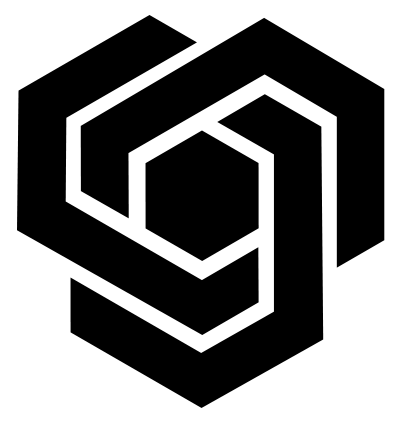
**Технически университет – София**

Факултет по Компютърни системи и технологии



Дипломна работа

Проектиране на уеб-базирана студентска система за връзка с преподавателите

Изготвил*:*

**Грета Петрова Петрова** – фак.№ 121219115

Научен ръководител*:*

**доц. д-р инж Явор Томов**

Увод

Системата “TU-Connect” е разработена с цел да опрости и подобри престоя на студентите по време на тяхното обучение в Техническия университет в София и да улесни взаимодействието на преподавателите с тях. Тази система не само решава единични проблеми, но и се справя с множество предизвикателства, които често възникват през академичната година. Студентите често се нуждаят от контакт с преподаватели поради различни причини - нужда от потвърждение на информация от трета страна, въпроси относно отсъствия, спешни срокове или съвместна работа по дипломни проекти. В такива случаи понякога е трудно да се свържат с преподавателя, тъй като той може да е зает с лекция или да не е в офиса си и последващо да забрави да върне обаждането поради заетостта и отговорностите на професията. Някои преподаватели предлагат само служебен телефон, други споделят общ телефон, а повечето не публикуват своя график на достъпност за студентите. Така, намирането на преподавател понякога се превръща в предизвикателство за студентите.

Системата “TU-Connect” предоставя решения за тези проблеми по няколко начина. На първо място, тя дава възможност на всеки преподавател да въведе своите лекции, упражнения и други ангажименти, които заемат времето му през седмицата. Това позволява на студентите да намерят преподавателя в определена зала или да изчакат завършването на неговия ангажимент, за да не го безпокоят по време на работа. Друга функция на системата е улесняването на организирането на часове за консултации. Всеки преподавател може да въведе информация за своите часове за консултации, включително периода, в който е наличен за дискусии на живо или онлайн, номера на стаята, началния и крайния час, както и допълнителни бележки, напрр линк за онлайн среща. Студентите имат достъп до тези данни и могат да заявят времеви блок за среща с преподавателя. Ако преподавателят одобри срещата, този времеви блок се блокира за други студенти, които търсят среща. Също така, има поле за указване на причината за срещата, което позволява на преподавателя да се подготви с необходимите материали или информация.

Освен това, системата “TU-Connect” предлага решение за улесняване на комуникацията с преподавателите в контекста на дипломните работи, като същевременно осигурява известна степен на анонимност. Студентите могат да публикуват своите идеи за дипломна работа, детайлно описвайки ги, и да ги предложат за разглеждане от преподаватели. Тези публикации са анонимни, за да се избегне лично предубеждение. Когато преподавател изрази интерес към някоя идея и предложи своята помощ като ръководител, системата автоматично изпраща имейл до студента с информация за преподавателя. Студентът може да прегледа всички предложения и да избере преподавателя, който му допада най-много. След приемане на предложението, системата изпраща имейл до преподавателя и разкрива контактите на двете страни, което позволява по-нататъшна комуникация извън системата. Всички други предложения се изтриват от системата. Този подход улеснява процеса на намиране на ръководител и дава възможност на студентите да предлагат множество идеи без притеснение, като същевременно им дава възможност да избягват нежелани преподаватели.

Съдържание

Увод2

Изложение разделено в глави5

Преглед на проблема по литературни данни5

Архитектура на приложението и избор на технологии6

REST архитектура6

Java8

Spring Framework9

HyperSQLDB10

Postman11

cURL12

Maven12

TypeScript13

Angular Framework13

Програмна реализация14

Изпращане на имейли15

Автентикация и JWT токени19

Преподаватели32

Студенти и дипломни работи35

Резервиране на среща с преподавател и графици43

Таблици и връзки между ентитита48

Прототипен потребителски интерфейс52

Заключение и бъдещи подобрения55

Източници56

Изложение разделено в глави

1. Преглед на проблема по литературни данни

При разработването на уеб-базираната студентска система за връзка с преподавателите, значително внимание трябва да се обърне към сравняването и анализа на съществуващите образователни платформи като Moodle и Microsoft Teams. Важно е да се разгледат техните ключови функционалности, предимства и ограничения, за да се идентифицират потенциални области за подобрение и иновация.

Moodle, като една от най-разпространените образователни платформи, предлага разнообразие от функции, включително споделяне на учебни материали, публикуване на съобщения и графици за изпити. Тази платформа се е доказала като надежден инструмент за управление на курсове и образователни ресурси. Въпреки това, се наблюдават някои ограничения, особено в областта на директната комуникация между студенти и преподаватели. Например, в Moodle липсва интегрирана система за бързо намиране на помощ от преподаватели или за организиране на консултации, което може да затрудни студентите при търсенето на подкрепа.

От друга страна, Microsoft Teams предоставя отлични възможности за организиране на срещи, благодарение на своя интегриран календар, който улеснява координацията на срещите и управлението на времето. Въпреки това, Teams основно разчита на чат за директна комуникация, което може да доведе до претоварване със съобщения и затруднения в получаването на бързи отговори от преподавателите. Това подчертава необходимостта от по-динамична и ефективна комуникационна платформа в образователния контекст.

В отговор на тези наблюдения, новата система трябва да се стреми към разработването на подобрени и интегрирани функции за комуникация и взаимодействие. Това включва разработването на интуитивен интерфейс за визуализация на графиците на преподавателите, което ще позволи на студентите лесно да идентифицират подходящи времена за срещи или консултации. Също така, важно е да се включи иновативна платформа за споделяне на идеи за дипломни работи, която да предлага възможност за анонимно участие. Тази функция би стимулирала креативността и свободата на изразяване сред студентите, позволявайки им да представят своите идеи без притеснение от лични съдби или предубеждения. В допълнение, системата може да включва модул, който позволява на преподавателите да преглеждат и оценяват предложените идеи, като по този начин се насърчава конструктивният диалог и обратната връзка.

Всичко това ще улесни студентите и преподавателите в ежедневната им работа и ще спомогне за по-ефективното управление на времето и ресурсите.

1. Архитектура на приложението и избор на технологии
2. **REST архитектура**

Архитектурата REST се характеризира с няколко ключови принципи: безсъстояние, униформен интерфейс (еднакъв начин на комуникация между компонентите) и разпределена система. Безсъстоянието означава, че всяка заявка от клиент към сървъра съдържа всички необходими информации за изпълнението ѝ, без да разчита на предишни заявки или сесии. Униформният интерфейс осигурява стандартизиран начин за достъп и манипулиране на ресурсите чрез уеб, което улеснява комуникацията между различни системи и устройства. Разпределената система позволява функционалностите на приложението да бъдат разпределени между различни сървъри и услуги, което подобрява мащабируемостта и надеждността. Тези характеристики правят REST архитектурата гъвкава, лесна за разширяване и удобна за използване в различни приложения.

Важна характеристика на REST е подходът към управлението на данните чрез концепцията за 'ресурси', които са представени като различни видове информация и функционалност в приложението. Всяка единица данни или 'ресурс' е идентифициран с уникален адрес, обикновено URL, и се манипулира чрез общоприети уеб методи като GET, POST, PUT и DELETE. Този подход позволява ефективно и интуитивно взаимодействие с данните в приложениетоРазгледайки модела Model-Service-Controller, той се вписва в REST архитектурата по следния начин:

* **Model (Модел)**: Този компонент отговаря за управление на данните и бизнес логиката. Моделът представлява структурата на данните, с които работи системата, като например записи в база данни или XML файлове.
* **Service (Услуга)**: Службите действат като посредник между модела и контролера. Те съдържат бизнес логиката и правилата за обработка на данните, като например валидация, транзакции и обработка на заявки. Услугите са отговорни за извършването на операции върху модела и предоставят интерфейс, който контролерите могат да използват.
* **Controller (Контролер)**: Контролерите са връзката между потребителския интерфейс и службите. Те обработват входящите HTTP заявки, извикват съответните услуги и връщат отговори. Контролерите са отговорни за обработка на заявките от клиента и преобразуването на резултатите от услугите в подходящ формат за отговор, като например JSON или XML.

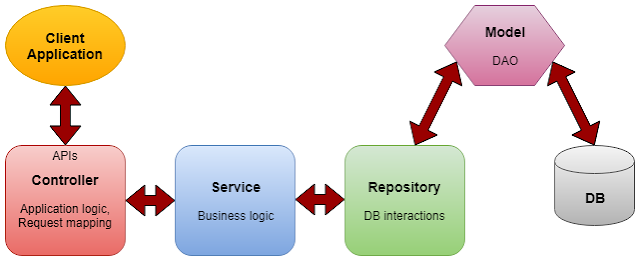


Figure 1: REST архитектурата (https://miro.medium.com/v2/resize:fit:640/0\*KikwdypTj1FVSpB2.png)

REST архитектурата се характеризира с лесна разширимост и простота, което я прави подходяща за разработването на разнообразни уеб приложения. Основните принципи на REST включват:

* **Безсъстояние**: Всеки HTTP заявка е независима и съдържа цялата необходима информация за обработка. Сървърът не съхранява информация за състоянието на клиента между заявките, което прави системата по-надеждна и мащабируема.
* **Кеширане**: Отговорите могат да бъдат кеширани за подобряване на производителността. Кеширането намалява натоварването на сървъра и ускорява времето за отговор.
* **Еднороден интерфейс**: Ресурсите са достъпни чрез предвидим и стандартизиран интерфейс, обикновено HTTP, който включва използването на стандартни HTTP методи.
* **Система от ресурси**: Всяка функционалност или данни в приложението се разглеждат като ресурс, който може да бъде идентифициран и адресиран.

В контекста на модела Model-Service-Controller, REST архитектурата позволява ясно разделение на отговорностите, което улеснява разработването, тестването и поддръжката на приложенията. Така, приложението става по-гъвкаво и адаптивно към изменения и разширения, което е особено важно в динамичната среда на уеб разработките и конкретно за нашата система. Ще бъде полезно да може да се разраства, за да се добавят нови функционалност и допълнения при нужда, като това винаги е възможно тъй като образователната система се изменя и надгражда непрекъснато. Също така надеждността на системата ще спомогне за по-леко преминаване на заявкии по-рядко натоварване на сървъра въпреки големия брой студенти, които се очаква да го изпозлват.

1. **Java**

Подходящ за нашата система е Java, един от най-популярните програмни езици, широко използван за разработването на разнообразни приложения, от мобилни апликации до големи корпоративни системи. Той е обектно-ориентиран език, което означава, че поддържа концепции като наследяване, инкапсулация и полиморфизъм, улесняващи структурирането на програмите и управлението на сложни системи.

Едно от ключовите предимства на Java е платформената независимост, благодарение на принципа "Write Once, Run Anywhere". Програмите, написани на Java, могат да работят на всяка машина, която има инсталирана Java Virtual Machine (JVM), което я прави идеален избор за разработването на крос-платформени приложения.

Java също така предлага богата стандартна библиотека, която включва широк спектър от инструменти и функции, от уеб разработка до работа с бази данни. Това улеснява и ускорява процеса на разработка, като позволява на разработчиците да се съсредоточат върху бизнес логиката на приложението, вместо да се занимават с ниско ниво програмиране.

Java е известен със своята стабилност и мащабируемост, което я прави подходяща за разработването на големи и сложни системи като студентската система за връзка с преподавателите. Тези характеристики осигуряват надеждност и възможност за разширяване на системата в бъдеще, без да се налага пренаписване на кода.

Освен това, Java поддържа многонишково програмиране, което е от съществено значение за изграждането на отзивчиви и ефективни приложения. Това позволява на системата да обработва множество заявки едновременно, увеличавайки производителността и подобрявайки потребителското изживяване. Силната общност и широкото приложение на Java в индустрията означават, че разработчиците имат достъп до обширни ресурси, включително образователни материали, библиотеки и поддръжка от други програмисти. Това е особено важно за студентите и начинаещите разработчици, които работят над дипломни проекти.

1. **Spring Framework**

Spring Framework е мощна и гъвкава платформа за разработване на Java приложения, широко призната в индустрията заради своята лекота и удобство при използване. Той предлага обширен набор от модули за различни аспекти на програмирането, включително уеб приложения, обработка на данни и интеграция с други системи.

Spring улеснява разработването на приложения чрез инверсия на контрола (IoC) и впръскване на зависимости (DI), което води до по-чист и по-организиран код. Тези концепции позволяват на разработчиците да се фокусират върху бизнес логиката, вместо да се затрудняват със сложни конфигурации. Той поддържа и развива аспектно-ориентирано програмиране (AOP), което позволява отделянето на аспекти на приложението като сигурност и управление на транзакции от основния бизнес код. Това води до по-прост код и улеснява поддръжката.

Framework-ът предлага и поддръжка за обработка на данни чрез JDBC, JPA, Hibernate и други популярни ORM технологии, което прави взаимодействието с бази данни по-лесно и интуитивно. Също така, Spring предоставя мощни средства за тестване, които позволяват лесно писане и изпълнение на тестове за приложенията. Той също така е идеален за създаването на RESTful уеб услуги благодарение на Spring MVC, което улеснява разработването на съвременни уеб приложения с чисти и лесни за поддръжка архитектури.

С тези характеристики, Spring Framework се превръща в отличен избор за разработването на сложни и мащабируеми приложения като студентската система за връзка с преподавателите, предоставяйки стабилност, гъвкавост и мощни инструменти за разработчиците.

1. **HyperSQLDB**

HSQLDB (HyperSQL DataBase) е лека, високопроизводителна SQL база данни, написана изцяло на Java. Тя е известна със своята бързина, надеждност и гъвкавост, което я прави подходяща за широк спектър от приложения, от разработване и тестване до вграждане в продуктови решения.

Едно от основните предимства на HSQLDB е нейната способност да работи в различни режими, като например вграден (embedded) и сървърен режим. Вграденият режим позволява на базата данни да работи като част от приложението, без нуждата от отделен процес на сървъра. Това е особено полезно за разработка и тестови среди, тъй като улеснява разработчиците да създадат и управляват база данни непосредствено в приложението. Това би влязло в употреба за по-лесно започване на работа по приложението в начален стадий. Като по всяко време може да се премине на сървърен.

HSQLDB поддържа голямо разнообразие от стандартни SQL функции и синтаксис, включително сложни заявки, външни ключове, множество типове данни и транзакции, което я прави мощен инструмент за обработка на данни. Това я прави подходяща за приложения, които изискват сложна обработка на данни и в същото време изискват лека и гъвкава база данни. Освен това, HSQLDB е изключително лесна за настройка и използване, което я прави идеална за бързо разработване и прототипиране. Тя може да се стартира и използва с минимална конфигурация, което е отлично за ситуации, където времето за разработка е критично.

В контекста на студентската система за връзка с преподавателите, HSQLDB предлага бърза и лесна за употреба база данни, която може лесно да се интегрира с Java приложения. Нейната способност да работи в различни режими и поддръжката на мощни SQL функции я правят подходящ избор за такъв тип приложение, където се изисква гъвкава и надеждна обработка на данни. Това е особено важно за образователни системи, където управлението и съхранението на образователни и лични данни трябва да бъдат извършвани по надежден и ефективен начин. HSQLDB осигурява баланс между функционалност и лекота, като в същото време предлага бързина и простота, което е от съществено значение за поддържането на гладка работа на такива системи.

1. **Postman**

Postman е популярен инструмент за разработка и тестване на API (Application Programming Interface). Той предоставя удобен графичен интерфейс, който позволява на разработчиците лесно да създават, изпращат и анализират заявки към уеб API-та. Postman е широко използван заради своята гъвкавост и мощни функции, които правят тестването и разработката на API по-проста и ефективна.

С помощта на Postman, могат да се симулират различни типове HTTP заявки, като GET, POST, PUT, DELETE, и други. Възможно е също настройването на хедъри, параметри на заявките и телата на съобщенията, което позволява детайлно тестване на API функционалностите. Postman поддържа и различни автентикационни механизми, като OAuth и Basic Auth, което го прави подходящ за тестване на API-та с различни нива на сигурност.

Освен това, Postman предлага възможност за съхраняване и споделяне на колекции от заявки, което улеснява работата по няколко проекта едновременно. Това позволява също организирни тестове и да споделяне на тестови сценарии, което спомага за повишаването на ефективността и последователността при тестването на API.

Postman може да бъде изключително полезен за тестване и усъвършенстване на API, който управлява комуникацията между различните компоненти на системата. Неговите мощни функции за тестване, автоматизация и сътрудничество правят Postman идеален инструмент за гарантиране на надеждността и ефективността на API в рамките на такава система.

1. **cURL**

Заявките могат да бъдат описани чрез curl като команден инструмент. Той позволява изпращането на заявки към сървър. Методът на заявката може да бъде разнообразен, като например POST, GET, PUT, DELETE, в зависимост от нуждите на заявката. URL адресът указва към сървъра, към който се изпраща заявката и може да бъде както локален, така и външен. Заглавията в заявката могат да включват Content-Type, който определя формата на данните, например application/json, text/plain, application/xml, и други, както и Cookie или други хедъри за управление на сесии или удостоверяване. Тялото на заявката съдържа данните, които се изпращат към сървъра; при POST заявки често е във формат JSON, докато при GET заявките тялото обикновено е празно и данните се предават чрез URL параметри. Тази структура е гъвкава и позволява адаптация за различни видове заявки и потребности.

1. **Maven**

Maven е инструмент за управление и автоматизация на компилирането на кода и пакетирането му в изпълними формати като JAR файлове, което улеснява управлението на зависимости (dependencies) и конфигурациите на проекта чрез своя POM (Project Object Model) файл. Maven автоматично изтегля необходимите библиотеки и плъгини в съответствие с дефинираните зависимости в POM файла, като осигурява консистентност и преизползваемост на кода. В сравнение с Gradle, който предлага по-голяма гъвкавост и поддръжка на Groovy и Kotlin за скриптове, Maven се характеризира със своята строга стандартизация и по-проста структура. Изборът между Maven и Gradle често зависи от нуждите на проекта – Maven е подходящ за проекти, които се нуждаят от стабилна и лесна за управление среда, докато Gradle е предпочитан за по-гъвкави и модерни сценарии на сборка.

1. **TypeScript**

TypeScript е отворен програмен език, разработен от Microsoft, който е суперсет на JavaScript. Той добавя статично типизиране и обектно-ориентирани характеристики към JavaScript, което помага за по-лесното управление на големи кодови бази и подобрява четливостта и поддръжката на кода. TypeScript се компилира до JavaScript, което означава, че всеки валиден JavaScript код е също валиден TypeScript код. Предимствата на TypeScript включват по-добро разпознаване на грешки при компилация, подобрена интеграция с развойни среди и улеснено управление на сложни проекти.

1. **Angular Framework**

Базирайки се на TypeScript, Angular е мощна платформа и фреймуърк за разработване на клиентски уеб приложения. Разработен от екипа на Google, Angular използва TypeScript като основен език и предлага усъвършенствани характеристики като двупосочно свързване на данни (two-way data binding), модуларна архитектура и инжектиране на зависимости, което улеснява създаването на големи и сложни уеб приложения. Angular поддържа и компонентно-базиран подход, което позволява да се създават повторно използваеми, лесни за поддържане и тестване компоненти.

Той се отличава със своята ефективност и гъвкавост, което го прави популярен избор за създаването на SPA (Single Page Applications) и други сложни клиентски приложения. Той също така предлага богат набор от инструменти и библиотеки, като Angular CLI (Command Line Interface), което улеснява автоматизацията на различни задачи в разработката, като създаване на проекта, добавяне на компоненти и управление на зависимости.

В контекста на студентската система за връзка с преподавателите, използването на TypeScript и Angular може да предложи редица предимства. TypeScript улеснява разработката и поддръжката на кода, особено при големи проекти, като в същото време предоставя по-добро управление на грешките и подобрена интеграция с инструменти за разработка. Angular, от своя страна, предлага структуриран и модуларен подход към разработката на уеб приложения, което улеснява създаването на интерактивни, отзивчиви и лесни за употреба интерфейси, което е от съществено значение за образователните платформи.

Съчетавайки тези технологии, системата би могла да предостави стабилна и функционална платформа, която да се интегрира в академичния живот на Техническия университет, облекчавайки ежедневната комуникация и координация между студенти и преподаватели.

1. Програмна реализация

A screenshot of a computer

Description automatically generatedПриложението бива реализирано, основавайки се на Java. Файловете са организирани в следната структура: [Фигура 2]

Figure 2: Файловa структура на "TU-Connect"

За по-детайлно разглеждане на програмната реализация ще се използва следния ред: разглеждане на папката config и конфигурациите използвани за имплементирането на функционалностите, решаващи проблемите пред системата, след това ще се навлезе в controllers, където ще се разгледат по-важните ендпойнти, моделите и сървисите, които те използват и ще се разгърне останалата логика чрез енъмите, които биват използвани на различни места в проекта.

1. **Изпращане на имейли**

Както името подсказва, основната функция на системата "TU-Connect" е да предостави връзка на студентите с преподавателите, по-лесна от тази, която те имат в момента. Затова ще започнем разглеждането с конфигурациите и по-точно с тази, с която ще се извършва изпращането на имейли.



Figure 3: Конфигурации

Анотацията @Configuration в класа **EmailConfig** означава, че той се използва от Spring Framework за дефиниране на конфигурационни параметри, които управляват създаването и управлението на обекти (beans) в контекста на приложението. Тази анотация казва на Spring, че класът съдържа бийн дефиниции, които трябва да бъдат регистрирани в контейнера на Spring.

A computer screen shot of a program

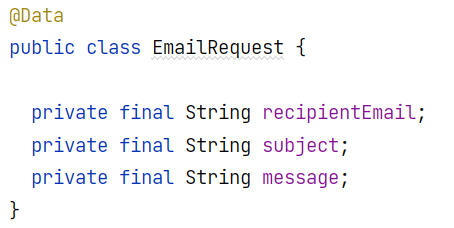
Description automatically generated

Figure 4: EmailConfig

Методът javaMailSender() е означен с анотацията @Bean, което означава, че той връща обект, който трябва да бъде управляван от Spring контейнера като бийн. В Spring бийнът е обект, който е инстанциран, управляван и конфигуриран от Spring. Използването на @Bean позволява на Spring да знае, че методът предоставя инстанция, която трябва да бъде включена в контекста на приложението и да бъде достъпна за инжектиране в други части на приложението.

JavaMailSenderImpl е клас, който имплементира интерфейса JavaMailSender, предоставящ методи за изпращане на имейл съобщения. Чрез него конфигурираме сървъра за изходящата поща и порта, задаваме потребителско име и парола за автентикацията на акаунта за изпращане на имейли. Допълнително активираме STARTTLS за сигурна връзка. Тази конфигурация ни позволява да изпращаме имейли сигурно и бързо, тя бива използвана в различни части на приложението.

Класът **EmailService**, маркиран с @Service, предоставя функционалност за изпращане на имейли в рамките на системата. Той използва JavaMailSender, който, както е погазано по-горе, е конфигуриран чрез бийн метод в контекста на Spring, за да осигури възможността за изпращане на имейли. Конструкторът на EmailService получава и инициализира JavaMailSender чрез @Autowired, което позволява инжектирането на зависимости от Spring контейнера.



Методът sendEmail приема обект **EmailRequest**, който съдържа информация за получателя, темата и съобщението на имейла. Той използва JavaMailSender за създаване на MimeMessage, който е форматът на имейл съобщенията в Java. MimeMessageHelper се използва за настройка на различни аспекти на имейла като получателя, темата и текста на имейла. Вътрешно, sendEmail изпраща създаденото имейл съобщение, като хваща и обработва всякакви изключения, които могат да възникнат по време на процеса на изпращане.

A computer screen shot of a program

Description automatically generated

**EmailController**, маркиран като @RestController, управлява HTTP заявките, свързани с изпращането на имейли. Той в момента не се използва никъде от външна страна, а само в приложението вътрешно. Затова и се разглежда заедно с кофигурацията за имейлите, а не с останалите контролери. Въпреки това е възможно да се наложи използването на подобен ендпойнт. Например е вероятно да се появи функционалност с изпращане на имейли директно от фронтенда с изпращане на готов оформен имейл към бекенда. Това може да стане с подаване на EmailRequest обект.

/send-email

**POST /**

Методът sendEmail, оформен като POST ендпойнт (@PostMapping("/send-email")), обработва входящите заявки за изпращане на имейли. Той приема обект EmailRequest като тяло на заявката (@RequestBody) и използва EmailService за изпращане на съответния имейл.



За по-лесно изпращане на имейли в различни части на приложението и за повече пригледност и подреденост имаме енъм, който представлява повечето сценарии, по които може да се праща имейл, под формата на заглавие (subject) на имейла. Той изглежда така:

A screenshot of a computer

Description automatically generated

Покрива случаите, в които се изпраща имейл при получаване на заявка от преподавател за ръководене на дипломна работа, аз приета такава заявка от студент, за заявена уговорена среща от студент към преподавател, за приемане на тази среща от преподавателя, уведомявайки студента, и за отхръляне на срещата.

1. **Автентикация и jwt токени**

Класът **JwtUtils**, маркиран като @Component в Spring, се използва за управление на операциите, свързани с JWT (JSON Web Tokens). Този клас осигурява функционалностите за генериране, анализиране и валидиране на токени, което е от съществено значение за сигурността на приложението.

A screenshot of a computer program

Description automatically generated

Figure 5: JwtUtils

Методът generateToken се използва за създаване на нов JWT, който съдържа потребителското име като субект. Той извиква вътрешния метод createToken, който използва библиотеката Jwts за конструиране на токена. Този метод задава различни атрибути на токена, като претенции (claims), субект (потребителско име), време на издаване, срок на годност и алгоритъм за подписване.

Секретният ключ, използван за подписване на токените, се дефинира статично в класа. Този ключ осигурява, че токените са защитени и не могат да бъдат фалшифицирани. Времето за изтичане на токена също е дефинирано, като по подразбиране е зададено на 5 часа, за да не се използва refreshToken и за да може да не се прекъсва дейността на студентите и преподавателите в сайта.Важен аспект на JwtUtils е валидацията на токените. Методът validateToken проверява дали токенът все още е валиден и дали съответства на даден потребител. Това се прави чрез проверка на потребителското име и проверка за изтекъл срок на токена.

JwtUtils е критичен компонент за управление на сигурността в приложения, които използват JWT за аутентикация и авторизация. Той предоставя средствата за сигурно създаване, анализиране и валидиране на токените, които са от съществено значение за поддържане на сигурността на информацията и данните на потребителите.

**JwtAuthFilter** е компонент на Spring, който се използва като филтър за проверка на JWT (JSON Web Token) при всеки HTTP заявка. Класът наследява OncePerRequestFilter, което означава, че филтърът ще бъде извикван веднъж за всяка заявка към сървъра. В конструктора на класа, JwtUtils и UserService са инжектирани чрез @Autowired.

A screenshot of a computer program

Description automatically generated

Figure 6: JwtAuthFilter

В метода doFilterInternal, филтърът първо проверява дали има JWT в хедъра Authorization на входящата HTTP заявка. Ако токенът е намерен и е валиден, филтърът извлича потребителските данни и създава UsernamePasswordAuthenticationToken, който се използва за установяване на потребителската идентификация в контекста на Spring Security. Това позволява приложението да знае, че текущият потребител е легитимно удостоверен. След което, filterChain.doFilter(request, response) препраща заявката към следващия филтър в веригата.Този механизъм осигурява, че всяка заявка преминава през необходимите проверки за сигурност, преди да достигне до останалата част от приложението. JwtAuthFilter гарантира, че само заявки с валиден JWT имат достъп до защитени ресурси и функционалности.

Следващата конфигурация, която имаме е **PasswordEncoderBean**, който ни връща BcryptPasswordEncoder. Това е механизъм за хеширане на пароли, предоставен от Spring Security, който използва BCrypt алгоритъма. Този алгоритъм е силно препоръчителен за сигурността на паролите, тъй като е устойчив на атаки със силова подбор (brute-force attacks) благодарение на своята способност за настройка на сложността. BCryptPasswordEncoder автоматично включва солене (salt) в хеширания процес, което добавя допълнителен слой сигурност, предпазвайки от различни атаки по сигурността. Използването му в приложения гарантира, че съхранените пароли са защитени по най-добрия възможен начин. В контекста на уеб приложения, BCryptPasswordEncoder е важен инструмент за защита на потребителските пароли и предотвратяване на неоторизиран достъп.

Класът **SecurityConfig** определя конфигурацията за сигурността на уеб приложението. Той включва анотациите @EnableWebSecurity и @EnableMethodSecurity, които активират защитата на уеб ниво и добавят допълнителни опции за сигурност като методи на базата на роли и анотации. В този клас се инжектират UserService и PasswordEncoder чрез конструктора. Методът filterChain конфигурира HttpSecurity, като изключва защитата от междусайтови фалшиви заявки (CSRF) и определя правилата за авторизация на заявките. Той налага изискването за аутентикация на определени API пътища - 'api/booking/\*\*', 'api/thesis/\*\*', и 'api/tutors/\*\*', което означава, че достъпът до тези пътища е разрешен само за потребители, които са влезли в системата. За всички останали заявки конфигурацията позволява свободен достъп без изискване за аутентикация. Тази настройка гарантира, че чувствителните API ендпойнти са защитени, докато останалата част от приложението остава достъпна за неаутентикирани потребители.

Методът corsConfigurer създава конфигурация за Cross-Origin Resource Sharing (CORS), която позволява на приложенията да взаимодействат между различни домейни, в този случай разрешавайки заявки от "<http://localhost:4200>", което е порта на нашия фронтенд проект. authenticationProvider създава и конфигурира DaoAuthenticationProvider, който използва userService за зареждане на потребителските данни и passwordEncoder за проверка на паролите. Това осигурява механизъм за аутентикация, базиран на потребителското име и парола, а authenticationManager методът предоставя AuthenticationManager, който е централен компонент в Spring Security за обработка на заявките за аутентикация.

SecurityConfig е отговорен за конфигурирането на различни аспекти на сигурността в приложението, включително аутентикация, авторизация, CORS политики и управление на пароли.

Сега преминаваме към разглеждането на контролерите. Тематично можем да започнем с UserController. Тук ще разгледаме и AuthenticationService, User модела, както и съответстващия му UserService.

A screenshot of a computer

Description automatically generated

Figure 7: Контролерите

**User** класът е дефиниран като абстрактен и служи като базов клас за потребителски модели. Той има няколко анотации. @MappedSuperclass указва, че този клас предоставя мапинг информация за неговите подкласове в базата данни. @NoArgsConstructor автоматично генерира конструктор без аргументи, а @AllArgsConstructor генерира конструктор, който приема един аргумент за всяко поле на класа. @Getter и @Setter автоматично генерират гетъри и сетъри за всички полета на класа, което елиминира необходимостта от ръчно написване на тези методи. @Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS) определя стратегията на наследяване, където всяка подклас таблица съдържа колони за всички полета в класа. @SuperBuilder: Позволява изграждането на обекти чрез шаблон за дизайн на строителя за подкласове.

Класът съдържа абстрактен метод getRole(), който бива имплементиран в наследяващите го класове, а getAuthorities(): Връща списък от права, които се отнасят за ролята на потребителя.

UserRole е изброен тип (enum/енъм), който дефинира различни роли на потребители в системата. Той се връща, когато се извика метода getRole(). Неговите стойности могат да бъдат „STUDENTE“ за студентите и „TUTORE“ за преподавателите.

След което можем да разгледаме UserService и методите, които съдържа той. Не всички от тях се използват в контролера в момента, но биха подпомогнали бъдещето развитие на системата, като предпоставят по-лесното разширяване с допълнителни функционалности. Ще разгледаме по-интересните от тях.

list() е метод, който връща списък от 2те категории потребители засега, студенти и преподаватели.

A screen shot of a computer code

Description automatically generated

getByUsernameAndPassword() търси потребител по потребителско име и проверява паролата с помощта на BCrypt.

A screen shot of a computer

Description automatically generated

usernameIsTaken() проверява дали дадено потребителско име вече съществува в системата.

loadUserByUsername() използва се от Spring Security за зареждане на потребителски данни по потребителско име.

addUser() добавя нов потребител в системата, като първо проверява за съществуващо потребителско име и след това записва потребителя в съответната категория (преподаватели или студенти) с кодирана парола, като при липсваща такава подадена, по подразбиране се задава студентска роля.

A computer screen shot of a program

Description automatically generated

UserController работи с **AuthenticationService** и UserService. AuthenticationService имплементира UserDetailsService. Тук важният метод е за автентикиране, който връща токен, с който потребителят може да използва системата.

A screen shot of a computer code

Description automatically generated

А от UserService се използва addUser метода и в резултат имаме следните 3 ендпойнта:



/users

Заявкита под /users отговарят за операции свързани с потребителите. Тук има доста възможности за развитие и функционалности, но за момента ще разгледаме само 3те имплементирани заявки, за регистриране, за автентикиране пред приложението и за извеждане на списък със всички потребители.

**POST /users/login**

**За успешна автентикация е нужно подаването на няколко аргумента в тялото на заявка.**

**Задължителни параметри:**

**“username” – това е потребителкското име, с което потребителят е бил регистриран**

**„password“ – това е паролата, с която потребителят е бил регистриран**

**Незадължителни параметри:**

**“role” - това е ролята на профила на потребителя.**

****

**Response:**

**Отговорът, който се очаква от системата при правилни детайли пратени със заявката и правилно поведение на системата е валиден jwt токен, който да може да се ползва от потребителя като се изпрати в хедър заедно с всяка заявка, изискваща автентикиран потребител.******

**Ако се опитаме да се автенитикираме с грешни детайли, например грешно изписана парола, няма да успеем да получим 200 ОК или валиден токен. Правилното поведение на системата ще изисква връщане на отговор със статус 403 и код Forbidden, със Stacktrace и причина за неуспешно минала заявка. Примерен отговор може да се по-долу, както и примерна заявка с грешна парола.**

**Request:******

**Response:******

**POST /users/register**

**Това е заявката, с която потребителят може да се регистрира в системата и да получи достъп до нея и повечето функционалности. Изискват се задължителни параметри за изпращане заедно със заявката.**

**Задължителни параметри:**

**“username” – това е потребителкското име, с което потребителят ще бъде регистриран**

**„password“ – това е паролата, с която потребителят ще бъде регистриран**

**Незадължителни параметри:**

**“role” - това е ролята на профила на потребителя**

****

**Note: Ролята се представлява от енъм, който е възможно да бъде със стойност “STUDENTE” или “TUTORE”, като при липсващ параметър се задава по подразбиране ролята на студент. За да бъде успешна заявката е задължително параметърът да не бъде изписан по какъвто и да е друг начин, а специфично с главни букви.**

**Response:  
При правилна заявка и очаквано поведение на системата, отговорът, който трябва да се върне ще бъде със статус 200 ОК без тяло.**

****

**GET /users**

**Заявка за взимане на колекция от потребители. Тази заявка може да се изпълни само при автентикиран потребител, затова се изисква хедър с jwt токена, който ще каже на бекенда, че съответната заявка е от потребител на системата и ще върне потребителите като резултат. Хедърът трябва да се казва “Authorization” и токенът трябва да бъда енкоднат??**

****

1. **Преподаватели**

Класът **Tutor** представлява модел на обект в базата данни, който съхранява информация за преподавателите. Използва анотации като @Entity и @Table за да определи свързването с таблица в базата данни, и @Getter, @Setter, @NoArgsConstructor, @SuperBuilder за автоматично генериране на гетъри, сетъри, конструктор без аргументи и билдър патърн. @Id и @GeneratedValue(strategy = GenerationType.AUTO) дефинират полето id като уникален идентификатор (първичен ключ) на Tutor ентитета. @GeneratedValue указва, че стойността на id ще бъде автоматично генерирана от базата данни, което гарантира уникалност.

Анотациите свързват полетата със съответните колони в таблицата TUTOR. Атрибутът length указва максималната дължина на стойностите, които могат да бъдат съхранени в тези колони. Комбинацията от анотации @ManyToOne и @JoinColumn(name = "FACULTY\_ID") дефинира връзка от много към едно между Tutor и Faculty. @JoinColumn указва, че колоната FACULTY\_ID в таблицата TUTOR е външен ключ, който свързва с таблицата Faculty.

@Column(name = "FACULTY") свързва полето facultyId с колоната FACULTY в таблицата TUTOR. @OneToMany(mappedBy = "tutor", cascade = CascadeType.ALL) определя връзка от едно към много между Tutor и Thesis, а mappedBy = "tutor" указва, че връзката се управлява от полето tutor в класа Thesis. Подсигуряваме се с cascade = CascadeType.ALL, че всички операции (като добавяне, изтриване и т.н.) върху Tutor автоматично се прилагат и към свързаните Thesis обекти. Допълнително всеки Tutor може да бъде свързан с един набор от часове за посещения и един график.



Методите isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired и isEnabled са за изпълнение на интерфейса UserDetails и в този случай връщат false, което може да означава, че тези аспекти на акаунта не се управляват активно. Тук методът getRole() връща UserRole.TUTORE. По-интересен метод е setVisitingHours(). Той прави проверка дали часовете са въведени правилно, и ако не са, няма да и сетне подобаващо.

A screen shot of a computer program

Description automatically generated

**TutorDTO** е клас за пренос на данни (Data Transfer Object), който представлява по-опростена версия на Tutor за използване в комуникацията между клиента и сървъра. Той използва @Builder за улесняване на създаването на обекти и включва метод convert, който трансформира Tutor обект в TutorDTO обект, в него могат да се видят детайлите, които се пазят в DTO обекта.

A computer screen shot of a program

Description automatically generated

**TutorController** е контролер, който управлява HTTP заявките свързани с преподавателите. Той е маркиран с @RestController и @RequestMapping, което определя URL пътя за API ендпойнтите. В този контролер се съдържат 2 метода: getAllTutors, който връща списък на всички преподаватели, и getTutorById, който връща информация за конкретен преподавател по неговото ID. В тези методи се извършва преобразуването на Tutor обекти в TutorDTO обекти за отговор на клиента. Примерни заявки биха изглеждали по следния начин:

**/api/tutors**

**GET** /



**GET** /tutor/{id}



1. **Студенти и дипломни работи**

Класът **Student** се различава малко от другите досега по анотации. Нещата, с които разширява класа User не са много. От полетата имаме groupNum, която съхранява групата, в която се намира студентът. Имаме връзка от едно към много между Student и Thesis, mappedBy = "student" указва, че връзката се управлява от полето student в класа Thesis. Имаме и дефинирано полето fkNum като уникален идентификатор (първичен ключ) на Student ентитито, вместо id. getRole() в този случай връща UserRole.STUDENTE.

A computer screen shot of a math equation

Description automatically generated

**Thesis** е ентити, представляващо дипломна работа в системата. Чрез @ManyToOne се свързват с преподавателите и студентите. Останлите полета са email, content и статус, които ще влязат в употреба в сървисите. Статусът на дипломните работи е публичен енъм с 3 стойности в зависимост от етапа на развитие на дипломната.

A screenshot of a computer

Description automatically generated

A white background with purple text

Description automatically generated

Статусът в класа Thesis е ограничен откъм редактиране извън класа и се сменя през публични методи.

A computer code with text

Description automatically generated

Класът съдържа списък thesisOffers, които не могат да бъдат редактирани никъде веднъж щом са инициализирани. Възможно е само единично добавяне на инстанции.



**ThesisOffer** представлява предложение за поемане на дипломна работа като дипломен ръководител. Съдържа полета за имейл на преподавателя, който заявява желанието и незадължителна бележка, note, за евентуални препоръки или идеи за доразвиване на дипломната работа.



@JsonIgnore предотвратява сериализацията на цялата дипломна работа при визуализацията на предложението.

И трите се използват активно в **ThesisService**. Той менажира както дипломните работи (Thesis), така и предложенията от преподавателите към тях (ThesisOffer):

A close-up of a computer code

Description automatically generated

Имаме методи за запазване на инстанции в базата, както и за търсене на такива по id. Интересен е методът за добавяне на ThesisOffer към работата:



Тук получаваме като аргумент офертата, от нея взимаме самата дипломна работа, за която е тя. Тук вече използваме гореспоменатите методи за сменяне на статуса. Той става PENDING в момента, в който постъпи заявка за ръководител и тя съдържа дипломна в себе си, имаме null check. След това тя се добавя към дипломната работа и запазваме промените в базата. Тук се използва вече познатия ни EmailService за изпращане на имейл до студента, уведомявайки го за настъпилата заявка. След което връщаме ResposneEntity, със съответния статус и съобщение в тялото.

Използването на ResponseEntity в такива сценарии дава възможност за ясно и точно комуникиране на статуса на заявката, както и предоставяне на подходяща обратна връзка към клиента, което е важно за разработването на интуитивни и надеждни RESTful API-та. ResponseEntity в Java, особено в контекста на Spring Framework, се използва за представяне на цялия HTTP отговор, включващ не само тялото на съобщението, но и статус кодовете и хедърите. Този обект позволява по-фин контрол върху отговора, който се изпраща обранто към клиента.

Контролерът **ThesisController** и заявките в него се занимават с менажирането на функционалностите около дипломните задания. А те са споделяне на идеи за дипломно задание, поемане на дипломно задание от преподавател и анонимност преди приемане на преподавател за ръководител.

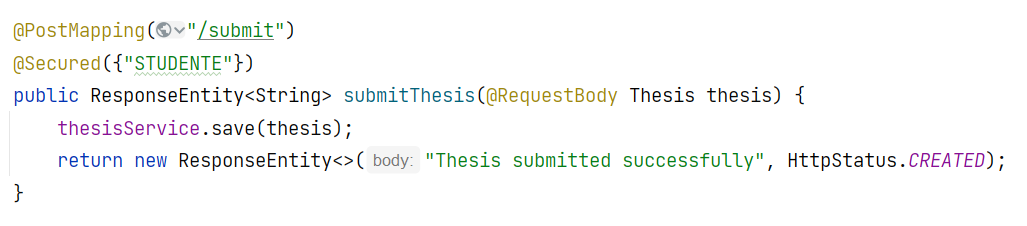
Приложението пази записани идеи за дипломни работи, подадени от студенти. Всеки може да добави своя идея като трябва да опише детайлно проблемите, които решава дипломната, технологиите и всичко, което ще даде представа на преподавателите с какво желае да се занимава дипломантът и дали биха могли да му помогнат. При интерес към някоя, преподавателите могат да заявят желанието си да я поемат, при което се изпраща мейл до студента с информация за преподавателя и евентуални препоръки или идеи за бъдещо развитие на идеята. Студентите могат да извадят списък с всички преподаватели, които са изявили желание да поемат идея за дипломна работа. Като всеки студент може да изпрати повече от една идея. Избирайки един от всички, останалите предложения се изтриват, както и самата идея и имейл бива изпратен и до двамата заинтересовани. Системата съдържа и функционалност за запазване на личността на студента, докато не бъде приета дипломен ръководител. Личният имейл на студента се изпраща на преподавателя чак тогава и така той вече знае с кого ще работи по дипломното задание. А студентът получава мейл с потвърждение, че дипломната работа вече има ръоводител и той е уведомен. Оттук комуникацията преминава извън приложението. Целта е да се премахне евентуално притеснение от страна на студените и да могат свободно да изпращат идеи за проекти, които ги вълнуват, както и да се премахне фактора предпочитания и лично отношение от страна на преподвателите. Така цялата комуникация е фокусирана върху самия проект и остава неафектирана.

/api/thesis

**POST** /submit

С тази заявка се изпраща от студента идеята му за дипломна работа. Тук задължително присъства jwt токена в хедър “Authorization”, който ще допусне заявката да мине. Имаме нужда и от подаден студента, съдържанието на текста на идеята и имейла на студента, за да получи той информация за развитието на идеята.





**POST** /offer

Тази заявка изпраща оферта от някой преподавател да бъде ръководител на съответната дипломна работа. Тук са нужни имейл адрес на преподавателя, незадължителна бележка, и данни за дипломната идея.



A close-up of a computer code

Description automatically generated

**POST** /accept/offer/{id}

Тази заявка се изполва за приемане на преподавател за дипломен ръководител от студент. Добавя се съобщение, което е написано от студента. С него той започва личната кроеспонденция с ръководителя. Възможност за разширяване на логиката чрез съобщение по подразбиране.



A screenshot of a computer program

Description automatically generated

**POST** /decline/offer/{id}

Напълно очаквано имаме и метод за изтриване на оферта за ситуациите, когато студентът знае, че не иска този преподавател от самото начало за ръководител. Или не може да избере веднага един и премахва постепенно докато се спре на последния. За това е нужно просто jwt токен и id на офертата като path параметър и офертата бива изтрита от базата.



A computer screen shot of a program

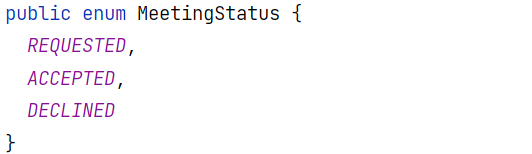
Description automatically generated

1. **Резервиране на среща с преподавател и графици**

Класът **BookedMeeting** в системата е ключов компонент, който представлява уговорени срещи между студенти и преподаватели. Всяка среща има уникален идентификатор, генериран автоматично, както е указано от анотациите @Id и @GeneratedValue. Взаимовръзките между срещите и преподавателите, както и студентите, са определени чрез @ManyToOne и @JoinColumn, което позволява всяка среща да бъде свързана с конкретен преподавател и студент.



Класът съдържа детайли като времетраене на срещата, стаята, в която ще се проведе, причината за срещата и статуса ѝ. Статусът на срещата е особено важен и се управлява чрез изброения тип MeetingStatus, който включва статусите REQUESTED, ACCEPTED и DECLINED. Тези статуси отразяват текущата фаза на срещата в процеса на управление.



**BookingService** играе централна роля в логиката за управление на срещите. Този сервиз използва BookingRepository за достъп и манипулиране на данни за срещи. Методите в сервиза позволяват извличане на всички срещи за определен студент или преподавател, регистриране на нови срещи и промяна на статуса на вече съществуващи срещи. При промяна на статуса на среща, методите в сервиза отговарят за обновяване на информацията в базата данни, като например промяна на статуса на срещата от REQUESTED на ACCEPTED или DECLINED.

A screen shot of a computer program

Description automatically generated

**BookingController** е интерфейсът между BookingService и потребителите на системата. Той предоставя API ендпойнти за извличане на информация за срещи, регистриране на нови срещи и промяна на техния статус. Контролерът използва @Secured анотации за ограничаване на достъпа до някои функционалности само за определени потребителски роли, STUDENTE или TUTORE. Това осигурява, че операциите като регистриране на нова среща или приемането на среща са достъпни само за авторизирани потребители. Контролерът съдържа методи за връщане на всички срещи за даден студент или преподавател, както и за приемане или отхвърляне на заявки за срещи.

/api/booking

**GET** /student/{id}

Тази заявка извежда всички уговорени срещи от съответния студент. Би било добре да се имплементира и метод или job, който почиства срещите, които вече са минали.



**GET** /tutor/{id}

По същия начин като горната заявка, тази ни връща всички срещи, които преподавателят е приел.

**POST** /book

Следващата заявка е едно от най-важните в този контролер. Тя е заявката, която се използва от студента, за да поиска среща с преподавателя. Тази заявка трябва да се обработи правилно, за да се отрази заявката за среща при преподавателя и той да получи мейл. За последното е важно всички данни да бъдат пратени правилно. В примерната заявка са изпратени преподавателя и студента само с id, разбира се има още информация за попълване.



**POST** /accept/meeting/{id}

Тук преподавателят приема предложената среща със студент.



**POST** /acceptWithChanges/meeting/{id}

Тази заявка има почти същата функция като горната с разликата, че изпращаме цял обект BookedMeeting, защото е възмжна промяна по данните, касаещи срещата. Заявката се използва, ако преподавателят желае да направи срещата, но иска да я измести с час например или да я отложи за друг ден. Възможна е и промяна на кабинета, в който ще се проведе срещата. Именно за такива случаи е добавен този ендпойнт.

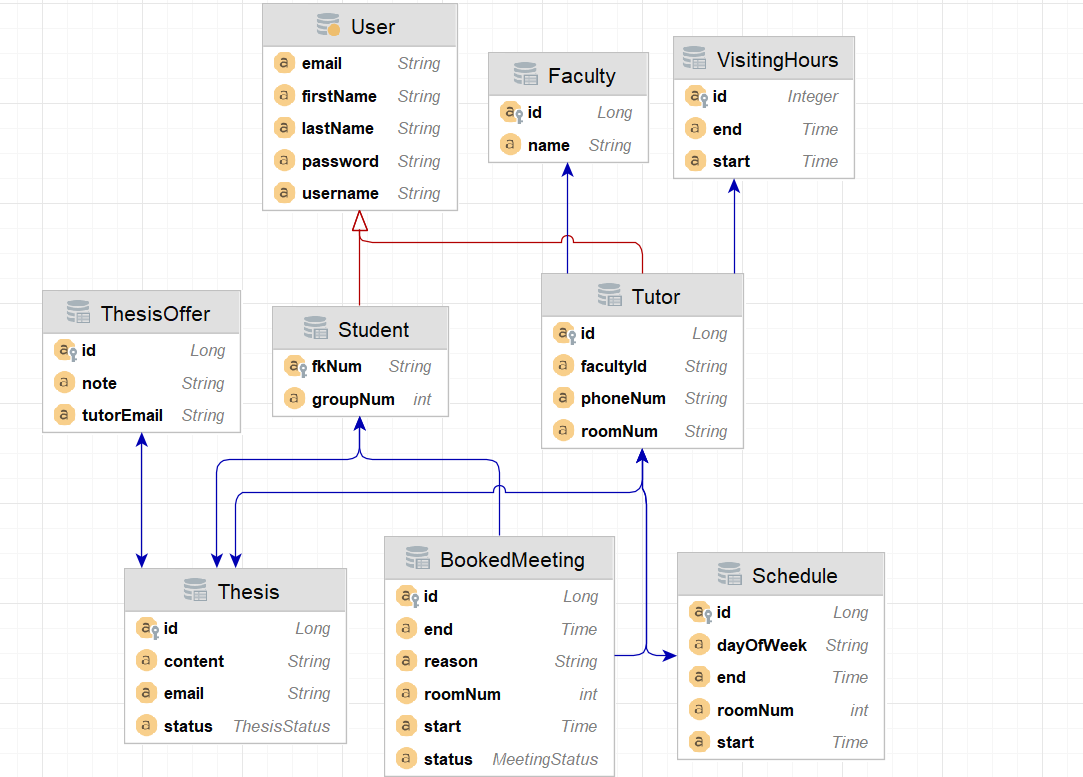


**POST** /decline/meeting/{id}

Последната заявка е за отхвърляне на среща. Може би е нещо, което двамата са успели да изговорят по случайност в университета, или е изникнал ангажимент в последния момент и е по-добре да бъде отложена срещата засега без насрочено ново време. Тази заявка се позлва при такива обстоятелства. При нея не се добавя нищо в тялото, единствено се изисква, както при другите, токен и id на срещата като path параметър.

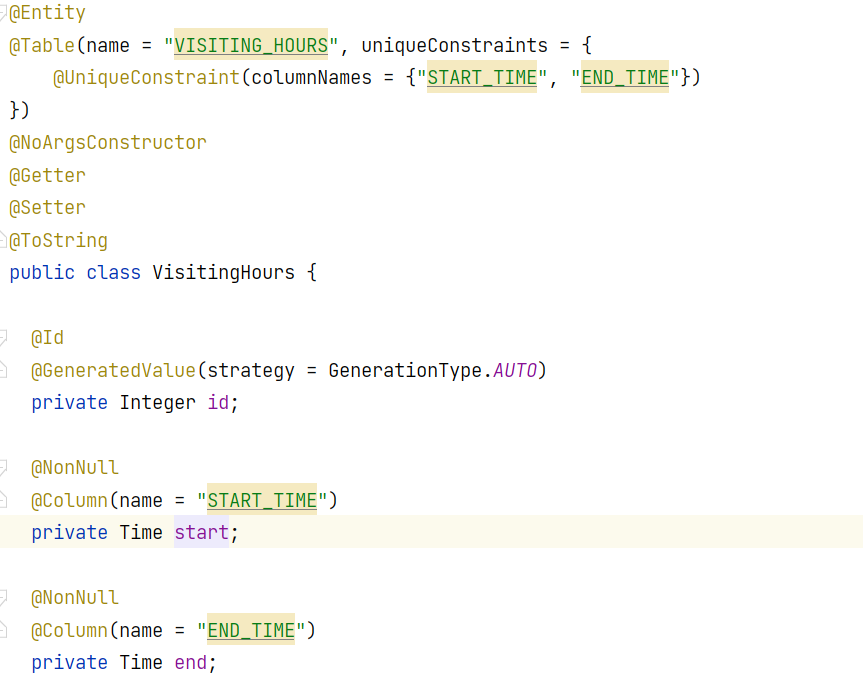
1. **Таблици и отношения между ентитита**

Схемата по-долу предоставя визуално представяне на структурата на базата данни, която поддържа "TU-Connect". Тя показва как различните ентити се взаимодействат помежду си и формират основата на информационната система, която служи на потребностите на студентите и преподавателите в университета. Връзките са представени със стрелки, показващи отношенията между таблиците - например, еднопосочните стрелки показват, че една запис в родителската таблица може да бъде свързана с множество записи в дъщерната таблица.



Класът User служи като базово ентити от което наследяват Student и Tutor, като по този начин студентите и преподавателите се третират като специализирани видове потребители, обогатени със специфични за тях атрибути. В контекста на архитектурата на системата, Tutor е обвързан с няколко ключови ентитита. Има асоциация с Faculty, като всяка асоциация определя принадлежността на преподавателя към определен факултет. Това се реализира чрез връзката @ManyToOne.

Свързан е с **VisitingHours**, което позволява на преподавателите да имат дефинирани часове за посещения, през които са достъпни за консултации със студенти. Тази връзка също е тип @ManyToOne. Има връзка @OneToMany с BookedMeeting, което означава, че преподавателите могат да участват в резервирани срещи със студенти и могат да имат повече от 1 такава. Свързан е и с ThesisOffer чрез връзка @OneToMany, показваща възможността на преподавателите да предлагат себе си за ръководители на дипломни работи.



Student е свързан със Thesis посредством @OneToMany връзка, която позволява на студентите да предлагат, да работят върху дипломни теми и да следят техния статус и развитие.

Schedule е свързан с Tutor, предоставяйки информация за графика на преподавателите, включително определени часове за преподаване и местоположение на занятията. Тази връзка също е реализирана чрез @OneToMany.

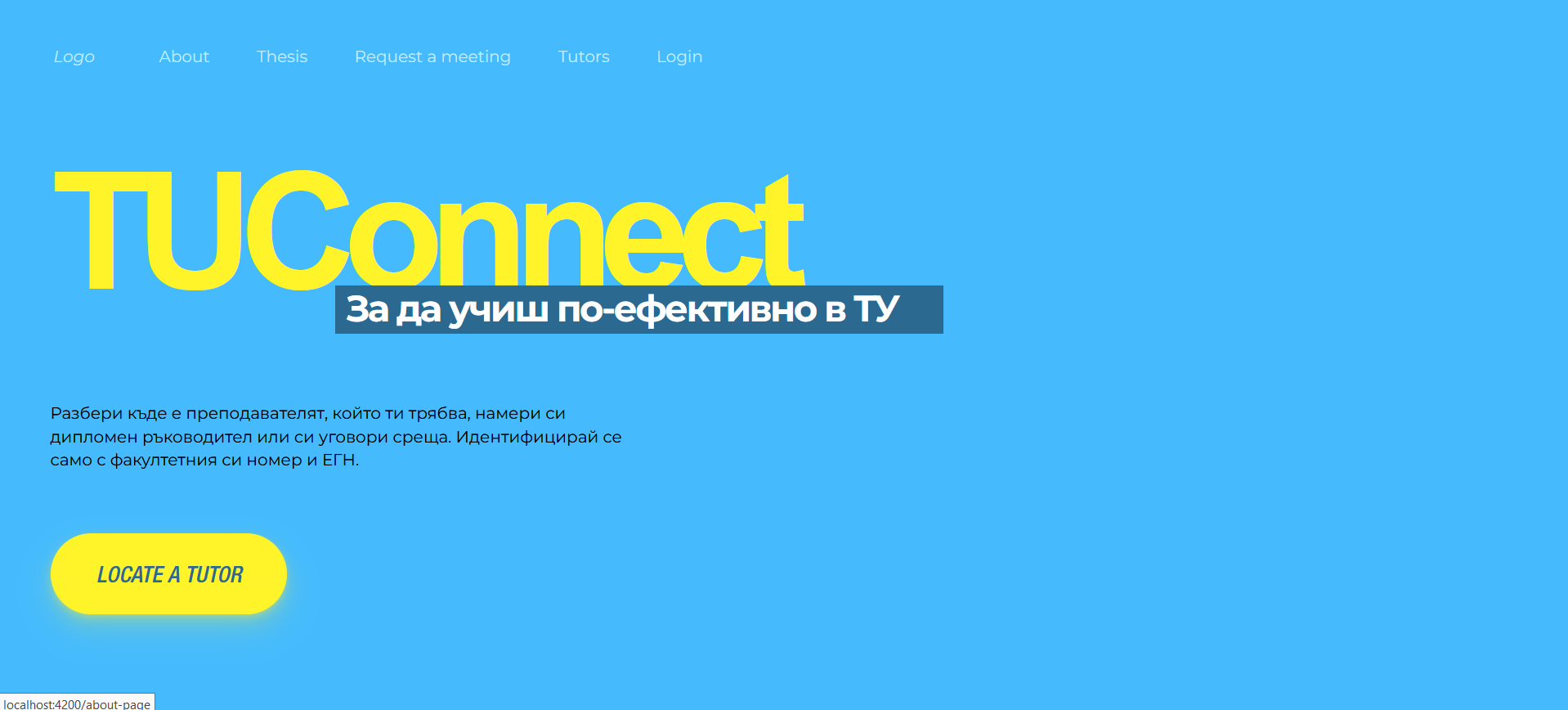
A computer screen shot of a computer code

Description automatically generated

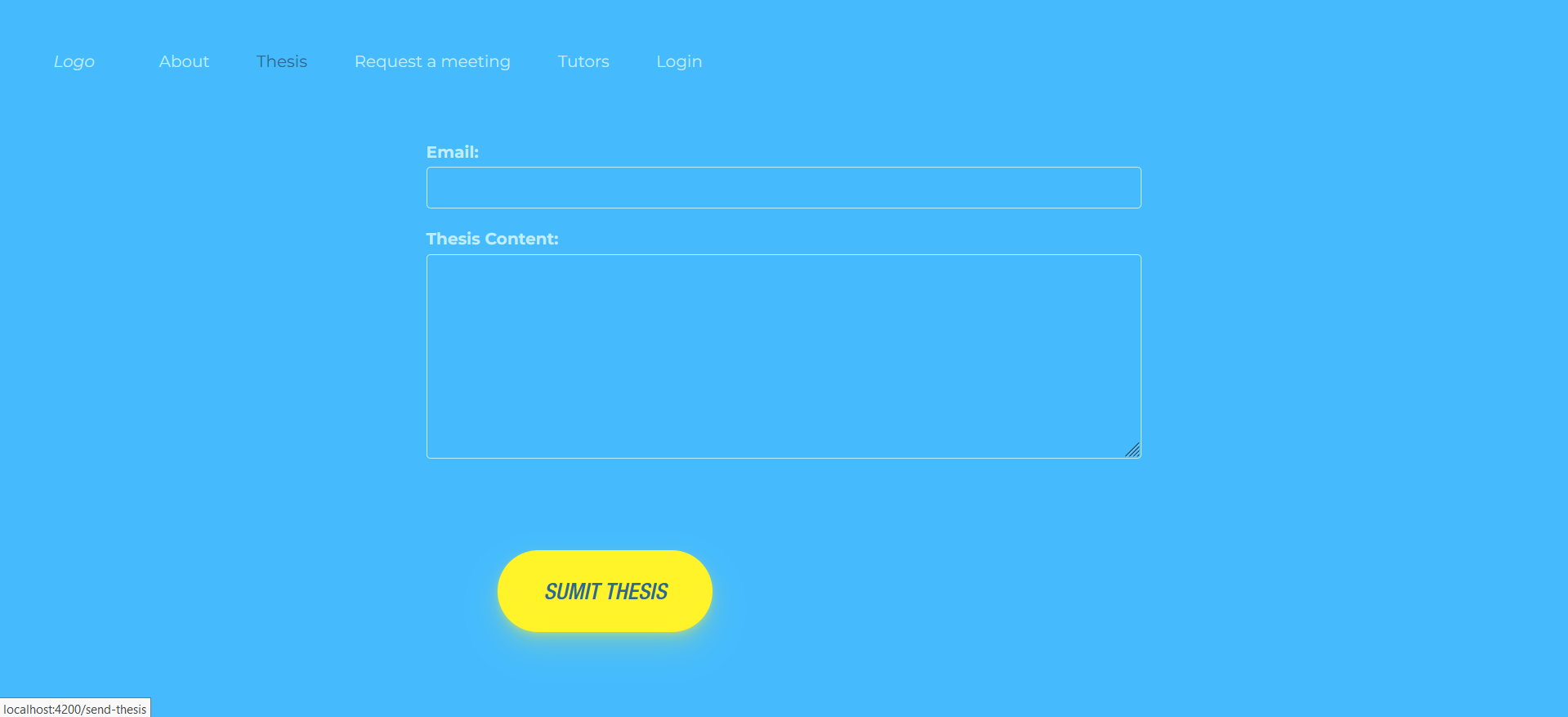
1. Прототипен потребителски интерфейс

Интерфейсът е имплементиран на Angular. За момента са планирани 5 страници. Прототипен е, защото единствената функционалност, която е свързана с бекенда е регистрирането на потребители и логин.

Първата страница е **About** page. Тя е първ в навигационния бар в горната част на екрана и съдържа кратко описание на приложението и бутон за бързо препращане към най-изпозлваната функционалност. Тук е визуализирано с извеждането на графиците на преподавател, но може и да се избере друга функционалност.

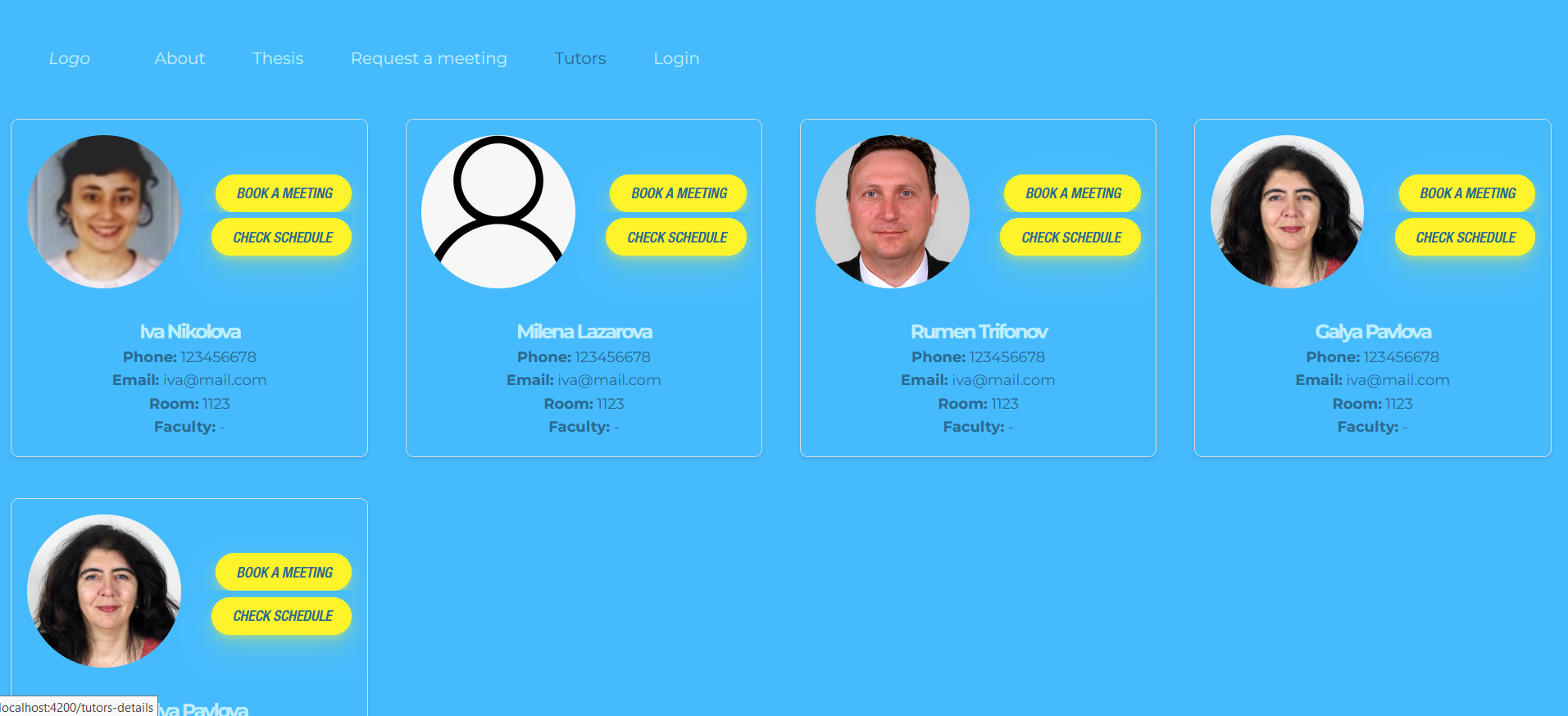


Следващата страница е **Thesis** page. Тя ще има два варианта в зависимост от ролята на потребителя. Студентът трябва да може да въведе своя идея за дипломна работа.Тук е нужна доста проста форма, която да приеме имейл адрес и описание на идеята. Възможно е да се промени логиката и да се взима имейл адреса директно от профила на студента, но за момента е оставено с поле за него. Оттук при натискане на бутона, ще се очаква от системата да запази дипломната идея в базата. Страницата за преподавателите ще се очаква да показва списък с всички подадени идеи за дипломни работи без подател, анинимни. Ще е нужен бутон за изпращане на заявка от преподавателя с желание за поемане на идеята. Тази логика още не е имплементирана.



Следващата страница би била за уговаряне на срещи с преподавател, **Request a meeting**  от менюто в горната част. За нея не е решено как ще изглежда все още, тъй като и логиката още не е измислена около визуализирането и обектите в бекенда, които ще се позлват. Основната идея е да се избере преподавател през търсачка и да се визуализират часовете му за посещения или само свободните такива. При изборане на час да има бутон и да се изпраща заявка и съответно мейл до преподавателя.

Последната страница за автентикирани потребители е **Tutors** page. Тук трябва да се изведе списък с всички преподаватели и техните най-важни детайли за връзка с тях. Тези детайли ще се въвеждат от преподавателите лично, тоест се очаква да са надеждни и актуални. За момента е изготвена следната визуализация.



Последната страница е **Login** page. Най-вероятният сценарий ще бъде да се скрива от навигационния бар при автентикиран потребител и да е единствената страница, която потребителят може да достъпи, ако не е. Би било интересно да се пробва някаква интеграция с Moodle или Teams вместо Google и Facebook както е в момента на визуализацията. Но това е само идея за момента. При успешен логин, потребителят бива редиректнат към **About** страницата.

A screen shot of a login screen

Description automatically generated

1. Заключение и бъдещи подобрения

TU-Connect успешно въвежда редица иновативни функционалности, които значително подобряват комуникацията и взаимодействието между студенти и преподаватели. Постигнат е успех в реализацията на системата за предложения на дипломни работи от студенти, както и възможността за преподавателите да преглеждат тези предложения и да подават заявки за поемане на ролята на дипломен ръководител. Това спомага за по-ефективно управление на дипломните проекти и стимулира активното участие на студентите в образователния процес.

Също така, системата успешно предоставя информация за преподавателите в удобен за използване списък, което улеснява студентите при избора на подходящи преподаватели за техните научни интереси. Това допринася за повишаване на достъпността и прозрачността на информацията, което е от съществено значение за ефективното функциониране на академичната общност.

Въпреки че все още предстои работа по някои аспекти като усъвършенстване на графиците на преподавателите и оптимизиране на потребителския интерфейс, основата, която TU-Connect вече е заложила, е твърда и обещаваща. Предвидените подобрения в областта на визуализацията и управлението на графиците и уговарянето на срещи ще допринесат допълнително за улесняването на взаимодействието между студенти и преподаватели, както и за подобряване на общото потребителско изживяване.

TU-Connect вече е важна стъпка напред в подобряването на академичния опит и с нетърпение очакваме бъдещите ѝ разработки и иновации.

Източници

1. [Technical University of Sofia](https://www.tu-sofia.bg/department/preview/27?dep_id=90)
2. [Enabling Cross Origin Requests for a RESTful Web Service](https://spring.io/guides/gs/rest-service-cors/)
3. [Angular - HTTP POST Request Examples](https://jasonwatmore.com/post/2019/11/21/angular-http-post-request-examples)
4. [Angular 9 - Basic HTTP Authentication Tutorial & Example](https://jasonwatmore.com/post/2020/04/29/angular-9-basic-http-authentication-tutorial-example)
5. [Cookie-based JWT Authentication with Spring Security](https://medium.com/spring-boot/cookie-based-jwt-authentication-with-spring-security-756f70664673)
6. [How to Use Cookies in Spring Boot](https://dzone.com/articles/how-to-use-cookies-in-spring-boot)
7. [Github - jsonwebtoken](https://github.com/auth0/node-jsonwebtoken?ref=blog.angular-university.io)
8. [Angular Authentication With JSON Web Tokens (JWT): The Complete Guide](https://blog.angular-university.io/angular-jwt-authentication/)
9. [JWT.IO](https://jwt.io/)
10. [Github - Flex Cheetsheet](https://yoksel.github.io/flex-cheatsheet/#section-flex)
11. [Angular - Common Routing Tasks](https://angular.io/guide/router)

Кодът може да бъде разгледан тук -> [Github repository](https://github.com/gretaptrv/TU-Connect)

[Clone using the web URL](https://github.com/gretaptrv/TU-Connect.git)