началото на хилядолетието, **Agile** решава съществуващия тогава проблем, който е породен от доминираща по това време методология - **Waterfall**. Проблемът е именно липсата на гъвкавост, защото при **Waterfall** комуникацията с клиента обикновено се случва след анализ и представяне на готов продукт, което, при грешка в анализа например, би могло да доведе до нужда от допълнителна



работа за разработчиците. При гъвкавата методология за създаване на софтуер, този процес се елиминира чрез следните похвати:

#### **1.6.1. Разработка разделена на итерации**

При **Agile** обикновено работата е разделена на итерации наречени **Sprint**-ове. Най - често са разделени на две седмици и се съсредоточават върху по - малки откъси от функционалности, като има ясно определена и съгласувана цел, която всички замесени групи трябва да следват.

#### **1.6.2. Обратна връзка и гъвкавост**

Обръща се внимание на значението на постоянната обратна връзка от всички замесени страни при създаването и внедряването на едно софтуерно решение. Често тази обратна връзка води и до промяна в изисквания, които да трябва да бъдат преработени. Постоянната връзка дава чувство за сигурност, че всички бъдещи пречки ще бъдат решени своевременно.

#### **1.6.3. Сътрудничество**

Един от най - важните принципи на **Agile** е естественото сътрудничеството, което методологията поражда. Не е рядкост да се реализират срещи на множество екипи наведнъж (разработчици, отговорници по качество, бизнес анализатор, мениджъри на проекти, дизайнери, клиенти, анкетирани и т.н.), за да се подsigури продължително синхронизирането на информация между различните групи и ясната цел.

#### **1.6.4. Планиране**

За да има ясно заложена цел, която да бъде изпълнена в края на всяка

итерация, трябва преди това тя да бъде поставена. Това се случва в началото на всеки **sprint**. При така нареченото планиране отново се събират всички замесени екипи и се обсъждат технически възможности, досегашни постижения, прехвърля се останала от миналия период на работа, ако има такава, оценяват се усилията нужни за изпълнението на задачите и всеки човек, според попрището си, си назначава задачи, които трябва да свърши в даден срок и по приоритет.

### **1.6.5. Тестване**

Постоянното тестване е от изключително значение за успешно внедряване на софтуерни системи. Тъй като функционалностите се разработват на итерации, отговорниците по качеството правят постоянни тестове, както на новите функции, за да бъдат одобрени от тях, така и на вече въведени част от приложението, тъй като се случва нова част от кода да пречи на стара или да причинява непредвидени грешки, които не са били хванати в етап на разработка.

## **1.7. Резултати от анализ**

Въз основа на проведен анализ, с помощ от технологичния институт на ВСУ “Черноризец Храбър”, като университетски нужди са изложени следните изводи:

### **1.7.1. Автоматизация на процесите**

- Бързо и лесно създаване на резервации.
- Автоматично обновление за информация за заетост на стаи

### **1.7.2. Анализ и отчетност**

- Генериране на отчети за заетост, приходи и други показатели
- Проследяване за отворени сметки и заплащането им

### **1.7.3. Организация**

- Създаване, триене, редактиране и четене на
  - Потребители
  - Гости
  - Стаи
  - Легла
  - Материална база
  - Каси
  - Планове за плащане
  - Сметки
  - Фирми

## 2. ГЛАВА ВТОРА - РАЗРАБОТВАНЕ НА ХОТЕЛСКА СИСТЕМА ЗА НАСТАНЯВАНЕ ЗА УНИВЕРСИТЕТСКИ НУЖДИ

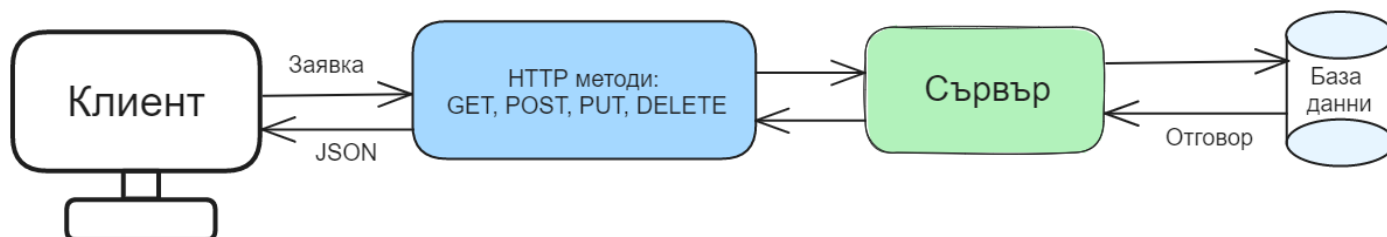
### 2.1. Архитектура



*Фиг. 1 - Архитектура на системата за настаняване*

На фигура 1 се вижда изобразена стандартна структура от тип “клиент - сървър”, на която е показано ясно разделение между сървъра или също частта, която съдържа бизнес логиката на приложението и осъществява връзката с базата данни, също наричана **BE (Back end)**, и клиентската част или също наричано **FE (Front end)**. Връзката между двете части на цялостното решение се извършва посредством **API**. За осъществяване на комуникацията между **FE** и **BE** се използва **JSON** формат. Самият програмен интерфейс следва **REST (Representational State Transfer)** конвенцията - фигура 2.

*Фиг. 2 - Архитектура на RESTful API*



**REST** е конвенция и съвкупност от напътствия, която служи като ориентир за изграждане на архитектура на уеб приложения в **BE** частта. Авторът ѝ се казва Рой Филдинг - известен учен в света на информационните технологии, съосновател и дългогодишен член на борда на директорите на софтуерната фондация "Apache". Той разработва **REST** архитектурата по време на докторантурата си. Тя има пет задължителни принципа:

- Връзка клиент - сървър - означава, че клиентът и сървърът са разделени от интерфейс, като всеки се грижи за неговите задачи без да се интересува от функционалността на другия. Например **FE** приложението не се интересува от бизнес логиката на **BE**, а е само изпраща заявки и консумира отговорите. Също така след изпращането на отговор, сървърът не се интересува от това как ще се използва той.
- Без състояние - английският термин е "stateless". Означава, че сървърът не запазва никаква информация за никакъв ресурс т.е. подобно на принципа на работа на крайните автомати имаме вход и изход, като никъде в кода не се пази никаква информация.
- Възможност за кеширане - клиентът може да кешира информацията, която идва от сървъра.
- Слоеве - клиентът не знае с кой слой от **BE** системата общува.
- Единен интерфейс - **FE** и **BE** частите са разделени чрез интерфейс и използват единен език за общуване - **XML** или **JSON**.

За да се счита един **API** за **RESTful**, той трябва да следва горепосочените принципи в унисон с **HTTP** принципите. Самата архитектура определя данните като ресурси и те трябва да могат да бъдат достъпвани чрез **URL-и**.

Типът на данните, които се поддържат: **JSON** или други медиен тип. Основни **HTTP** методи: "GET", "PUT", "POST", "DELETE".

## Архитектура на клиента

При създаването на всяко софтуерно решение е нужно да се спазва предварително избрана архитектура, особено ако има ясна представа каква е ще е цялостната структура на приложението. Наложеният през годините стандарт гласи, че трябва да се следват принципи на модулност, логическо разделение и да се строи преизползваем код, при нужда. При клиента ще има ясно разделение на модули спрямо критерии на честота на преизползваемост и значимост. Добра практика е FE модулите да бъдат разделени на:

- Core - модул “ядро”, който отговаря за основните компоненти, от които обикновено ще съществува само една инстанция, като навигацията.
- Shared - модул на споделени компоненти, които биват използвани множество пъти и в другите модули, като бутони, форми, зареждачи и т.н.
- Страници - всяка страница ще бъде в свой собствен модул като всяка подстраница на тази страница ще бъде към него

По този начин задачите между различните части на приложението ще бъдат логически разделени и зависимостите между тях ще бъдат сведени до минимум, а по възможност - елиминирани изцяло.

## Архитектура на сървъра

За строенето на сървъра е използвана така наречената “трислойна архитектура”. Тя също следва принципи на модулност, като разделя приложението на три главни части:

- Презентационен слой - слой, който дава достъп на външния свят до приложението. В контекста на **REST API**, това се случва най - често чрез така наречените контролери. Контролерите са инструмента, който програмните интерфейси използват, за да излагат части от бизнес логиката към клиентите. Те седят на различни пътища и очакват определен тип **HTTP** методи.
- Бизнес слой - слой, който се грижи за бизнес логиката. Тук се съдържа функционалността на приложението, като най - често се използват интерфейс класове (в езиците, които използват обектно ориентирано програмиране и позволяват инжекция на зависимости), също наричани договори, като средство за абстракция между презентационния слой и функционалността. Това обикновено е най - сложната част за строене, тъй като областта на дейност на софтуера се определя от функционалността.
- Слой на данните - слой, който се грижи за връзката с базата данни, както и операциите с нея. Този слой не трябва да се занимава с логика, а трябва да вземе вече обработени от бизнес слоя входни данни или заявки и да извърши исканите действия в базата данни.

Тук също се спазва принципът за модулност - всеки по - нисък слой е абстракция над дадена функционалност и връзката между тях отново е препоръчително да става чрез абстракции или вътрешни интерфейси. По този начин по - високо седящите модули не се интересуват от имплементацията на по - долните и отново е постигнато логическо разделение между различните части на приложението. По този начин, например, може да бъде сменена

имплементацията на базата данни без контролерите и бизнес логиката да бъдат засегнати и да се налага да се правят промени и в тях.

**Фиг. 3 - Трислойна архитектура**



## 2.2. Сигурност

### Автентикация

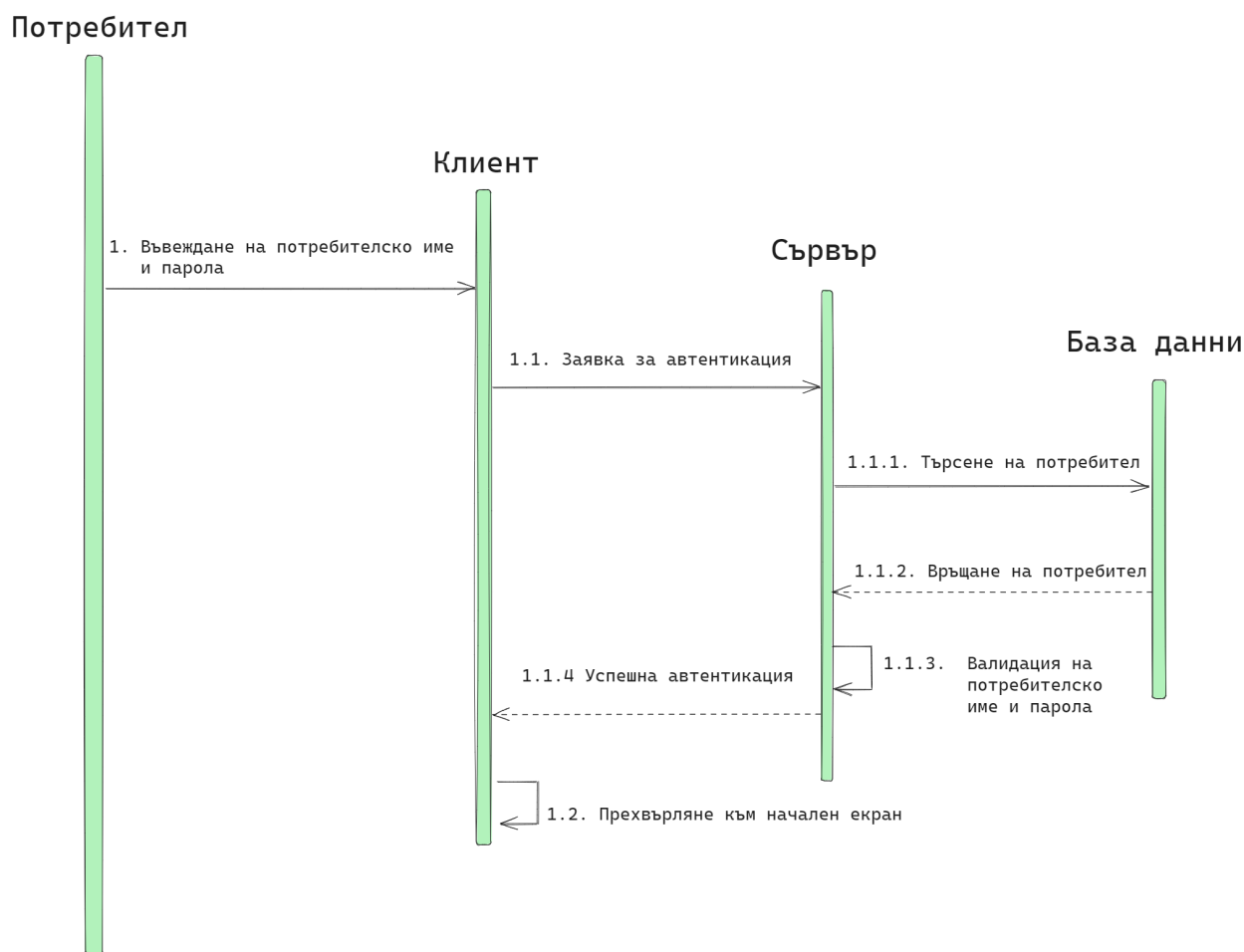
За автентикация приложението ще използва **Bearer** автентикация, при която се изпращат **JWT** издадени и подписани отново от същото приложение. Базата данни съдържа таблица за потребители, в която има колони за име и хеширани пароли. След като приложението установи, че потребителят съществува и неговите данни са правилни, приложението генерира и криптира **JSON Web Token-а** чрез алгоритъма “SHA 256”. Подписът се прави от генериран в кода сертификат за подписване. Самият токен съдържа така наречени твърдения, които служат за даване на повече информация при автентикация и оторизация за инстанцията, която го изпраща. Когато потребител успее да се автентикира пред приложението, клиентът ще получи такъв токен със следните твърдения:

- Sub - уникален идентификатор субектът, който е поискал токена, в случая ще е неговия идентификационен номер
- Username - потребителското име
- Email - електронната поща на потребителя



- Roles - ролите към които потребителят е записан
- Valid From - дата и час на изкарване на токена
- Valid To - дата и час, до които е валиден токена
- Authority - инстанцията, която изкарва токена
- Audience - инстанцията, която ще използва токена

От страна на клиента самите **JWT** ще се кешират и пазят, като ще бъдат изпращани на всяка заявка от така наречен прехващач.



**Фиг. 4 - Процес на автентикация**

## Оторизация

След успешна автентикация, за оторизация ще се използва стратегия базирана на ролите на потребителя, които ще присъстват в клеймовете. Понастоящем съществуват две роли - “Администратор” и “Оператор”. Администраторите имат неограничени права, а операторите нямат достъп до всички прозорци в настройките, както и до потребителите на приложението.

*Фиг. 5 - Имплементация на автентикация и оторизация на сървъра*

```
builder.Services.AddAuthorization()
    .AddAuthentication(options =>
    {
        options.DefaultScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
        options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.Authority = "vfu-hotel";
        options.Audience = "vfu-hotel";
    });
```

За защита на самите пътища на програмния интерфейс, са използвани анотации за определяне това, коя роля да има достъп до дадения метод.

**Фиг. 6 - Защита на път в контролер**

```
[ApiController]  
[Authorize(Roles = "Admin")]
```

Това може бъде направено както на индивидуални методи, така и на цели контролери, както например е защитен контролерът за потребителите.

### **2.3. Технологии**

За разработване на системата е използван различен набор от технологии спрямо нуждите на приложението, като технологиите са разделени на следните категории:

#### **Клиент**

##### **2.3.1.1. Angular**

За **FE** частта на приложението е използвана рамката за разработване на **FE** приложения “Angular”, продукт с отворен код на “Google”. Една от най - предпочитаните технологии за разработване, известна със строгата си структура, изключително богата документация, голямата си общност от софтуерни инженери, множество библиотеки и установени стандарти за изграждане на клиентски решения. “Angular” се използва от световно известни компании, като едни от най - известните продукти, които са писани с негова помощ включват “Teams” на “Microsoft”, “Gmail” на “Google” и “PayPal” на “PayPal”.

#### 2.3.1.2. HTML5

**HTML5** е текущата версия на **HTML (HyperText Markup Language)**. Той е част от така наречените маркиращи езици, които са съвкупност от анотации оформящи цялостната структура на документ. Използва се разширението **.html** за файловете.

#### 2.3.1.3. SCSS

**SCSS** е вторият синтаксис на **SASS (Syntactically Awesome Stylesheet)**. Използва се заедно с **HTML**, като това позволява стилизиране и придаване на по - красив вид на създадената с маркиращия език структура на страницата. Всеки валиден **CSS3** файл е също валиден **SCSS** файл. Използва се разширението **.scss** за файловете.

#### 2.3.1.4. TypeScript

**TypeScript** е програмен език с отворен код създаден от “Microsoft”. Той добавя силно изписване на типовете на променливите, за разлика от “JavaScript”, което дава допълнителна безопасност и предупреждения още преди компилиране на кода. Всеки **TypeScript** код се компилира в “JavaScript”.

#### 2.3.1.5. Visual Studio Code

**Visual Studio Code** е инструмент, разработен от “Microsoft” с отворен код, който служи за писане на код за клиентската част на приложението.

## Сървър

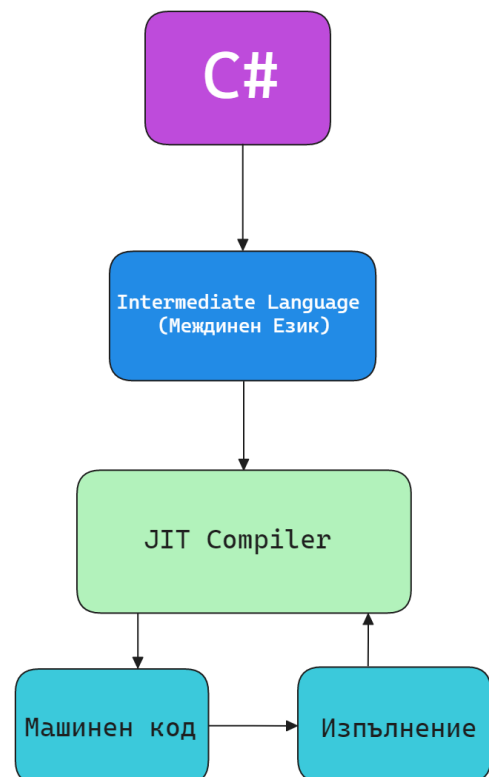
### 2.3.1.6. .NET 7

**.NET 7** е една от двете текущо поддържани версии на “.NET” рамката (Другата е **.NET 6**). Тя дава множество инструменти за разработване на сървърната част от софтуерното решение. Самата платформа е подходяща за всички главни операционни системи, като дава възможността на разработчиците да пишат и изпълняват код на “MacOS”, “Windows” и “Linux”. Включва множество пособия подходящи за всякакъв тип разработка и различни инженерни поприща: WEB, AI, Desktop, и т.н..

### 2.3.1.7. C# 11

**C# 11** е версията на езика съвместима с **.NET 7**. Отново продукт с отворен код на “Microsoft”, с изключителна прецизност и грижовност към безпрепятствената работа на инженерите. Езикът позволява използването на множество парадигми на програмиране: ООР, функционално програмиране, императивно програмиране, декларативно програмиране, процедурно програмиране. Използва се силно изписване на типовете на променливите по подразбиране, което дава чувство за безопасност, тъй като грешка би била забелязана още при компилиране на кода. За да

**Фиг. 7 - Компилиране**



се използва езикът е нужно да средата на разработка да има инсталиран така наречения “.NET runtime”, който е виртуалната машина, която изпълнява C# кода. Компиляторът се казва “Roslyn”. Стратегията на компилиране се казва **JIT (Just In Time) Compilation**, което означава, че при компилация на кода, той бива преобразуван в **IL (Intermediate Language)**, което е междинен език - между C# и машинен код и в последствие извиканите функции биват компилирани при изпълнение на машинен език - **binary**, още познат като двоичен код (фигура 7).

#### 2.3.1.8. ASP.NET Core

**ASP.NET**, идващо от Active Server Pages, е рамка за разработване на приложения, обикновено **BE**, давайки възможността за реализирането сървърната част и програмния интерфейс на софтуерното решение. Самата рамка включва широка гама средства и библиотеки за създаване на уеб базирани решения. **ASP.NET Core** е последната версия на **ASP.NET** написана отначало, която предоставя възможност за писане на многоплатформен код, като за разработка на хотелската система за настаняване, е използван **Web API** шаблон, който е използван за създаване на нашият **API** следвайки **REST** конвенцията за създаване на цялостни сървърни решения, концептуално приемайки данните за ресурси.

#### 2.3.1.9. Entity Framework Core

**Entity Framework Core (EF)** е **ORM (Object Relational Mapper)**, който позволява преобразуване на резултати от заявка към базата данни към C# обекти, които могат да се използват директно в кода. Инструментът дава абстракция над кода за достъп до базата. Изключително подходящ е за работа

с релационни данни, тъй като връзките биват поети от самата библиотека, позволявайки на разработчиците да се съсредоточат върху правилното изписване на бизнес логиката на приложението.

### **2.3.1.10. Среда за разработка**

**Visual Studio** е използваната интегрирана среда за разработка на хотелската система за настаняване. Тя е средата по подразбиране за **C#** разработчици, но поддържа и множество други езици: VB.NET, C++, F#, Python и други. Има лесна интеграция с облачното решение на “Microsoft” - “Azure”, лесна разширимост, хиляди библиотеки (NuGet пакети), интерфейс и шаблони за контейнеризация и оркестрация на контейнери.

### **База Данни**

За база данни на уеб приложението е използван **MySQL** - една от най - широко разпространените системи за управление на релационни бази данни. Тя е безплатна и с отворен код. За заявки и манипулация на данни тя използва **SQL**. Една от най - надеждните системи за управление на данни, тя също е многоплатформена, като не ограничава до дадена операционна система. Интеграцията с различни приложения е изключително улеснена и безпрепятствена за инженерите. **Entity Framework** има разработена библиотека за работа с **MySQL** бази данни.

## Хостинг

### 2.3.1.11. Клиент

За хостване на клиентското приложение е използвана онлайн платформата “Netlify”. Тя предоставя безплатен хостинг до трафик 100 GB на месец.

### 2.3.1.12. Сървър

За хостинг на сървъра е използван **IIS**. Това е безплатно решение за хостинг на “Microsoft”, който идва с лиценза към “Windows”.

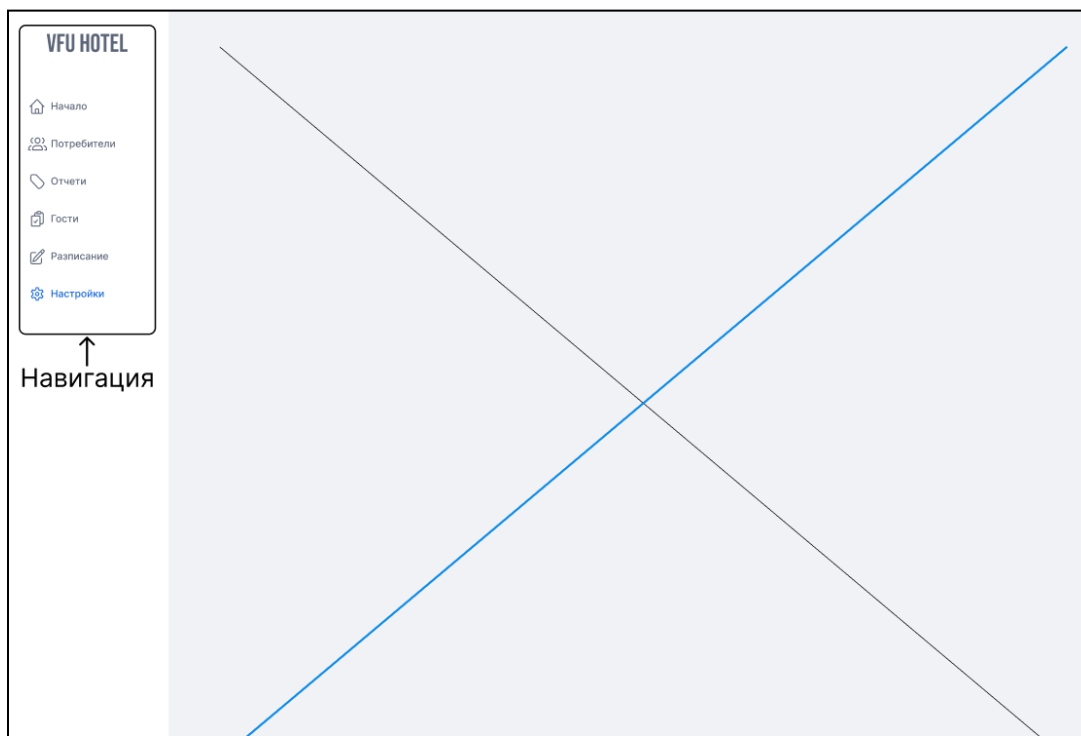
Дадените инструменти са използвани, защото са световно признати, безплатни, оптимизирани за разрастване на решенията при нужда и дават възможността за доброто използване на крайните продукти от потребители.

## 2.4. Функционални възможности

Целта на приложението е улеснено управление на хотелска система за настаняване на университетски нужди, при което висши учебни заведения отдават техни помещения на гости. Самият софтуер е предназначен да бъде използван от персонала на висшето учебно заведение с минимални или никакви затруднения. Използвани са стандартни похвати при строенето на структурата на страницата, за да може интерфейсът да е максимално лесен за ползване и интуитивен. Използват се похвати като **lazy loading** и разделяне на модули както при клиента, така и при базата данни, за да може цялостното изживяване на потребителя при използването на софтуера да бъде бързо.



**Фиг. 8 - Структура на интерфейс**



Тъй като **Angular** като рамка за разработване на **FE** приложения по подразбиране строи така наречените **SPA (Single Page Application)**, цялостната структура на приложението е непроменена. Тя е разделена на две части:

- Лява част - навигационно меню, чрез което потребителят може да навигира през различните страници на приложението. Пътищата са разделени чрез модуларно чрез така наречената които са както следва:
  - Начален екран
  - Потребители
  - Отчети
  - Гости
  - Разписание

- Настройки
- Дясна част - меню със съдържание, което държи съдържанието на самата страница, на база на избраната от лявото меню навигационна опция

Самото приложение, като технология, само рендира новото съдържание, което потребителят се опитва да достъпи, без да рендира цялата страница наново.

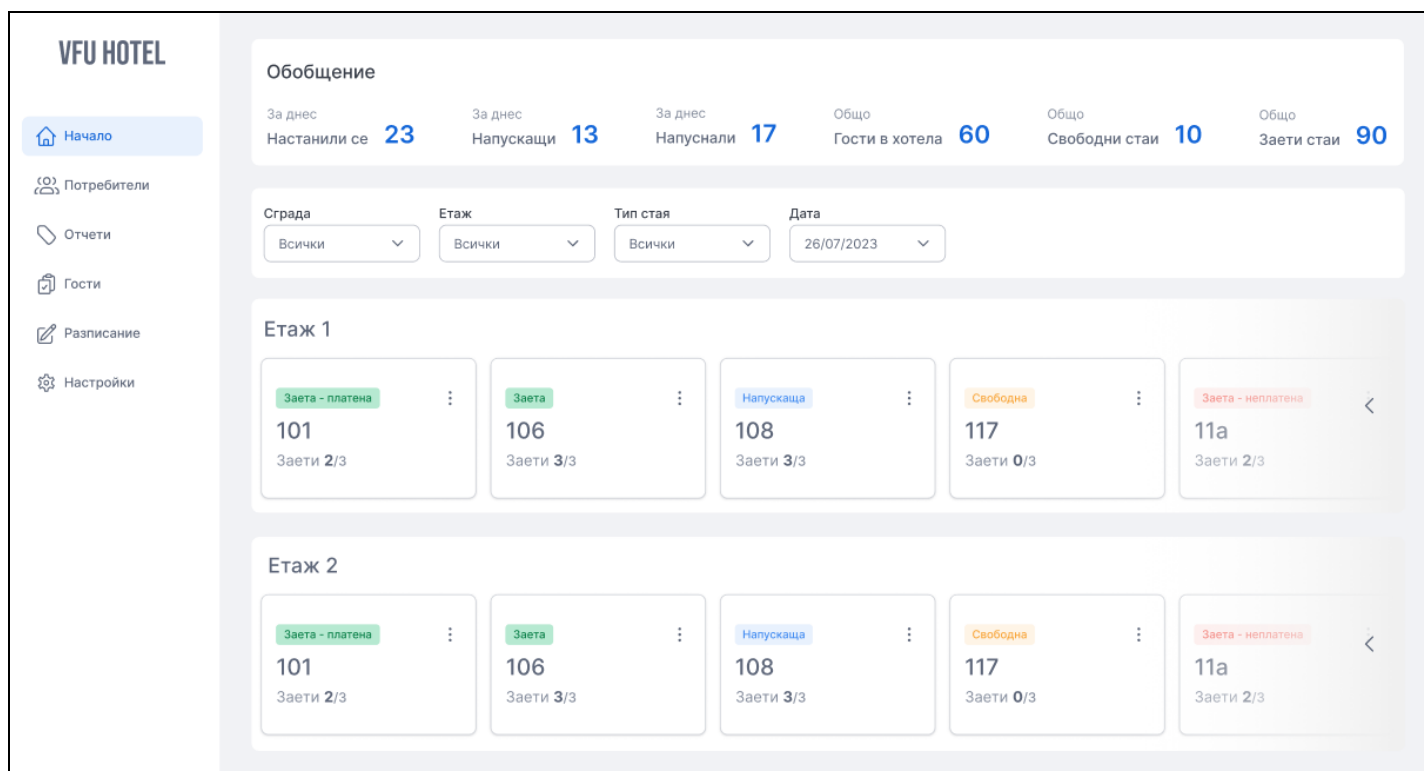
### **Начален екран**

За да е осъществимо е нужно да има ясно разпределение на стаите, престоят към тях, вида и състоянието на заетост. Стаите могат да бъдат филтрирани и групирани по:

- Сграда
- Етаж
- Тип
- Дата

Всяка стая може също да има един от следните статуси:

- Заета - платена
- Заета - неплатена
- Свободна
- Напускаща
- Резервирана
- Ремонт



**Фиг. 9 - Начален екран**

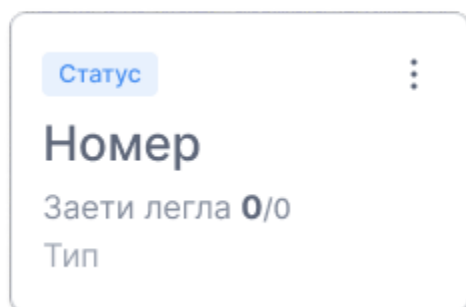
На началния екран е показано разделение на съдържателната част както следва:

- **Обобщение** - съдържа информация за днешен ден като настанили се, напускащи, вече напуснали, общо гости и т.н.
- **Филтър** - дава възможност на потребителя да филтрира стаите по различни критерии
- **Динамични елементи за етажи** - стаите от даден етаж ще бъдат показани на едно ниво, за да има ясно разграничение между тях

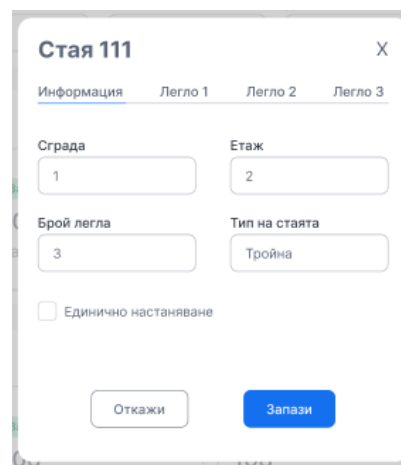
В елементите за етажи има така наречени елементи от тип “карта” на стая. Тя дава повече информация за конкретната стая като:

- **Статус на стая** - заета, свободна, напускаща и т.н.

- Номер - номера на стаята
- Легла - общ брой и застост
- Тип - апартамент, единична, двойна и т.н.



**Фиг. 10 - стая**

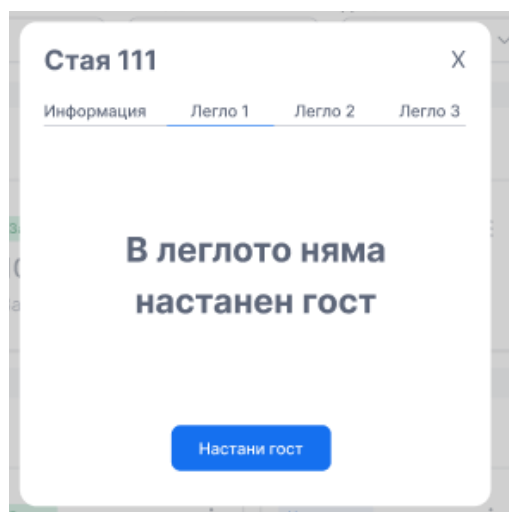


**Фиг. 11 - информация за стая**

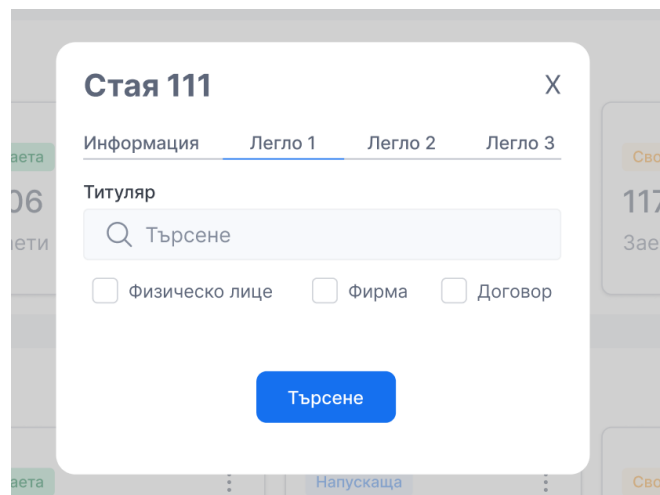
При натискане директно върху картата на стаята (фигура 10) от потребителя, се показва нов диалогов прозорец, който съдържа няколко подменюта (фигура 11). На първото меню се вижда информация за стаята, като сграда, етаж, брой легла, тип и има опция за единично настаняване на гост. Всяко от полетата може да бъде редактирано при нужда. Следващите менюта са динамични и броят им зависи от броя легла в стаята. При навигиране към следващ прозорец с легла са изведени два варианта. Първият е когато в текущото легло не е настанен гост. Тогава ще бъде изписано, че в леглото няма настанен гост и ще се покаже бутон, който при натискане дава възможност да се въведе информация за нов гост и да бъде настанен. Има търсачка, която търси вече регистрирани гости, които се добавят от страницата “Гости”. Ако търсачката не открие гост в базата, ще излезе прозорец, който да попита потребителя дали иска да добави госта в базата данни. Трябва да бъде отбелязано с

отметка дали ще се търси физическо лице, фирма или договор. При открит резултат информацията за госта се попълва на екрана. Оставащите стъпки са да се изберат дати на престой и ценови план. Крайната сума се изчислява автоматично. Следващият вариант е при вече настанен гост, тогава директно ще бъде показано, че леглото е заето и ще бъдат предоставени следните възможности - да бъде редактирана информацията за госта, гостът да напусне леглото, да бъде преместен в друга стая или да заплати престоя си, ако вече не е, като състоянието ще бъде показано вдясно от крайната сума. За да се потвърди плащане се натиска състоянието на плащането (ако е неплатено) и потребителят бива отведен към следващ прозорец, където се потвърждава заплащането на престоя му. При другите два бутона пътят е сходен: потребителят е отведен към съответен прозорец към функционалността, която иска да достъпи. При плащане на стаите се потвърждава крайната сума, потвърждава се, че гостът е платил и бива създадена от сървъра фактура в PDF формат, която се принтира след това. Номерата на фактурите се въвеждат от администратори в базата данни през прозореца за “Настройки”.

**Фиг. 12 - легло без настанен гост**



**Фиг. 13 - търсене на гост**



**Фиг. 14 - Намерен титуляр**

The screenshot shows a mobile application interface for a reservation system. At the top, there's a header 'Стая 111' with a close button 'X'. Below it are four tabs: 'Информация', 'Легло 1', 'Легло 2', and 'Легло 3'. The 'Информация' tab is active. The form contains several input fields: 'Титуляр' (Guest Name) with the value 'Георги Любомиров Блажев', 'Пристига' (Arrival) with 'Tue, Mar 2' and a calendar icon, 'Брой дни' (Number of days) with '2', 'Напуска' (Departure) with 'Tue, Mar 4' and a calendar icon, 'Цена' (Price) with a dropdown menu 'Ценови план', and 'Крайна сума' (Total amount) with '100.00 лв'. A blue button labeled 'Запази' (Save) is at the bottom.

**Фиг. 15 - Направена резервация**

This screenshot shows the same reservation form as Figure 14, but with additional elements indicating a confirmed reservation. The 'Крайна сума' field now has a red button labeled 'Неплатено' (Unpaid) next to it. At the bottom, there are two new buttons: 'Преместване' (Move) in orange and 'Напускане' (Check-out) in green.

The screenshot shows a dropdown menu with the title 'Статус' (Status). The menu is open, displaying three options: 'Потвърди плащане' (Confirm payment), 'Потвърди напускане' (Confirm check-out), and 'Блокирай' (Block).

**Фиг. 16 - падащо меню**

При натискане върху бутона за още опции (фигура 10) се спуска падащо меню (фигура 16), което дава възможността да се потвърди плащане, да се

потвърди напускане или да бъде блокирана стаята. Важно е да се спомене, че бутонът за падащото меню, както и опциите в него, ще бъдат видими, само ако са приложими за дадената стая, т.е. ако стаята в момента е празна и потребителят не е администратор, за да може да блокира стая, нито една от опциите няма да бъде приложима и съответно и бутонът няма да бъде видим.

## Потребители

Един от най - важните екрани в цялото приложение е този с потребителите. Трябва да се упомене, че тази страница е достъпна само за потребителите с роля на администратор и администраторски права, като за обикновените потребители с роля на оператор, тя дори няма да е видима в навигационното меню. Дори и потребител да се опита да влезе на нея без да има достъп, ще получи съобщение за грешка с **HTTP** статус **403 Forbidden**, което е изискване по **REST** архитектурата. Ако потребителят успее да влезе успешно в страницата, на нея ще може да се види таблица с всички съществуващи потребители и допълнителна информация за тях:

- Идентификационен номер
- Тип
- ЕГН
- Име
- Презиме
- Фамилия
- Потребителско име
- Електронна поща
- Права
- Дали потребителят е изключен

- Допълнително спускащо се меню за всеки потребител

Над таблицата се вижда по - малко меню с две опции - да се търси потребител (независимо по какъв критерий) и да се добавят нови потребители (фигура 17). При натискане върху бутона, за добавяне на нов потребител, подобно на началния екран, ще се покаже нов диалогов прозорец, на който ще бъде показана форма с полета за новия потребител, който текущият такъв трябва да въведе.

**Фиг. 17 - екран с потребители**

🔍

Търсене на потребители

Добави потребител

Номер	Тип	ЕГН	Име	Презиме	Фамилия	Потребителско име	Ел. поща	Права	Исключен	
#001	Администратор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	++	Не	⋮
#002	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#003	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#004	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#005	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#006	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#007	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#008	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#009	Оператор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	+-	Не	⋮
#010	Администратор	1234567890	Георги	Любомиров	Блажев	georgi	193010043@vfu.bg	++	Не	⋮

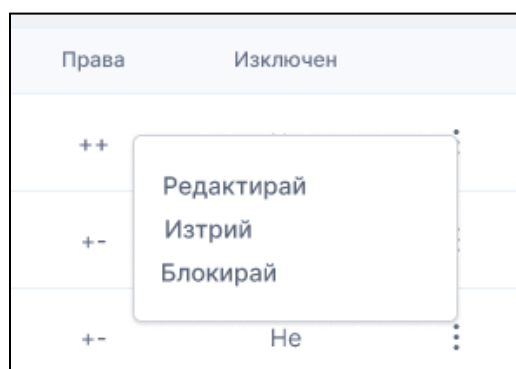


Ще могат да се въведат три имена, ЕГН, потребителско име, парола, електронна поща, тип на потребителя (Администратор или Оператор) и права за достъп. Сървърът прави проверка дали този потребител съществува по три критерия - потребителското име, електронния му адрес и ЕГН-то. Ако дори един от тях вече съществува

**Фиг. 18 - Добавяне на нов потребител**

в базата, няма да бъде разрешено създаването на потребителя. Ако няма такъв конфликт, потребителят бива успешно създаден. В менюто за допълнителните опции, което седи до всеки ред на потребител, могат да бъдат избрани три функции: да се редактира съществуващия потребител, да се изтрие или да бъде блокиран. Ако администратор реши, той може да изтрие дадения потребител изцяло от базата данни, или просто да го блокира, като по този начин ще попречи на бъдещо влизане в акаунта му. Ако се натисне върху опцията да бъде редактиран потребителя, ще изскочи отново диалогов прозорец с полета за въвеждане на информация. Полетата са същите като при създаване, с разликата, че оттук може да се смени парола на съществуващ потребител. Търсачката за потребители служи като филтър на потребителите, които виждаме.

**Фиг. 19 - Падащо меню с опции**



**Фиг. 20 - Редактиране на потребител**

The image shows a form titled 'Потребител 1' (User 1) with a close button 'X' in the top right corner. The form is divided into several sections:
 

- Информация** (Information): Fields for 'Име' (Name) with value 'Георги', 'Презиме' (Surname) with value 'Любомиров', 'Фамилия' (Family name) with value 'Блажев', and 'ЕГН' (EIN) with value '1234567890'.
- Оторизация** (Authorization): Fields for 'Потребителско име' (Username) with value 'georgi', 'Парола' (Password) with value '\*\*\*\*\*', and 'Тип' (Type) with a dropdown menu showing 'Администратор' (Administrator). There is a blue button 'Смени паролата' (Change password) next to the password field.
- Е-мейл** (Email): Field with value '193010043@vfu.bg'.
- Права за достъп** (Access rights): Two checkboxes, 'Оператор' (Operator) and 'Администратор' (Administrator), both of which are unchecked.

 At the bottom of the form are two buttons: 'Откажи' (Cancel) and 'Запази' (Save).

## Гости

За да може да бъде направена една резервация успешно, то гостът трябва да бъде записан като такъв в таблицата “Гости” (Guests) в базата данни. Страницата за гостите е достъпна от всички видове потребители, тъй като тя без нея е невъзможно да функционира приложението, защото при настаняване се изисква съществуващ запис на човек в базата. Когато потребител достъпи екрана, подобно на предния екран за потребителите, той ще види таблица с всички съществуващи гости, със следните колони:

- Номер
- Три имена
- ЕГН
- Факултетен номер
- Падащо меню с допълнителни опции

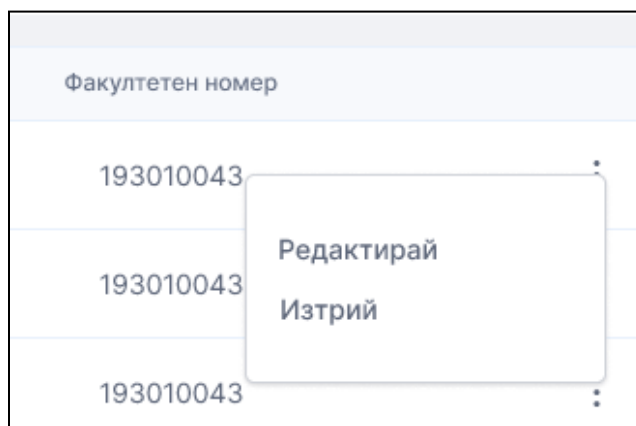
В падащото меню могат да бъдат намерени опции за редактиране и изтриване на госта. Над таблицата, отново, има меню с търсачка и бутон за създаване добавяне на гост.

**Фиг. 21 - Екран с гости**

Номер	Име	ЕГН	Факултетен номер	
#001	Георги Любомиров Блажев	1234567890	193010043	⋮
#002	Георги Любомиров Блажев	1234567890	193010043	⋮
#003	Георги Любомиров Блажев	1234567890	193010043	⋮
#004	Георги Любомиров Блажев	1234567890	193010043	⋮
#005	Георги Любомиров Блажев	1234567890	193010043	⋮
#006	Георги Любомиров Блажев	1234567890	193010043	⋮
#007	Георги Любомиров Блажев	1234567890	193010043	⋮
#008	Георги Любомиров Блажев	1234567890	193010043	⋮
#009	Георги Любомиров Блажев	1234567890	193010043	⋮
#010	Георги Любомиров Блажев	1234567890	193010043	⋮

Търсачката отново служи като филтър за гостите. При натискане върху бутона “Добави гост”, изскача прозорец с полета за информация за новия гост:

- Име
- Презиме
- Фамилия
- ЕГН
- Факултетен номер
- Пол
- Дата на раждане



**Фиг. 22 - Падащо меню**

- Държава
- Адрес
- Електронна поща
- Мобилен телефон
- Бележка
- Лична карта №
- Картинка
- Издадена от
- Дата на издаване
- VIP
- Нежелан гост

Сървърът ще провери за вече съществуващ гост на база ЕГН и факултетен номер и ще допусне създаването му ако не съществува такъв гост с тези данни

**Фиг. 23 - Добавяне на гости**

в базата. Когато потребител иска да редактира информацията за потребител, ще излезе отново диалогов прозорец със същите полета, разликата ще бъде, че заглавието на прозореца ще бъде “Редактиране на съществуващ гост”. Проверките за ЕГН и факултетен номер също присъстват тук.

## Отчети

За да може една институция, която отдава хотелски части, да изпълнява задълженията си и да има проследимост за заетостта, печалбата и производителността си, е задължително да има ясни отчети, които да могат да бъдат изваждани своевременно и по всяко време. Страницата “отчети” е

достъпна до всички видове потребители. Самият екран има лесен за разбиране интерфейс, който се състои от меню с филтри и таблица с резултати. Опциите за филтриране могат да бъдат по справка, начална и крайна дата. Справките сами по себе си могат да бъдат:

- Справка за дължими суми на факултети за период
- Разширена справка за дължими суми, факултети
- Опис на гостите по факултети
- Списък плащания за дата
- Всички гости за период
- Издадени фактури за дата
- Справка за брой реално заети легла
- Справка за брой продадени легла
- Справка плащания за дата (счетоводство)

The screenshot shows a web interface for selecting a report. It features a dropdown menu labeled 'Избери справка' (Select report) with the option '2. Разширена справка за дължими суми, факултети.' (2. Extended report for due sums, faculties) selected. To the right are two date input fields: 'От' (From) with the date '01.08.2023' and 'До' (To) with the date '31.08.2023'. A blue button labeled 'Търсене' (Search) is positioned on the far right.

**Фиг. 24 - Меню с филтри за отчети**

Таблицата с резултати е динамична и с различни колони за всеки вид справка, тъй като различните отчети връщат различни резултати, които да послужат на институцията ползваща софтуера да има прозрачност с финансовото представяне и заетостта на хотелските си части. Информацията от тези таблици е изцяло генерирана от системата т.е. няма и не би могло да има никаква намеса на потребител в този прозорец. Тази страница предоставя

информация, която може единствено да бъде четена, но не и манипулирана.

Избери справка

2. Разширена справка за дължими суми, факултети.

От

01.08.2023

До

31.08.2023

Търсене

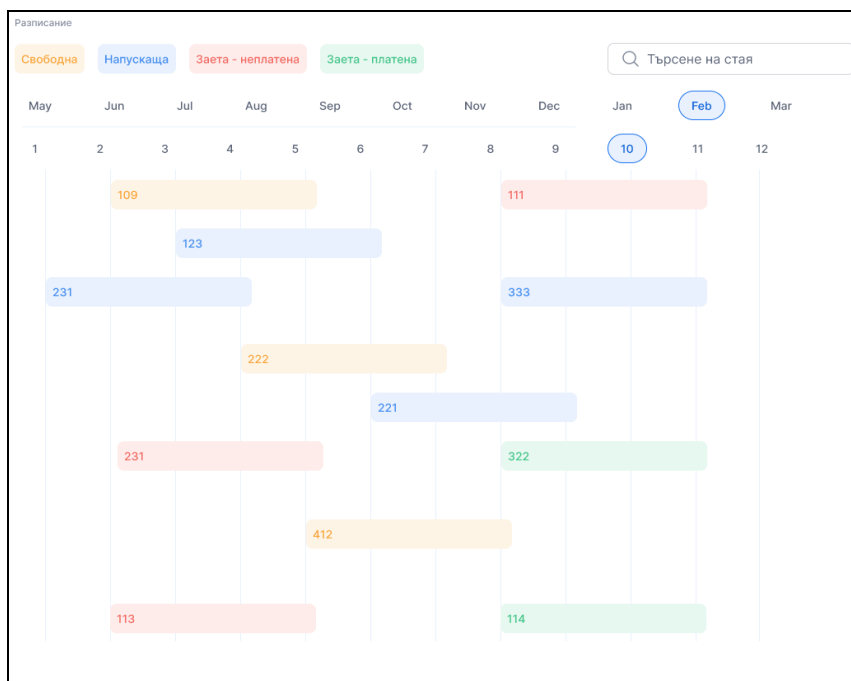
Име на факултет	Служебна	Апартамент	Трона	Двойна	Единична	Брой нощувки	Сума
Юридически	-	-	-	-	-	-	-
Юридически	-	-	-	-	-	-	-
Юридически	-	-	-	-	-	-	-
Юридически	-	-	-	-	-	-	-
Юридически	-	-	-	-	-	-	-

**Фиг. 25 - Таблица с резултати от отчети**

## Разписание

Страницата “Разписание” предоставя отново информация единствено за четене, която е достъпна от всички видове потребители, без значение ролята им. Тя предоставя разбивка на настаняванията по стаи, както и статуса им спрямо даден престой. С цел улеснение на потребителя, филтрите са над разбивката и могат да се контролират чрез движения на мишката. Има плъзгане наляво и надясно, което се случва с хващане с левия бутон на мишката и плъзгане в желаната посока. Има два такива плъзгача - за месеците и за дните от тях, като годините се сменят на база на плъзгането, а дните на база на текущия месец. Има легенда за статусите на стаите, както и търсачка, ако потребител иска да потърси конкретна стая в конкретния период. На база на цветовете от легендата, може да се види състоянието на всяка една стая в дадения период. Легендата е интерактивна и може да служи като допълнителен филтър, освен търсачката, за да се покажат всички стаи в това състояние през периода.

**Фиг. 26 - Разписания**

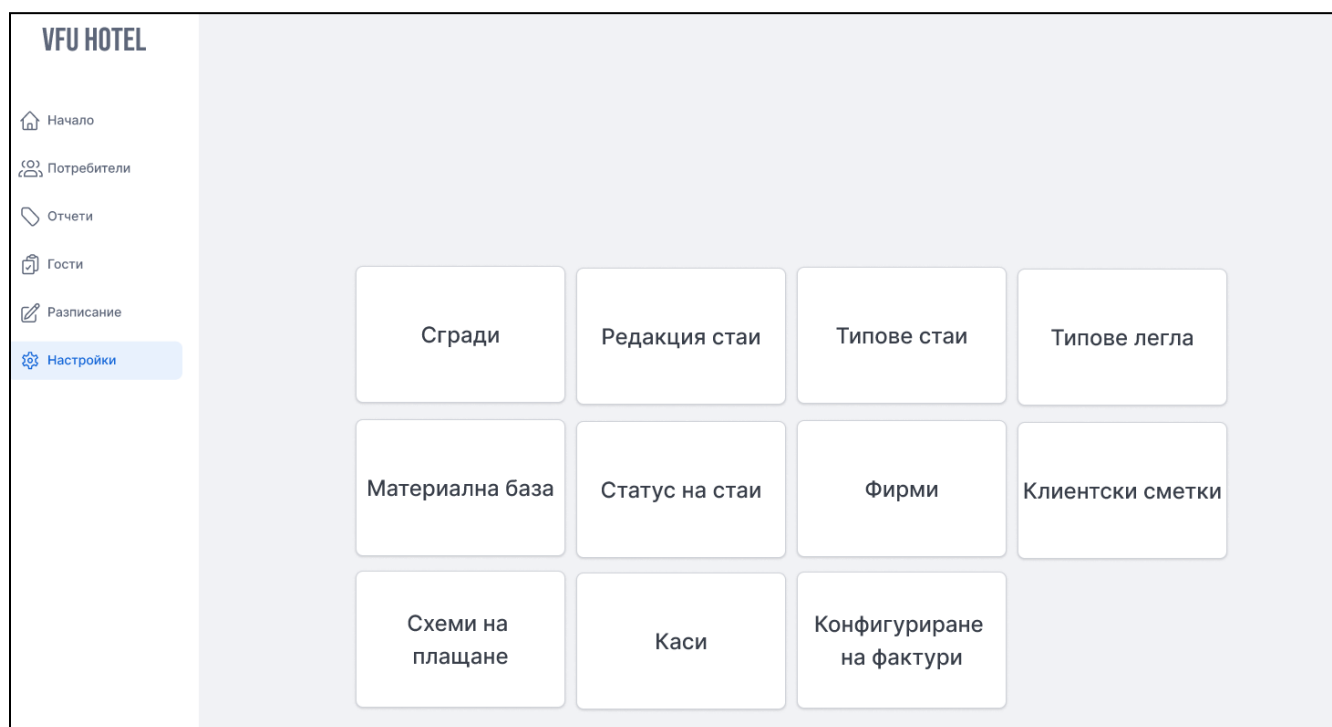


## Настройки

За да има завършеност една система, тя обикновено има нужда от допълнителна и честа конфигурация. Всяка институция има своите особености и рядко нещата са еднакви навсякъде. Затова е дадена възможност за допълнителни настройки от съответния екран. В страницата “Настройки” са зададени общо единадесет опции, които могат да се настройват:

- Сгради - служи за добавяне, редактиране и триене на сгради
- Редакция стаи - служи за добавяне, редактиране и триене на стаи
- Типове стаи - служи за добавяне, редактиране и триене на стаи
- Типове легла - служи за добавяне, редактиране и триене на легла
- Материална база - служи за добавяне, редактиране и триене на артикули за стаите

- Статуси на стаи - служи за добавяне, редактиране и триене на статуси на стая
- Фирми - служи за добавяне, редактиране и триене на фирми, които могат да са ползватели на стаите
- Клиентски сметки - служи за добавяне, редактиране и триене на клиентски сметки, като под клиентска сметка се има предвид всяко едно настаняване и плащането към него
- Схеми на плащане - служи за добавяне, редактиране и триене на платежни планове
- Каси - служи за добавяне, редактиране и триене на каси
- Конфигуриране на фактури - добавяне и редактиране на номера на фактури, които се използват при тяхното изваждане при плащане на престой



**Фиг. 27 - Екран с настройки**



Редно е да се спомене, че въпреки че общият брой опции са единадесет, пълният им брой ще бъде видим само от потребители с администраторски права, като потребител с операторски достъп ще вижда само пет от тях - статуси на стаи, фирми, клиентски сметки, схеми на плащания и каси. Интерфейсите на всяка от подстраниците е сходен с предните екрани, където имаме меню с бутони за добавяне и таблица на вече съществуващите неща.



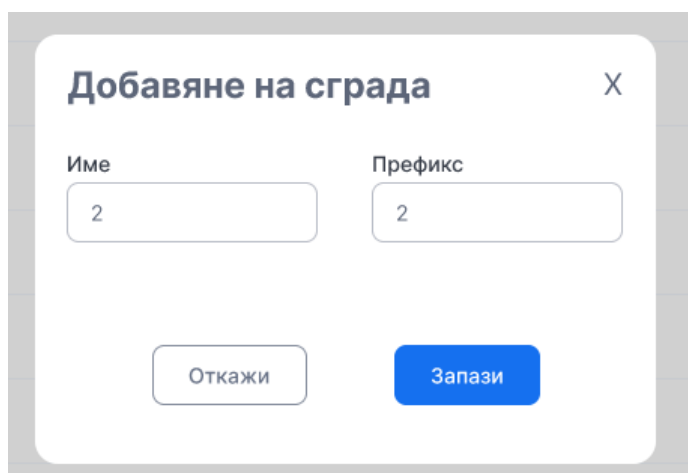
Номер	Име на сграда	Префикс	
#001	Сграда 1	P	⋮
#002	Сграда 1	P	⋮
#003	Сграда 1	P	⋮
#004	Сграда 1	P	⋮
#005	Сграда 1	P	⋮

**Фиг. 28 - Подстраница със сгради**

При натискане на бутон за добавяне изскача прозорец, чиито стойности са динамични на база на това, което е зададено като избор за създаване. (фигура 29).

**Фиг. 29 - Добавяне на сграда**

За редактиране и триене на данни, отново има падащо меню в края на всеки ред от таблицата, което при редактиране показва изскачащ прозорец, а при триене - изтрива данните, които са зададени.



### Добавяне на сграда

Име

Префикс

Откажи

Запази

## ЗАКЛЮЧЕНИЕ И ПРЕПОРЪКИ

---

Успешната дигитализация и внедряване на хотелска система за настаняване за университетски нужди улеснява работния процес, който висшите учебни заведения изпълняват всекидневно, като спомага за повишаване на производителността на персонала им като автоматизира процеси, които иначе трябва да се вършат на ръка, повишава прозрачността и дава ясна картина за положението на всички стаи както понастоящем, така и за изминали, и бъдещи периоди. Способността на системата да вади отчети, които не могат да бъдат манипулирани, както и ниската вероятност да бъдат допускани грешки, дават чувство на сигурност. Бъдещи ползи от системата за настаняване са внедряване на системата във всички филиали да даденото висше учебно заведение, както и синхронизация и внедряване с други системи на университета. Приложението може да се използва във всякакъв вид учебни заведения, които предлагат настаняване в хотелските си части. Лесно конфигурируемият софтуер дава улеснено внедряване на цялостната система.

Като препоръка може да се каже, че потенциалът на софтуерната не свършва дотук, а може да бъдат добавяни нови функционалности, като изпращане на известия при напускане както на администраторите на системата, така и на гостите. Може да бъде разработено допълнително мобилно приложение, чрез което гости да заявяват престой, както и по - обширни отчети от различно естество.

## ИЗПОЛЗВАНИ ИЗТОЧНИЦИ

---

Henderson, C. (2020). System integration process. AnyConnector. <https://anyconnector.com/system-integration/system-integration-process.html>

Beck, K. (2001). Manifesto for Agile Software Development. Agile Alliance. <https://agilemanifesto.org/iso/bg/manifesto.html>

Fielding, Roy Thomas (2000). Chapter 5: Representational State Transfer (REST) [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

Microsoft. (2022). Introduction to the .NET platform. <https://learn.microsoft.com/en-us/dotnet/introduction-to-dotnet>

Angular. (2022). Angular Documentation. <https://angular.io/>

TypeScript. (2023). Documentation. TypeScript Language. <https://www.typescriptlang.org/docs/>

Microsoft. (2023). C# documentation. Microsoft Docs. <https://docs.microsoft.com/en-us/dotnet/csharp/>

RESTful API. (2023). RESTful API. <https://restfulapi.net/>

Wikipedia contributors. (2022). Client–server model. Wikipedia. [https://en.wikipedia.org/wiki/Client%E2%80%93server\\_model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model)

DOJO Creative. (2020). Front End vs Back End of Your Website: Everything You Need to Know. <https://letsgodojo.com/front-end-vs-back-end/>

## **БЪДЕЩО РАЗВИТИЕ НА ПРОЕКТА**

---

Потенциалът за развитие на проекта е огромен и в множество насоки, някои от които са:

### **1. Множество висши учебни заведения**

Приложението може да поддържа различни институции, които имат нужда от такава система. В зависимост от това, дали даденото висше учебно заведение има възможността данните му да бъдат преобразувани, или не, ще подскаже дали ще може да се използва същата съществуваща база данни на приложението, или ще се направи нова инстанция специално за него със съответен домейн.

### **2. Автоматизация на услугите**

Внедряване на автоматични системи за контрол на климата, осветлението, както и преглед на стаите, които гостите могат да управляват чрез мобилното приложение или директно през системата за настаняване.

### **3. Лоялностни програми**

Интеграция на модул за програми за лоялност, където клиентите натрупват точки за всяка резервация, които след това могат да обменят за отстъпки или допълнителни услуги.

# ПРИЛОЖЕНИЕ

---

## 1. Програмен код за вход в системата

### 1.1. Клиентска част

#### 1.1.1. HTML

```
<div class="wrapper">
  <mat-card class="card">
    <mat-card-content class="content">
      <div class="login-window">
        
        <form [formGroup]="loginForm" (ngSubmit)="onSubmit()"
class="form">
          <mat-form-field appearance="outline">
            <mat-label>Потребителско име</mat-label>
            <input matInput formControlName="login"
placeholder="Enter your login">
            <mat-error
*ngIf="loginForm.get('login').hasError('required')">
              Потребителското име е задължително.
            </mat-error>
          </mat-form-field>

          <mat-form-field appearance="outline">
            <mat-label>Парола</mat-label>
            <input matInput type="password"
formControlName="password" placeholder="Enter your password">
            <mat-error
*ngIf="loginForm.get('password').hasError('required')">
              Паролата е задължителна.
            </mat-error>
```

```

        <mat-error
*ngIf="loginForm.get('password').hasError('minLength')">
        Password should be at least 6 characters long.
    </mat-error>
</mat-form-field>
    <button mat-raised-button type="submit" color="primary"
[disabled]="loginForm.invalid">Вход</button>
</form>
</div>
</mat-card-content>
</mat-card>
</div>

```

### 1.1.2. SCSS

```

.wrapper {
    display: flex;
    height: 100%;
    width: 100%;
}

.card {
    margin: auto;
    width: fit-content;
    padding: 3rem;
}

.logo {
    height: 4rem;
    margin-bottom: 2rem;
}

.content {
    display: flex;
}

.form {
    display: flex;
    flex-direction: column;

```

```

}

::ng-deep .message-bar {
  --mdc-snackbar-container-color: white;
  --mdc-snackbar-supporting-text-color: var(--card-title-color);
}

```

### 1.1.3. TypeScript

#### 1.1.3.1. Код на компонента

```

export class AuthComponent {
  loginForm = new FormGroup({
    login: new FormControl('', Validators.required),
    password: new FormControl('', [Validators.required])
  });

  constructor(public authService: AuthService, private _snackBar: MatSnackBar) {}

  onSubmit() {
    const formValue = this.loginForm.value;
    console.log(this.loginForm.value);

    const loginRequest: LoginRequest = { username: formValue.login, password:
formValue.password};
    this.authService.login(loginRequest).subscribe({
      next: (guest) => {
        this.authService.getAuthenticatedUser();
      },
      error: (error) => {
        if (error.status === 400) {
          this.openSnackBar('Грешно изписване на име или парола', 'Затвори')
        } else if (error.status === 401) {
          this.openSnackBar('Грешно име или парола', 'Затвори');
        }
      }
    })
  }
}

```

```

openSnackBar(message: string, action: string) {
  this._snackBar.open(message, action, {
    duration: 2000,
    panelClass: ['message-bar']
  });
}
}

```

### 1.1.3.2. Код на оторизиращата заявка

```

import { HttpClient, HttpResponseError } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { BehaviorSubject, throwError } from 'rxjs';
import { Observable } from 'rxjs';
import { catchError } from 'rxjs';
import { tap } from 'rxjs';
import { LoginResponse } from 'src/app/models/login-response.model';
import { LoginRequest } from 'src/app/models/login.model';
import { User } from 'src/app/models/users.model';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  private user$$: BehaviorSubject<User> = new BehaviorSubject<User>(null);
  public user$ = this.user$$$.asObservable();

  constructor(private http: HttpClient, private router: Router) {
    if(this.isLoggedIn()) {
      this.getAuthenticatedUser();
    }
  }
}

```



```

login(login: LoginRequest) {
  const url = 'https://localhost:7218/api/auth';

  return this.http.post<LoginResponse>(url, login).pipe(
    tap(response => {
      console.log(response.token);

      this.setToken(response.token);
      this.router.navigate([''])
    })
  );
}

getAuthenticatedUser() {
  const url = 'https://localhost:7218/api/auth';
  this.http.get<User>(url).subscribe(user => {
    this.user$.next(user);
  })
}

setToken(token: string): void {
  sessionStorage.setItem('auth_token', token);
}

logout() {
  sessionStorage.removeItem('auth_token');
}

isLoggedIn() {
  const token = sessionStorage.getItem('auth_token');
  if (token) {
    return true;
  }
  return false;
}

```

```

ngOnDestroy() {
  this.user$.next(null);
}
}

```

### 1.1.3.3. Код на прехващач на заявки

```

import { HttpClient, HttpResponseError } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { BehaviorSubject, throwError } from 'rxjs';
import { Observable } from 'rxjs';
import { catchError } from 'rxjs';
import { tap } from 'rxjs';
import { LoginResponse } from 'src/app/models/login-response.model';
import { LoginRequest } from 'src/app/models/login.model';
import { User } from 'src/app/models/users.model';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  private user$: BehaviorSubject<User> = new BehaviorSubject<User>(null);
  public user$ = this.user$.asObservable();

  constructor(private http: HttpClient, private router: Router) {
    if(this.isLoggedIn()) {
      this.getAuthenticatedUser();
    }
  }

  login(login: LoginRequest) {
    const url = 'https://localhost:7218/api/auth';

```

```

return this.http.post<LoginResponse>(url, login).pipe(
  tap(response => {
    console.log(response.token);

    this.setToken(response.token);
    this.router.navigate([''])
  })
);
}

getAuthenticatedUser() {
  const url = 'https://localhost:7218/api/auth';
  this.http.get<User>(url).subscribe(user => {
    this.user$.next(user);
  })
}

setToken(token: string): void {
  sessionStorage.setItem('auth_token', token);
}

logout() {
  sessionStorage.removeItem('auth_token');
}

isLoggedIn() {
  const token = sessionStorage.getItem('auth_token');
  if (token) {
    return true;
  }
  return false;
}

ngOnDestroy() {
  this.user$.next(null);
}
}

```

#### 1.1.3.4. Код на пазача на заявки

```
import { CanActivateFn, Router } from '@angular/router';
import { inject } from '@angular/core';

export const AuthGuard: CanActivateFn = (route, state) => {
  const token = sessionStorage.getItem('auth_token');
  const router = inject(Router);
  if (!token) {
    router.navigate(['/auth'])
    return false;
  }
  return true;
};
```

#### 1.1.3.5. Код на модула за оторизация

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AuthComponent } from './components/auth/auth.component';
import { AuthRoutingModule } from './auth-routing.module';
import { MatCardModule } from '@angular/material/card';
import { MatInputModule } from '@angular/material/input';
import { MatFormField } from '@angular/material/form-field';
import { ReactiveFormsModule } from '@angular/forms';
import { MatButtonModule } from '@angular/material/button';
import { MatSnackBarModule } from '@angular/material/snack-bar';

@NgModule({
  declarations: [
    AuthComponent
  ],
  imports: [
    CommonModule,
```

```

    AuthRoutingModule,
    MatCardModule,
    MatInputModule,
    ReactiveFormsModule,
    MatButtonModule,
    MatSnackBarModule
  ]
})
export class AuthModule { }

```

### 1.1.3.6. Код на пренасочването към модула за оторизация

```

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AuthComponent } from '../components/auth/auth.component';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  { path: '', component: AuthComponent },
]

@NgModule({
  declarations: [],
  imports: [
    CommonModule,
    RouterModule.forChild(routes)
  ]
})
export class AuthRoutingModule { }

```

## 1.2. Сървърна част

### 1.2.1. Модел за потребители

```
namespace VFUHotel.DAL.Entities;

public partial class User
{
    public int Id { get; set; }

    public sbyte? Type { get; set; }

    public string? Egn { get; set; }

    public string? Name1 { get; set; }

    public string? Name2 { get; set; }

    public string? Name3 { get; set; }

    public string? Login { get; set; }

    public string? Passwd { get; set; }

    public int PassLen { get; set; }

    public DateTime PassDate { get; set; }

    public string PassHist { get; set; } = null!;

    public string? Mail { get; set; }

    public string? Rights { get; set; }

    public int UnkId { get; set; }

    public sbyte StudyType { get; set; }
```

```

    public sbyte WhichSpecs { get; set; }

    public sbyte Disabled { get; set; }
}

```

### 1.2.2. Интерфейс на бизнес логика за оторизация

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VFUHotel.Models.Requests.User;
using VFUHotel.Models.Responses;

namespace VFUHotel.BLL.Services.Contracts
{
    public interface IAuthenticationService
    {
        Task<AccessToken> ValidateCredentials(LoginRequest loginRequest);
    }
}

```

### 1.2.3. Клас на бизнес логика за оторизация

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Authentication;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using VFUHotel.BLL.Exceptions;
using VFUHotel.BLL.Services.Contracts;
using VFUHotel.DAL.Entities;

```

```

using VFUHotel.Models.Requests.User;
using VFUHotel.Models.Responses;

namespace VFUHotel.BLL.Services
{
    public class AuthenticationService : IAuthenticationService
    {
        private readonly IUserService _userService;
        private readonly IJwtProvider _jwtProvider;

        public AuthenticationService(IUserService userService, IJwtProvider
jwtProvider)
        {
            _userService = userService;
            _jwtProvider = jwtProvider;
        }

        public async Task<AccessToken> ValidateCredentials(LoginRequest
loginRequest)
        {
            User user = await
_userService.GetByUsernameAsync(loginRequest.Username);

            var hashedPassword =
CalculateMD5Hash(loginRequest.Password).ToLower();

            return user.Password!.ToLower() == hashedPassword
                ? _jwtProvider.Generate(user)
                : throw new
InvalidCredentialsException(Constants.InvalidCredentialsMessage);
        }

        private string CalculateMD5Hash(string input)
        {
            byte[] inputBytes = Encoding.ASCII.GetBytes(input);
            byte[] hashBytes = MD5.HashData(inputBytes);

            StringBuilder sb = new StringBuilder();

```



```

        for (int i = 0; i < hashBytes.Length; i++)
        {
            sb.Append(hashBytes[i].ToString("X2"));
        }
        return sb.ToString();
    }
}

```

#### 1.2.4. Интерфейс на логика за JWT

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VFUHotel.DAL.Entities;
using VFUHotel.Models.Responses;

namespace VFUHotel.BLL.Services.Contracts
{
    public interface IJwtProvider
    {
        AccessToken Generate(User user);
    }
}

```

#### 1.2.5. Клас на бизнес логика за издаване на JWT

```

using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;

```

```

using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using VFUHotel.BLL.Options;
using VFUHotel.BLL.Services.Contracts;
using VFUHotel.DAL.Entities;
using VFUHotel.Models.Responses;

namespace VFUHotel.BLL.Services
{
    public class JwtProvider : IJwtProvider
    {
        private readonly JwtOptions _options;

        public JwtProvider(IOptions<JwtOptions> options)
        {
            _options = options.Value;
        }

        public AccessToken Generate(User user)
        {
            List<Claim> claims = GetClaims(user);

            SigningCredentials credentials = new(
                new SymmetricSecurityKey(
                    Encoding.UTF8.GetBytes(_options.SecretKey)),
                SecurityAlgorithms.HmacSha256);

            var token = new JwtSecurityToken(
                _options.Issuer,
                _options.Audience,
                claims,
                DateTime.UtcNow,
                DateTime.UtcNow.AddHours(1),
                credentials);

            string tokenValue = new JwtSecurityTokenHandler().WriteToken(token);

```

```

        return new AccessToken() { Token = tokenValue };
    }

    private List<Claim> GetClaims(User user)
    {
        List<Claim> claims = new()
        {
            new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new Claim(JwtRegisteredClaimNames.Email, user.Mail),
            new Claim(ClaimTypes.Role, Constants.Roles[user.Rights!])
        };

        return claims;
    }
}

```

### 1.2.6. Интерфейс на хранилище за потребители

```

using VFUHotel.DAL.Entities;

namespace VFUHotel.DAL.Repositories.Contracts
{
    public interface IUserRepository
    {
        Task<IEnumerable<User>> GetAsync();
        Task<User?> GetByIdAsync(int id);
        Task<User?> GetByEmailAsync(string Email);
        Task<User?> GetByUsernameAsync(string Username);
        Task<User> CreateAsync(User user);
        Task<User> UpdateAsync(User user);
        Task DeleteAsync(User user);
    }
}

```

### 1.2.7. Клас на бизнес логика за хранилище на потребители

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VFUHotel.DAL.Entities;
using VFUHotel.DAL.Repositories.Contracts;

namespace VFUHotel.DAL.Repositories
{
    public class UserRepository : IUserRepository
    {
        private readonly VfuHotelContext _context;

        public UserRepository(VfuHotelContext context)
        {
            _context = context;
        }

        public async Task<User> CreateAsync(User user)
        {
            await _context.Users.AddAsync(user);
            await _context.SaveChangesAsync();
            return user;
        }

        public async Task DeleteAsync(User user)
        {
            _context.Users.Remove(user);
            await _context.SaveChangesAsync();
        }

        public async Task<User?> GetByIdAsync(int id)
```

```

    {
        return await _context.Users.FindAsync(id);
    }

    public async Task<IEnumerable<User>> GetAsync()
    {
        return await _context.Users.ToListAsync();
    }

    public async Task<User> UpdateAsync(User user)
    {
        _context.Users.Update(user);
        await _context.SaveChangesAsync();
        return user;
    }

    public async Task<User?> GetByEmailAsync(string Email)
    {
        return await _context.Users.FirstOrDefaultAsync(user => user.Mail ==
Email);
    }

    public async Task<User?> GetByUsernameAsync(string Username)
    {
        return await _context.Users.FirstOrDefaultAsync(user => user.Login ==
Username);
    }
}
}

```

### 1.2.8. Интерфейс на бизнес логика за потребители

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

```

```

using System.Threading.Tasks;
using VFUHotel.DAL.Entities;
using VFUHotel.Models.Requests.User;

namespace VFUHotel.BLL.Services.Contracts
{
    public interface IUserService
    {
        Task<IEnumerable<User>> GetAsync();
        Task<User> GetByIdAsync(int Id);
        Task EditUser(int userId, EditUser editUser);
        Task<User> GetByUsernameAsync(string Username);
        Task<User> CreateAsync(CreateUser user);
        Task DeleteAsync(int id);
    }
}

```

### 1.2.9. Клас на бизнес логика за потребители

```

using System.Security.Cryptography;
using System.Text;
using VFUHotel.BLL.Services.Contracts;
using VFUHotel.DAL.Entities;
using VFUHotel.DAL.Repositories.Contracts;
using VFUHotel.Models.Requests.User;

namespace VFUHotel.BLL.Services
{
    public class UserService : IUserService
    {
        private readonly IUserRepository _userRepository;

        public UserService(IUserRepository userRepository)
        {
            _userRepository = userRepository;
        }
    }
}

```

```

public async Task<User> CreateAsync(CreateUser createUser)
{
    User user = new User()
    {
        Id = 0,
        Type = createUser.Type,
        Egn = createUser.Egn,
        Name1 = createUser.Name1,
        Name2 = createUser.Name2,
        Name3 = createUser.Name3,
        Login = createUser.Login,
        Passwd = CalculateMD5Hash(createUser.Passwd),
        PassLen = createUser.PassLen,
        Mail = createUser.Mail,
        Rights = createUser.Rights,
        PassHist = ""
    };

    return await _userRepository.CreateAsync(user);
}

public async Task EditUser(int userId, EditUser editUser)
{
    User user = await GetByIdAsync(userId);
    user.Type = editUser.Type;
    user.Egn = editUser.Egn;
    user.Name1 = editUser.Name1;
    user.Name2 = editUser.Name2;
    user.Name3 = editUser.Name3;
    user.Login = editUser.Login;
    user.PassLen = editUser.PassLen;
    user.Passwd = user.Passwd == editUser.Passwd ? editUser.Passwd :
CalculateMD5Hash(editUser.Passwd!);
    user.Mail = editUser.Mail;
    user.Rights = editUser.Rights;

    await _userRepository.UpdateAsync(user);
}

```

```

    }

    public async Task DeleteAsync(int Id)
    {
        User user = await GetByIdAsync(Id);
        await _userRepository.DeleteAsync(user);
    }

    public async Task<IEnumerable<User>> GetAsync()
    {
        return await _userRepository.GetAsync();
    }

    public async Task<User> GetByIdAsync(int Id)
    {
        return await _userRepository.GetByIdAsync(Id);
    }

    public async Task<User> GetByUsernameAsync(string Username)
    {
        return await _userRepository.GetByUsernameAsync(Username);
    }

    private string CalculateMD5Hash(string input)
    {
        using (MD5 md5 = MD5.Create())
        {
            byte[] inputBytes = Encoding.ASCII.GetBytes(input);
            byte[] hashBytes = md5.ComputeHash(inputBytes);

            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < hashBytes.Length; i++)
            {
                sb.Append(hashBytes[i].ToString("X2"));
            }
            return sb.ToString();
        }
    }

```



```
}  
}  
}
```

### 1.2.10. Контролер за оторизация

```
using Microsoft.AspNetCore.Authorization;  
using Microsoft.AspNetCore.Mvc;  
using VFUHotel.BLL;  
using VFUHotel.BLL.Services.Contracts;  
using VFUHotel.Models.Requests.User;  
  
namespace VFUHotel.Controllers  
{  
    [ApiController]  
    [Route("api/auth")]  
    public class AuthenticationController : ControllerBase  
    {  
        private readonly IAuthenticationService _authenticationService;  
        private readonly IUserService _userService;  
  
        public AuthenticationController(IAuthenticationService  
authenticationService, IUserService userService)  
        {  
            _authenticationService = authenticationService;  
            _userService = userService;  
        }  
  
        [HttpPost]  
        public async Task<IActionResult> LoginMember([FromBody] LoginRequest  
loginRequest)  
        {  
            return Ok(await  
_authenticationService.ValidateCredentials(loginRequest));  
        }  
    }  
}
```

```
[HttpGet]
[Authorize]
public async Task<IActionResult> GetAuthenticatedUser()
{
    var authenticatedUserId = int.Parse(User.Claims.FirstOrDefault(claim
=> claim.Type == Constants.SubClaim).Value!.ToString());
    var user = await _userService.GetByIdAsync(authenticatedUserId);
    return Ok(user);
}
}
```