

**Дипломна работа**

**Тема:**

„Проектиране и реализация на система за прием и обучение на докторанти“

Дипломант:......................... Ръководител: …………

/Илиян Кирилов Костов/ /преп. инж. Стефка Попова/

фак.№ 23651224

Специалност : СИ

[1 Увод 2](#_Toc38459790)

[1.2 Ръководене на университет в дигитална среда 3](#_Toc126678352)

[1.3 Предимства и недостатъци 4](#_Toc170909910)

[1.4 Цел на системата 5](#_Toc1830869249)

[1.5 Дефиниране на целите на бизнеса спрямо проучването на конкурентите 5](#_Toc2101042557)

[2.0 Анализ на проблема и избор на технологии 10](#_Toc325977249)

[2.1 Инженерен проблем 10](#_Toc1319218649)

[2.2 Избор на технологии 12](#_Toc1129978892)

[2.2.1 Език за програмиране 13](#_Toc5472366)

[2.2.1.1 Java 14](#_Toc1271199160)

[2.2.1.2 Javascript 15](#_Toc741488864)

[2.2.2 Фреймуърк (Framework) 15](#_Toc1015563888)

[2.2.2.1 Quarkus 15](#_Toc429635849)

[2.2.2.2 ReactJS 16](#_Toc999930675)

[2.2.2.3 NextJS 16](#_Toc1153601755)

[2.2.3 Контейнеризация 17](#_Toc547036933)

[2.2.3.1 Docker 17](#_Toc611386623)

[2.2.3.2 Kubernetes 18](#_Toc1569244292)

[2.2.4 База от данни 20](#_Toc679580664)

[2.2.4.1 PostgreSQL 20](#_Toc258430973)

[2.2.4.2 ElasticSearch 21](#_Toc945793300)

[2.2.5 Автоматизация 21](#_Toc1516753555)

[2.2.5.1 GNU Make 21](#_Toc165888900)

[2.2.5.2 Nix 22](#_Toc690466757)

[2.2.5.3 Woodpecker CI 23](#_Toc144334387)

[2.2.6 Облачни системи 23](#_Toc699819643)

[2.2.6.1 AWS (Amazon Web Services) 23](#_Toc1099664726)

[2.2.6.2 Azure 23](#_Toc198618761)

[2.2.7 Мониторинг 24](#_Toc806040531)

[2.2.7.1 Grafana 25](#_Toc329335976)

[3.0 Проектиране на системата 27](#_Toc1690775593)

[3.1 Структура на проекта 27](#_Toc1336825305)

[3.2 Дефиниция на модулите на системата 29](#_Toc1070747272)

[3.3 Проектиране на отделните модули 30](#_Toc60271199)

[3.4 Прототип на потребителския интерфейс 31](#_Toc917449634)

[3.5 Архитектура на високо ниво 52](#_Toc60338760)

[3.5.1 UML диаграма 53](#_Toc1931020416)

[3.5.2 Site-map диаграма 53](#_Toc260127736)

[3.5.3 Mind-map диаграма 54](#_Toc697321030)

[3.5.4 Концептуален модел на базата от данни 54](#_Toc1567622411)

[3.5.5 Модел на Чен диаграма 63](#_Toc1317920714)

[3.5.6 Sequence диаграма 64](#_Toc1601103707)

[4.0 Реализация на системата 65](#_Toc1420251651)

[4.1 Първоначална настройка на програмната локална и облачна среда 65](#_Toc231411679)

[4.1.1 Сървърно приложение 66](#_Toc1844628610)

[4.1.1.1 Windows 66](#_Toc1157940417)

[4.1.1.2 Linux 66](#_Toc580852463)

[4.1.2 Клиентско приложение 71](#_Toc532366773)

[4.1.2.1 Windows 71](#_Toc1067739449)

[4.1.2.2 Linux 72](#_Toc353640521)

[4.2 Функционалности на приложението 72](#_Toc429170094)

[4.3 Нефункционалности на приложението 74](#_Toc1492589162)

[4.4 Удостоверение 74](#_Toc180539075)

[4.5 Автоматизация на внедряването на приложението 76](#_Toc358305818)

[4.6 Изисквания към апаратното осигуряване 79](#_Toc1271494590)

[4.7 Инструкции за поддържане на системата 79](#_Toc1321500712)

[4.8 Възможности за развитие на програмата 81](#_Toc1323751225)

[5.0 Тестове и резултати 83](#_Toc323293327)

[5.1 Тестване на бакенд средата 83](#_Toc1067416902)

[5.1.1 Unit тестове 83](#_Toc930985077)

[5.1.2 Integration тестове 87](#_Toc1043362540)

[5.2 Тестване на фронтенд средата 92](#_Toc891173646)

[5.2.1 Резултати при Unit и Snapshot тестове 92](#_Toc1049345377)

[5.2.2 Резултати при E2E тестове 98](#_Toc1605319550)

[6.0 Заключение 98](#_Toc124815580)

[7.0 Литература 99](#_Toc2060241297)

[7.1 Цитати 101](#_Toc1576629322)

[7.2 Глосар 101](#_Toc1798440722)

# **1** **Увод**

Образованието е един от най-мощните инструменти за формиране на обществото. Когато образованието е структурирано на индивидуално ниво, най-важният резултат от това е организацията и изпълнението на най-важната функция - откриване на таланти и иновация. По този начин студентите могат да бъдат ефективни в социалното, икономическо и политическо ниво. За да се удовлетворят тези очаквания, учебните програми трябва да бъдат създадени базирано от научни и систематични разбирания.

Учебните програми могат да бъдат дефинирани като детерминанта на човешките характеристики, които могат да бъдат поставени като образователна цел - където има начало и край.

Дигиталното образование има за цел да даде възможност на студентите да се развият в личният и професионален живот и да се интегрират в пазара на труда и да заемат висококвалифицирани работни позиции.

Дигитализиране на учебната програма преобразува учебните матирали към персонализирана колекция за индивидуален студент и едновременно се придържат към общите цели на обучението. Видове ресурси, които не могат да се репликират в традиционният начин на обучение са видео клипове, аудио и интерактивно съдържание.

Дигиталната трансформация на образованието изисква постепенно подновяване и добавяне на нови функционалности в обучаващата система, което може да влоши обучаващият и административен процес на студентите и преподавателите. За това и необходимо първо да се анализира маркета на съществуващите обучаващи системи, да се направят изводи за дигиталните изисквания и накрая да се разберат нуждите на съвременно образование, които не могат да се интегрират в традиционният образователен процес.

## **1.2 Ръководене на университет в дигитална среда**

Дигиталното образование има за цел да даде възможност на студентите да се развият в живота, да станат ангажирани граждани и да се интегрират по -добре в пазара на труда във все по-дигитализираният свят [1].

Дигиталната среда на образованието и науката представя нови изисквания за обучение на дипломирани и докторанти, така и развитието им за изграждане на инженерно мислене.

Взаимодействието между университети и бизнес индустрии по отношение на дигитална трансформация и дигитализация се счита за необходимост за развитието и модернизацията на университети.

*Еванс, Миклосик и Дю (2023) [3]* счита че сътрудничеството между бизнеса и университетите се добива ценни ресурси, като създаване на стандарт за добре свършена работа, създаване на повече възможности и подходи за обучаване на студент и преподаване на учебен материал, така и в резултат ще доведе до финансова печалба и повече кариерни възможности.

Те също твърдят че релациите между университети и бизнес индустрии се променят от физическо към виртуален и дигитален формат.

Накратко, процесът на дигитализация в университетите представлява процес на трансформация, в което нейният успех повече зависи от какви са съществуващите процеси на университета за цялостното обучаване на студента, отколкото да зависи изцяло само от техническата част. Адаптирането на дигитализацията в университетите отваря възможности за подобрено обучение на студента, по-добро сътрудничество между студента и връстниците му и пряк достъп до информация и учебни материали. Тези аспекти също могат да се доразвият като се направят интерактивни и забавни за студента в дигиталната образователна система.

Ползите се простират както за студенти, така и за преподаватели. Дигитализацията също дава възможност на университетите да си подобрят административните си процеси, подобрят учебните си програми и цялостното развитие на образованието.

## **1.3 Предимства и недостатъци**

Макар че дигитализацията може на пръв поглед да изглежда да бъде цялостно предимство за студентите и университетите, трябва да вземем предвид че не всеки студент е еднакъв в начина и средата за усвояване на знания. Дори ако процента на студенти, които желаят да преминат на традиционният вид обучение е по-малък, в сравнение с студентите които желаят да проведат обучението си изцяло дигитално, това не доказва че дигитализацията е по-добрия вариант на обучение, а по-скоро приоритизиране на комфорт мулти-таскване отколкото отнемане на цялото си внимание към материала и социализиране и споделяне на знания с връстниците си.

Ето няколко фактора, които дигитализацията вреди обучението на студентите:

* проблеми с предоставянето на образователно съдържание, като се вземат предвид възрастта на студентите
* характеристики на студента, свързано с ефективният начин на учене
* При дигитализация се увеличава зависимостта за мотивация за самообразование отколкото мотивация която е окуражена от преподавателите

Преподавателите в дигитална среда на обучението губят контрол върху класната стая, като не могат да предоставят материала към студентите си в по правилният начин. Те също не могат да разберат дали този техен начин на преподаване е ефективен или не, особено ако студентите не са си включели уеб камерите си по време на учебен час. За компенсация на това, повечето преподаватели прочитат съдържанието на учебния материал ред по ред, т.е. не се опитват да направят студентите по-заинтересовани и интерактивни в материала, който преподавателите преподават и го променят в такъв начин.

Това в последствие правят студентите да не задават повече въпроси свързано с материала и целият проведен час става ‘скучен’ за всички участващи.

Във висшето образование, взаимодействието на ‘преподавател-студент’ е фундаментна част в образованието, като показва как процеса на обучението се провежда. Терминът ‘дигитални жители’ *(digital natives) [2].* Според М. Пенски, дигиталните жители са опитни в мулти-таскването , те разбират как да използват дигиталните технологии за тяхно предимство и считат дигиталният свят като основната им част от ежедневието си. Те са свикнали с внезапната промяна на средата и да се адаптират бързо спрямо нея. Следователно тези дигитални жители им е трудно да преминат в традиционната среда на обучение, където там средата е в пъти-по различна.

Те повече предпочитат да се самообразоват (т.е те са независими), и предпочитат да се обучават по тяхната си скорост.

М. Пенски също сравнява студентите, които са станали жертва на дигитализацията и са се превърнали в ‘дигитални имигранти’ *(digital immigrants)[2].* Тези студенти по-скоро са от по-старо поколение, където не са се родили в дигиталната ерата.

Независимо от това, дигитализацията създава повече свобода на студентите и им дава предимства, които не са възможни в традиционното обучение:

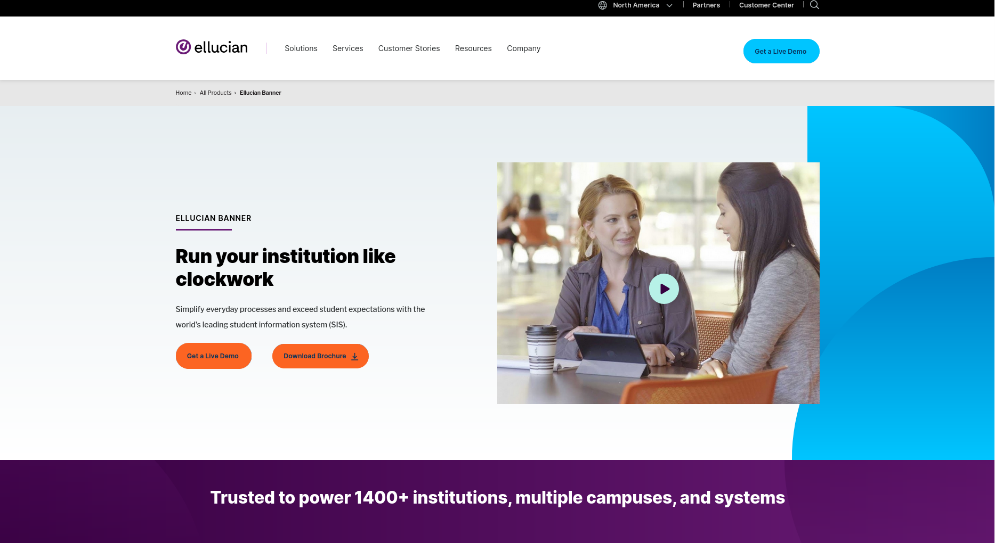
* възможност за мулти-таскване на допълнителни задачи
* по-бързo усвояване на информация
* По-добро Превключване от една задача в друга
* Не са ограничени да провеждат обучението си само в университета, а навсякъде
* Намалени разходи на персонал и поддръжка от страна на университетите
* Отваряне на възможност за обучение на международни студенти

## **1.4 Цел на системата**

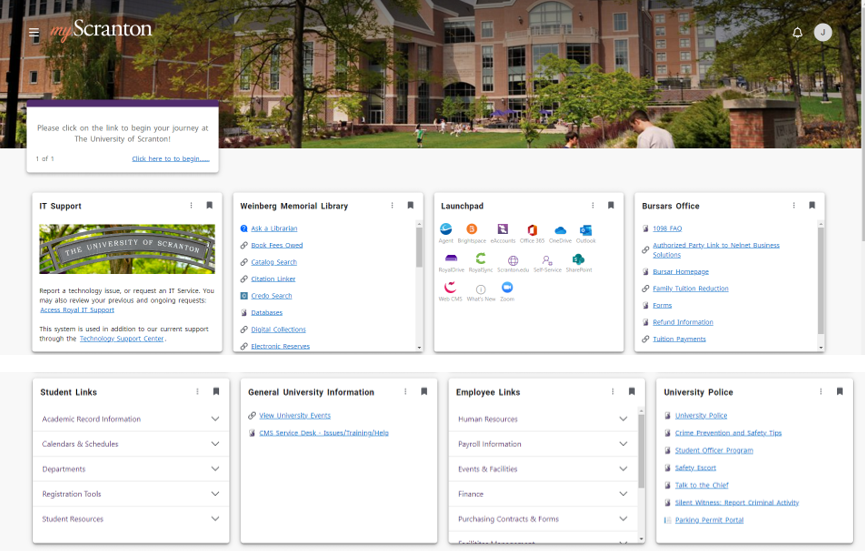
Проекта следва да разработи инженерно решение на дигитализиране на информационна система за докторанти и кандидати, с цел да бъде използван от висшите учебни заведения, който би се интегрирал в систематичния процес за ръководене и управление на така нареченият „дигитален университет“.

## **1.5** **Дефиниране на целите на бизнеса спрямо проучването на конкурентите**

**Ellucian**



Фигура 1 Фигура 1 Примерна страница на Ellucian приложение



Фигура 2 Начална страница на Ellucian приложение

Характеристики :

Ellucian е облачно базирано приложение, който помага на висшите учебни заведения с професионални решения да създадат свързан кампус. Създадена е за университети от всякакви размери, като предоставя решения за финанси, студентски услуги, човешки ресурси, приемане на персонал и прием.

Модулите на Ellucian включват планиране на ресурси на предприятия с продукти като Banner, PowerCampus. Elevate and Quercus: Управлението на отношенията на състава предлага продукти като: CRM Advise, Degret Works, CRM набиране и CRM Advance.

Предимства

* цялата информация за студента е на едно място
* интуитивен потребителски интерфейс
* Централизирани функционалности за по-добра ефективност от страна на потребителя
* Помагайки на студентите със средства за самообслужване на платформата
* Анализ в реално време за по-интелигентни решения
* Достъп до текущо обучение и консултации

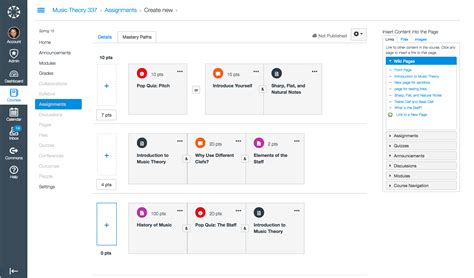
Недостатъци

* Остарял и излишен дизайн на потребителския интерфейс
* Трудно използване на платформата за нови потребители
* Данните са съхранени в общ формат, като създава миграцията към друга платформа трудна.

**Canvas**



Фигура 3 Примерна страница на Canvas приложение



Фигура 4 Начална страница на Canvas приложение

Характеристики :

Canvas е уеб-базирана система за управление на обучение на студенти. Използва се от учебни институции, преподаватели и студенти за достъп и управление на материали за онлайн курсове и комуникация за развитието на обучението на студента.

Canvas включва разнообразие от персонализирани инструменти за създаване и управление на курса, анализи на статистически данни за курсове и потребители във вътрешни средства за комуникация.

Институциите могат да предоставят на потребителите профил в Canvas или отделните потребители могат да пробват безплатната им версия, като се регистрират на собствен личен профил.

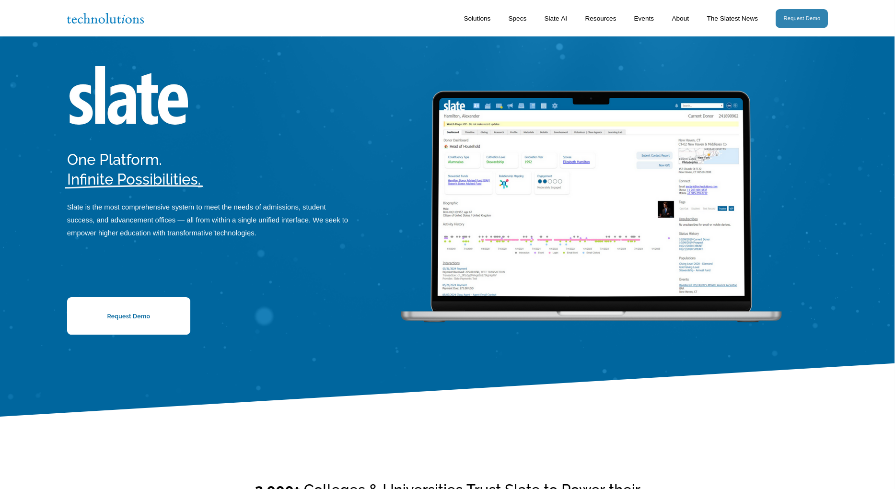
Предимства

* Добър потребителя интерфейс
* Надеждна поддръжка на крайните потребители (*customer support*)
* Редовни актуализации и подобрения на платформата
* Мобилна достъпност
* Персонализиране на платформата според нуждите на потребителя
* Интеграция с Microsoft Teams

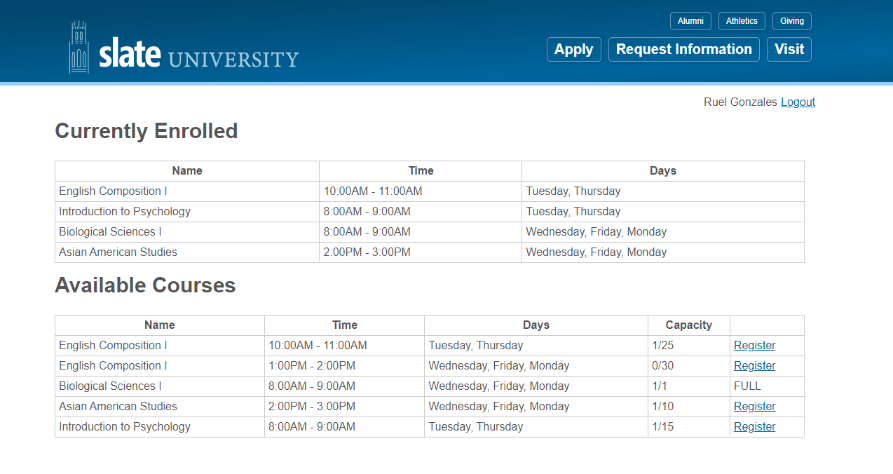
Недостатъци

* Трудно използване за сложни функции
* Неконсистентно изпълнение в различните браузърите
* Цена за определени институции
* Лимитирана функционалност за офлайн режим

**Slate**



Фигура 5 Примерна страница на Slate приложение



Фигура 6 Начална страница на S/ate приложение

Характеристики :

Slate AI е платформа с изкуствен интелект, предназначен да подобри възможностите на Slate, цялостната платформа за управление на приема и записванията. Той използва машинно обучение за да предостави на институциите средства за анализ на данни, прогнозно моделиране и подпомагане при вземане на решения. Основната цел е да оптимизира процесите на набиране и прием, като предлага анализи, които водят до по-информирани и стратегически действия.

Той разполага с функционалности като визуализация на данни, които трансформират сложни набори от данни в интуитивни и интерактивни визуални представяния. Тези визуализации помагат на потребителите бързо да разберат ключови показатели и тенденции, улеснявайки по-доброто разбиране и комуникация на базирани на данни анализи. Той поддържа широка гама от типове диаграми и опции за персонализиране, което позволява на институциите да приспособят визуалните отчети към своите специфични нужди.

Платформата предлага възможности за обработка на естествен език (NLP), което му позволява да анализира и интерпретира текстови данни от различни източници, като есета за кандидатстване или препоръчителни писма и социални медии. Тази функция помага на институциите да получат по-задълбочена представа за мотивацията, интересите и потенциалното им съответствие с институцията. Чрез използване на естествения език за учене (NLP), Slate AI може да подобри цялостния процес на преглед и да подпомогне по-нюансирано вземане на решения.

Потребителите могат да генерират подробни отчети за различни аспекти на процеса на прием, от тенденциите в кандидатстването до процентите на успеваемост. Тези отчети предоставят обратна връзка, която може да информира усилията за непрекъснато усъвършенстване. Аналитичните възможности поддържат сравнителен анализ с подобни институции, помагайки на потребителите да идентифицират силни страни и възможности за растеж.

Предимства

* Средства за управление на събития
* Заявки за идентифициране на различни подмножества от населението в рамките на набора от кандидати
* Проследяване на депозитите за записване на студенти

Недостатъци

1. Възможност за регистрация на студенти на събитие, без да е необходима Wi-Fi мрежа
2. Възможност за стартиране на приложение, без да е необходимо веднага да се задава потребителско име и парола
3. Трудно за актуализиране на записи за лица, които се местят в друга организация

# **2.0** **Анализ на проблема и избор на технологии**

## **2.1 Инженерен проблем**

След като сме вече задали целите на бизнеса спрямо проучването на конкурентите, техните предимства и недостатъци и сме изградили прототип на потребителския интерфейс за уеб-приложението, имаме ясен изглед в необходимите функционалности на приложението, което ще помогне в процеса на създаване на информационна система за докторанти в ефективен и по-бърз начин.

Изисквания на приложението:

Целта на този проект е да се разработи приложение за прием и обучение на докторанти, което заимствано ще помогне както на административните лица да управляват достъпа към системата, така и на обучаващите се (факултетен персонал) и обучавани (докторанти) лица, които ще използват системата с цел подготвяне на докторанти да се специализират в техните си научни дисциплини.

Също така е необходимо приложението да покрива основните функционалности за обучаването на докторанти .

Задачи на приложението:

1. **Системен дизайн**:

Проектиране на удобен потребителски интерфейс, осигуряващ безпроблемна навигация между кандидатури, отчети, изпити и администрация. Внедряване на системата за управление на идентичността е също задължително.

2. **Интеграция**:

Интегриране на бакенд средата с фронтенда чрез създаване на единен интерфейс за комуникация.

Също така ще бъде необходимо интеграция с облачно базирани услуги (Azure Active Directory за управление на идентичността, съхраняване на файлове с AWS S3, хостване на приложенията във виртуална машина AWS EC2)

3. **Автоматизирано** т**естване**:

Тестване на пълната функционалност и надеждност на системата, включително фронтенд и бакенд средата, чрез създаване на автоматизиран процес при всяка промяна на сорс кода.

4. **Среда за разработване**:

Създаване на среда на разработване за разработчиците на приложението, която е лесна за разбиране, бърза за стартиране и е възпроизводима *(reproducable)*. Средата за разработване трябва да описва всички зависимости на приложението по декларативен начин.

5. **Система за уведомяване**:

Внедряване на стабилна система за уведомяване, която информира членовете от комитета за оценяване на изпит на докторанта директно от приложението.

Също така ще е необходимо да се разработи система за известяване на кандидатите за докторантура относно техния статус на приемане или отказ чрез имейл съобщение.

**6. Административен панел**:

Разработване на сигурен и интуитивен административен панел, който позволява лесно управление на потребителите (докторанти, членове от комисията и членове на докторантски център).

**7. Поддръжка на качество на сорс кода** :

Поддръжка на еднакъв стил и формат на сорс кода и интеграцията му в автоматизирано тестване. Този вид тест подпомага поддръжката на кода, когато се поддържа повече от 1 разработчик.

**8. Мониторинг**:

Създаване на визуализации на състоянието на бакенд и фронтенд средата, когато настъпи проблем в продукционната среда, да може необходимите хора да се известят веднага и да могат да разрешат проблема възможно в най-скоро време.

## **2.2 Избор на технологии**

Необходимите програмни компоненти за имплементирането на информационната система са разпределени по следните категории:

* Език за програмиране: ще бъде необходимо да създадем клиентското и сървърно приложение. За да може да се направи правилно решение за програмните езици, е необходимо да има платформена преносимост.
* Фреймуърк (Framework): тъй като заданието за създаване на обучение и приемане на докторанти е сложа задача, ще бъде необходимо да се използват технически средства за улесняване на тази имплементация. Фреймуърка предоставя такиви функционалности и интеграция с външни облачни системи.
* Контейнеризация: след когато клиентското и сървърно приложение са разработени и готови за *production [1]*, внедряването му ще изисква да се контейнизират двете приложения като ги изолираме извън операционната система на компютърната машина, която ги поддържа, с цел да се намали *attack vector [2]* от хакери.
* База от данни: тъй като системата ще поддържа отчети, оценки, дисциплини, учебни планове, нотификации, данни за докторанта, членове от комитета и докторантския център, кандидатите и администраторите и т.н., ще бъде необходимо да се съхраняват тези данни, а да не се изгубят в случай при проблем от сървърното приложение
* Автоматизация: за да може да се поддържа добра практика през целия *Software Development Lifecycle [3], ще бъде необходимо през всичките фази на разработването на приложенията да се автоматизира колкото се може най-много – примерно тестването, преди новите промени да се внедрят в основната версия на приложението.*
* Облачни системи: най-съвременните системи следват *Micro-architecture [4]* парадигмата, сървърното и клиентско приложение ще имат необходимостта да взаимодействат с облачни услуги, с основната цел да бъдат колкото се възможно *loosely-coupled [5]*.
* Мониторинг: в случай че *production [1]* средите на клиентското и сървърно приложение имат проблем – дали свързано с мрежата, *bugs [6]*, недостиг на изчислителен ресурс и т.н., е необходимо да се известят необходимите членове за поддръжка на софтуера, за да могат да отстранят проблема във възможно най-скоро време.

### **2.2.1 Език за програмиране**

Правилният избор за използване на програмен език е важна част за постигане на успеха на софтуерният проект, и това решение ще влияе не само на оптималното изпълнение на софтуера, но и на разработчиците, които ще го поддържат.

Има няколко *antipattern (антипатерн) [7],* които разработчиците постоянно се объркват при избора на програмен език, които ще бъдат описани само някои от тях.

*1.* Golden hammer

Това е *antipattern [7]* в софтуерното инженерство, където разработчикът се опитва да реши всички проблеми, използвайки познат инструмент, технология или подход - независимо дали е най -подходящ. Той отразява мисленето на „Ако всичко, което имате, е чук, всичко изглежда като пирон“, често води до твърди, неефективни или прекалено сложни решения.

Най-често се получава този *antipattern [7],* когато в екипа на софтуерният екип има опитен разработчик на определен програмен език и може да разреши всякаква програмна задача с програмния език, с който е най-опитен.

2. Популярност

Този *antipattern* се получава не според бизнес изискванията или оптималното изпълнение на софтуера, а защото в момента на избора, това е бил най-популярният програмен език, а не считат опита в екипа или бизнес изискванията.

За да се предотвратят тези анти-патерни, разработчиците първо трябва да си зададат следните въпроси:

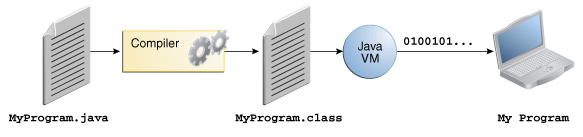
* кой е общо използваният програмен език в институцията ?
* приложението трябва ли да бъде оптимално изпълняваш се ?
* лесно ли е за намиране на нови разработчици с опита за програмният език ?
* лесен ли е за поддръжка ?
* често поддържан ли е от общността (community) ?

Ако на всички тези въпроси отговорът е „да“, тогава този програмен език е подходящ и може да се премине към фазата на планиране

#### 2.2.1.1 Java

Java е обектно-ориентиран програмен език на високо ниво, който е бил създаден през 1995 г. от Sun Microsystems.

Изходния код е първоначално написан първо в текстови файлове, завършващ с разширението .java. След това тези изходни файлове се компилират от Javac компилатора в разширение .class. Този файл съдържа така наречения байткод (bytecode) - машинният език на Java Virtual Machine (JVM). Накрая тези файлове се изпълняват от Java виртуалната машина (Java Virtual Machine JVM).



Фигура 7 Плоцес на JVM машина

#### 2.2.1.2 Javascript

JavaScript първоначално е създаден, за да „направи уеб страниците живи“.

Програмите на този език се наричат „скриптове“. Те могат да бъдат написани директно в HTML и да се изпълняват веднага, когато страницата се зарежда.

Този програмен език е интерпретатор, за разлика от компилаторните програмни езици като Java.

Интерпретаторът превеждат всяка инструкция ред по ред, в сравнение с компилаторът, което ги превежда всичко наведнъж. Интерпретаторът също създава нови редове от изходен код при всяко изпълнение, вместо да го произвежда всичко наведнъж, което означава че понякога интерпретаторът може да работи по-бавно от компилирания код.

### **2.2.2 Фреймуърк (Framework)**

Фреймуърка е структура, върху която служи като шаблон за изграждане на приложение. Те са съвкупност от класове и структури от определен програмен език, които предоставят редица функционалности, пакетирана и готова за използване (още наречен „*out of the box*“), за да може разработчика да се фокусира върху бизнес логиката, а да не си губи времето, опитвайки се да създаде нещо, което някой друг вече е създал (т.е *reinventing the wheel*).

#### 2.2.2.1 Quarkus

Quarkus е фулстак Фрейуърк на Java, направена за виртуални машини на Java (JVM) и така наречените „местни изпълними файлове“ (*native executables*), оптимизиран за контейнери, която позволява Java приложението да се превърне в среда без сървър (още наречено „*serverless*“), или да се внедри в облачната среда или да се контейнизира в Kubernetes.

Основния дизайн-патерн на който Quarkus работи e с инжектиране на зависимостите (*dependency injection*), който е подобен на Java Spring Фреймуърка. И двата Фреймуърка се основават на CDI (контексти и инжектиране на зависимост), която позволява на всеки Java компонент да се разшири и конфигурира според архитектурното решение на разработчика .

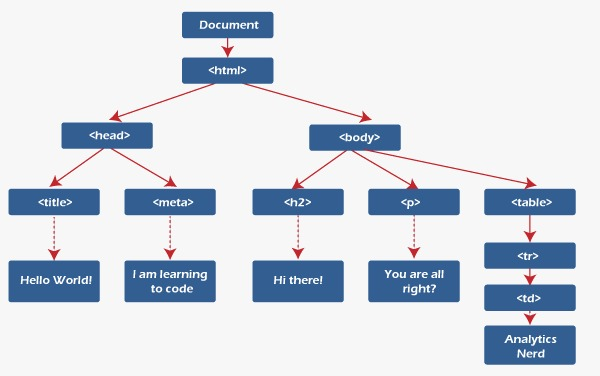
Той също може да взаимодейства с Graalvm машината (универсална виртуална машина за изпълнение на приложения) за изпълнение на местните изпълними файлове.

#### 2.2.2.2 ReactJS

ReactJS е JavaScript библиотека, използвана за изграждане на компоненти за многократна употреба на потребителския интерфейс.

Тези компоненти представляват данни, които се променят с течение на времето. React създава абстракция върху документния обектен модел DOM (*Document Object Model[8]*), като предлага по-опростен модел за програмиране и по-добра производителност.

React по стандарт се компилира от страна на крайният потребител, но е възможно то също да се компилира то страна на сървъра с помощта на NodeJS.



Фигура 8 Дървовидна струцтура на ReactJS virtual DOM

#### 2.2.2.3 NextJS

Next.js е Фреймуърк върху React за създаване на фулстак уеб приложения.

Next.js предоставя допълнителни функционалности и оптимизации като навигация към страници, автентикация, сървърно компилиране, сървърни филтри и динамично HTML компилиране.

По подразбиране Next.js абстрахира и автоматично конфигурира пакетирането и компилирането с използването на „презареждане на горещ модул“ (*Hot Module Reloading* HMR).

### **2.2.3 Контейнеризация**

Контейнеризацията е пакетиран софтуер, съдържащ библиотеките и зависимостите на операционната система, необходими за стартиране на програма. Финалният резултат е изпълним файл, наречен още контейнер - който работи самостоятелно и е лесно преносим във всяка инфраструктура.

#### 2.2.3.1 Docker

Docker е платформа за клиент-сървър, т.е. дава възможност на множество клиенти да контролират внедряването на един сървър.

1. **Шаблон**

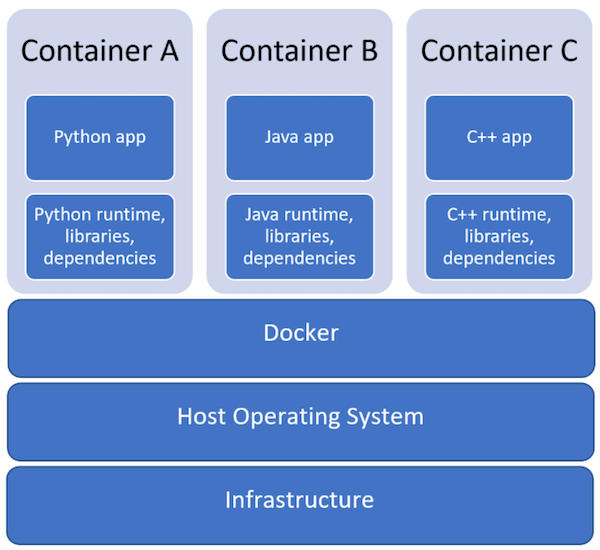
Преди да се създаде контейнер е необходимо да се създаде така наречения шаблон („*template*“), който съдържа последователни инструкции за изграждане на docker изображение, като първоначално трябва да се изпълнят тези инструкции за създаване на изображението.

2. **Изображение**

Изображението е изпълним пакет от софтуер, който включва всички необходими зависимости за стартиране на приложението. То информира *docker engine* как трябва да се създаде контейнер. След построяване на изображението се създава контейнера.

3. **Контейнер**

Контейнерите са виртуализирани среди за изпълнение на приложения. Той представлява пакет, съдържащ код, библиотеки, системни и програмни зависимости, необходими за стартиране на приложението. Всеки контейнер е независим и изолиран от останалите контейнери.



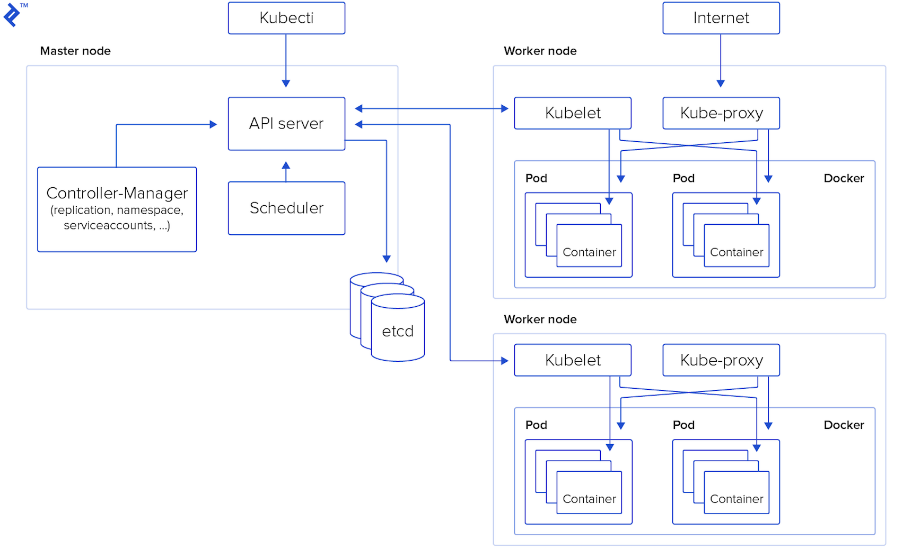
Фигура 9 Архитектура на Docker контейнер

#### 2.2.3.2 Kubernetes

Kubernetes се състои от контролен възел (*control node*) с набор от работни възли (*worker nodes*), които изпълняват контейнер приложенията. Всеки клъстер трябва да съдържа поне един работен възел, за да стартира така наречените вази (*pods*).

Работният възел хоства вазите, чрез което се разпределя натоварването на приложението. Контролния възел управлява работните възли и вази в клъстера.

**Архитектура**



Фигура 10 Архитецтура на master-worker node в Kubernetes

Компонентите на контролният възел управляват жизнения цикъл на клъстера, като стартиране на нова ваза, когато бройката на репликите е неудовлетворена.

**API сървър (*Apiserver*)**

Компонент, който разкрива API на Kubernetes към външния свят.

Той е предназначен с главната роля да мащабира клъстерът хоризонтално. Възможно е да се инициализират няколко инстанции на Kube-Apiserver и да се балансира трафика между тях.

**Etcd**

Хранилище със схемата ключ-стойност, което е висока достъпна, използвана за съхраняване на данни на клъстера.

**Планировчик (Scheduler)**

Компонент, който слуша и планира за новосъздадени вази без зададен възел, и избира съответния възел, който работи.

Изискванията за стартиране на планирането са: самостоятелни или набор от ресурсни изисквания, хардуерни или софтуерни ограничения, локалност на данните, намесване между работното натоварване.

**Мениджър на контролера (Controller-Manager)**

Компонент, който изпълнява процесите на контролера. Всеки контролер е отделен процес, но за да се намали сложността, всички те работят като едно цяло.

Има различни видове контролери. Някои от най-използваните са:

* **Контролер на възел:** Отговаря за известието, когато възлите спират.
* **Контролер на заданието**: Слуша за еднократни задачи с цел създаване на вази, за да изпълняват задачи.
* **EndPointSlice Controller**: Попълва обектите на EndpointSlice (за да предостави връзка между услуги и вази).
* **Контролер за услуга на профила (*ServiceAccount Controller*):** Създава ServiceAccounts по подразбиране за нови пространства от имена (*namespaces*).

**Облачен мениджър на контролера (*Cloud Controller-Manager*)**

Компонент, който внедрява специфичната за облака услуга в клъстера. Мениджърът на облачния контролер свързва API клъстера с облачния доставчик и отделя компонентите, които взаимодействат с тази облачна платформа от компоненти.

Облачният контролер-мениджър управлява само контролери, които са специфични за облачния доставчик.

Облачният мениджър на контролера и мениджърът на контролера комбинират няколко независими контролни цикъла в един общ, за да работи като един процес.

Съществуват контролери, които зависят от доставчика на облака, тези са:

* **Контролер на възела:** проверява доставчика на облака да определи дали възелът е изтрит в облака, след като спре да реагира.
* **Контролер на маршрута:** За настройка на маршрути в основната облачна инфраструктура.
* **Контролер на услугата:** За създаване, актуализиране и изтриване на балансиращи доставчици на облака.

### **2.2.4 База от данни**

#### 2.2.4.1 PostgreSQL

PostgreSQL е система за база от данни, която използва и разширява езика на SQL, комбиниран с множество функции и процедури, които съхраняват и мащабират данните.

**Функционалности**

**Обектно-релационен DBMS**

* данните са с обектно-релационен модел и са способни да обработват сложни процедури и правила. Примери са декларативните сложни SQL заявки, управление на последователността на версиите, многопотребителско управление , самите транзакции, оптимизация на заявките, наследяването и масивите.

**Разширяване**

* поддържа дефинирани от потребителя оператори, функции, методи за достъп и типове данни.

**Референтна цялост**

* използва се за осигуряване на валидността на данните в базата данни.

**Между-програмно езиково API**

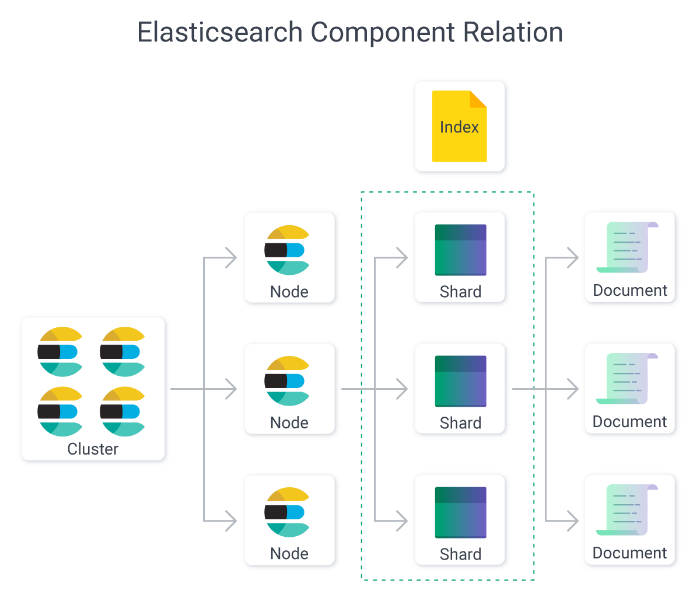
* позволява на доставчиците лесно да предоставят поддръжка за развитие на PostgreSQL. Те включват Pascal, Python, Perl, Php, Odbc, Java/JDBC, Ruby, TCL, C/C ++ и Pike.

#### 2.2.4.2 ElasticSearch

Elasticsearch е разпределена търсачка и анализ, с мащабируемо хранилище за данни и векторна база данни, изградена на Apache License.

Той е оптимизиран за скорост и релевантност на резултати при търсения от система или потребителя. Използва се предимно за търсене, индексиране, съхраняване и анализиране на данни от всички форми в почти реално време (*near real-time*).

В тази информационна система се използва главно за съхраняване и наблюдение на одит на събития (още наречен *logs*).



Фигура 11Компоненти на Elasticsearch

### **2.2.5 Автоматизация**

#### 2.2.5.1 GNU Make

Make изгражда изпълними програми и други файлове от изходния код на програмата.

Инструкциите на Make за изграждане на програмата е чрез файла Makefile, който изброява всеки от файловете, които не са източници. При Makefile се описват командите за инсталиране на зависимостите на приложението, тестване, изпълнение и т.н.



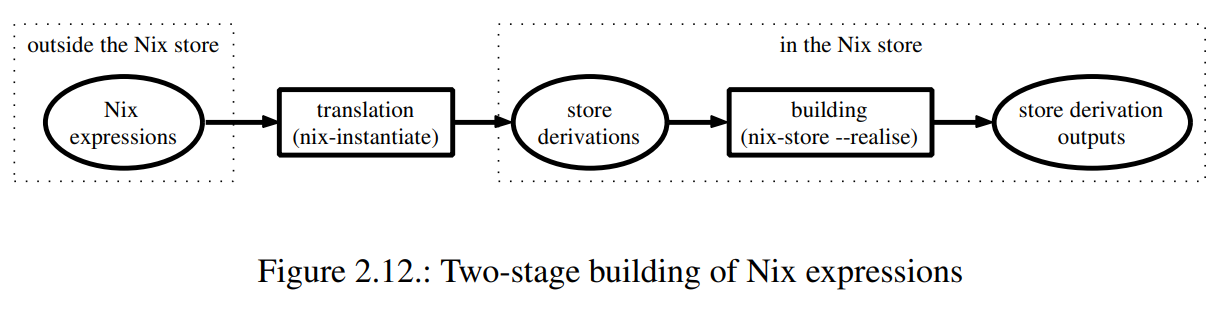
Фигура 12 Процеси на построяване на програма с GNU Make

#### 2.2.5.2 Nix

Nix е функционален мениджър на пакети. Това означава, че той третира софтуерните пакети като стойности на чисто функционални езици за програмиране както Haskell-те са изградени от функции, които нямат странични ефекти, и никога не се променят, след като са били изградени.

Nix съхранява всичките си инсталирани пакети в хранилище Nix, обикновено с директорията */nix/store*, където всеки пакет има своя уникална поддиректория като следния пример:

/nix/store/b6gvzjyb2pg0kjfwrjmg1vfhh54ad73z-firefox-33.1/



Фигура 13 Построяване на Nix derivation

където B6GVZJYB2PG0 ... е уникален идентификатор за пакета, който описва всичките му зависимости (това е криптографски хеш за зависимост от изграждането на пакета).

В информационната система се използва една индивидуална функционалност на Nix, която е свързана със създаването на среда на разработване – devenv.

Devenv е предназначен да подобри процеса на разработка, като осигурява бързи, декларативни, възпроизводим (*reproducable*) и композитни среди за разработване, използвайки Nix. Този опростява настройката, управлението и изпълнението на среди за разработване, което оптимизира работата на текущи и нови разработчици.

#### 2.2.5.3 Woodpecker CI

Woodpecker CI е лека, контейнер базирана система за продължителна интеграция (още наричан *continous integration*) с отворен код.

Woodpecker е клонинг (*fork*) от DroneCI, с разликата че се основава на свободно достъпна версия.

Той също може да се интегрира със системите за контрол на версиите (като GitHub, Gitlab и Gitea) за автоматизирано внедряване и тестване, при промяна на сорс кода.

### **2.2.6 Облачни системи**

#### 2.2.6.1 AWS (Amazon Web Services)

AWS е предоставена от Amazon, която използва разпределената ИТ инфраструктура, за да осигури различни ИТ ресурси, достъпни при поискване. Той предоставя различни услуги като инфраструктура като услуга (*IAAS*), платформа като услуга (*PAAS*) и пакетиран софтуер като услуга (*SAAS*).

AWS е платформа за облачни изчисления, която позволява на различните организации да се възползват от тяхната ИТ инфраструктура.

В информационната система на докторанти се използва AWS Amazon Simple Service Service (*Amazon S3*).

Това е услуга за съхранение на файлови обекти, предлагайки мащабируемост, наличност на данни и сигурност.

#### 2.2.6.2 Azure

Microsoft Azure е облачна компютърна платформа, която предоставя разнообразие от услуги, които не се изисква купуваме на хардуер от страна на крайният клиент. Той дава възможност за бързо развитие на решения и предоставя ресурсите при поискване (*on demand*).

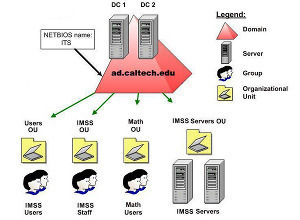
В информационната система на докторанти се използва Активна директория (*Azure Active Directory*).

Активната директория е създадена от Microsoft за Windows Domain Networks. Той е включен в повечето операционни системи на Windows Server като набор от процеси и услуги.

AD съхранява информация за обекти в мрежата и прави тази информация лесна за администраторите и потребителите да я намерят и използват.

**Основни компоненти:**

* **Домейн:** Домейнът е централна част от структурата на AD. Това е логична група от мрежови обекти, като потребители, компютри и устройства, които споделят една и съща база данни. Домейн се управлява от контролер на домейн.
* **Контролер на домейн** (DC): Сървър, който отговаря на заявките за удостоверяване на сигурността в домейн на Windows Server. Той съхранява информация за потребителския акаунт, удостоверява потребителите и прилага политики за сигурност за даден домейн.
* **Организационна единица** (OU): Това са контейнери в домейн, който може да обхваща потребители, групи, компютри и други OU. Те предоставят начин да организират обектите на директорията в логически йерархии, предлагайки по -подробно контрол върху управлението на тези обекти.
* **Глобален каталог** (GC): Това е разпределено хранилище на данни, което съдържа търсещо, частично представяне на всеки обект във всеки домейн в много-домейна рекламна гора. Той помага при намирането на обекти от всеки домейн, без да се нуждае от тяхното име на домейн.
* **Схема**: Схемата е компонент на AD, който определя всички обекти и атрибути, които услугата на директорията може да управлява. Той определя видовете обекти, които могат да се съхраняват в директорията и атрибутите на тези обекти.



Фигура 14 Домейн контролер в Active Directory

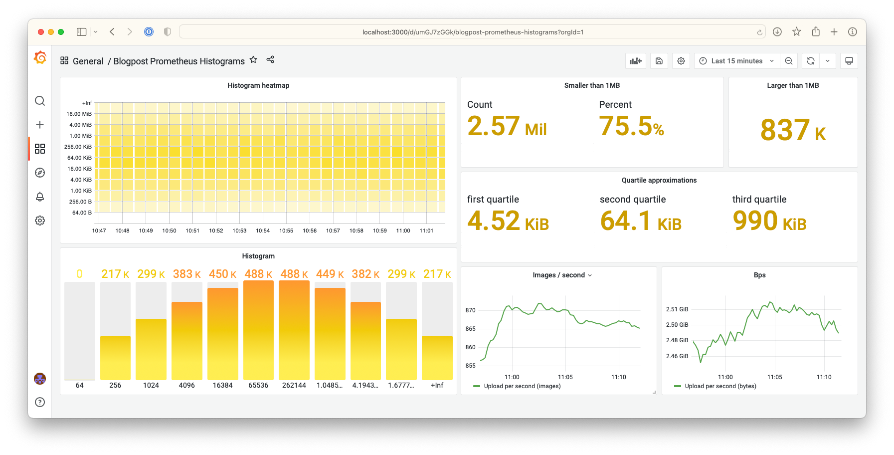
### **2.2.7 Мониторинг**

#### 2.2.7.1 Grafana

Grafana е уеб базирано приложение с публичен код за анализ и визуализация на показатели, събития и http следи и др.

Grafana основно стои във фазата за поддръжка в *SDLC [3],* тя може да се хоства локално или в облака, наричан Grafana Cloud.

Използва се за наблюдение на производителността на приложенията. Потребителите използват Grafana главно заради предимството че приемат данни от широк спектър от източници, също и за персонализиране на таблата и диаграмите си според различните типове потребители.



Фигура 15 Табло на управление в Grafana

Характеристики:

1. Визуализация:

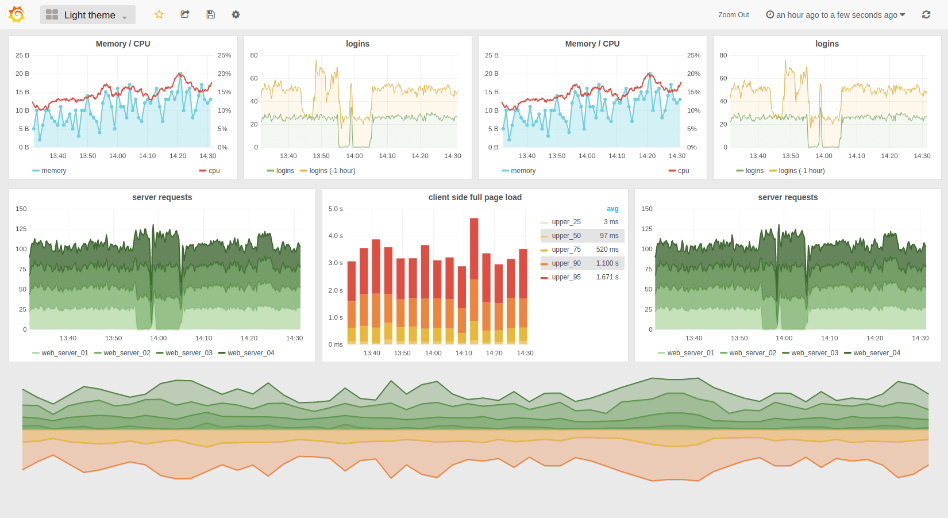
* визуализациите са разделени на „панели“, които след това се използват за изграждане на таблото за управление.
* панел е най-основният елемент на визуализацията в Grafana и се използва за визуализация на данни, а тези данни се изграждат от източник на данни, зададени на всеки индивидуален панел.
* Панелът може да бъде или вид графика (габарит, хистограма, лентова диаграма и т.н.) или регистрационни файлове или сигнали
* Панелите могат да се подредят по хоризонтален и вертикален начин, за съставяне на табло за управление, като всеки панел показва своя собствена информация

1. Сигнали

* Сигналите са предназначени за известие на потребителите, в случай ако приложението има повреда (мрежова, на програмно ниво, недостатъчни ресурси, т.н.). Grafana има вградена поддръжка за огромен брой канали (channels) за уведомяване, като имейл, Slack, Pagerduty, Microsoft Teams и т.н
* За създаване на сигнал, първо е необходимо да се създаде и конфигурира правило. Това правило служи като спусък (trigger) сигнал, така че всеки път когато посоченото правило е нарушено, известието се изпраща чрез канала за известие, който е конфигурирано в правилото

1. Анотации

* Анотациите са поставяне на коментари директно върху графиките. Тази функция е полезна за маркиране на важни точки на графиките, така и за напомняне за по-нататъшни действия в бъдеще.



Фигура 16 Табло на управление в Grafana

# **3.0 Проектиране на системата**

## **3.1** **Структура на проекта**

Структурата на проекта се основава на 2 различни типове системи: клиентска и сървърна.

**Клиентско приложение:**

Клиентското приложение е избран да бъде във формата на уеб приложение , като избран език за програмирането му е Javascript. Потребителският интерфейс е написан изцяло с помощта на NextJS и ReactJS фреймлурка, където уеб страницата (или още наричан *Document Object Model[8]*) се презарежда само за компонентите, които се променят, а не да се зареди цялата уеб страница само заради един компонент.

За избор на софтуерната архитектура е следване принципа на файлов базиран маршрут (file-based routing) моделът.

Този модел е начин да се конфигурират маршрутите (ресурси, които се намират под една обща адресна линия) с помощта на файловата система. Вместо да се дефинира структурата на тези маршрути само код, те се определят като съвкупност от файлове и директории, които представляват йерархията на маршрутите на приложението.

Структурата му е:

1. app: главната директория за съхраняване на всички маршрути, които потребителя може да ги достъпи
2. \_<route>: име на маршрут, който започва с долна черта се счита за личен ресурс и не може да се достъпи от потребителя
3. (route) - този маршрут с обградени скоби е повече за групиране на маршрути под обща директория и не променя маршрута на адресната линия за потребителят
4. Page.jsx/tsx - този файл прави маршрута публично достъпен за потребителят
5. Layout.jsx/tsx - този файл служи за оформлението е потребителски интерфейс, който се споделя между множество маршрути. Това оформление запазва състоянието на маршрутите, за да останат интерактивни
6. Templates - шаблоните са подобни на оформлението (layout) по това, че споделя между множество маршрути, но разликата е че шаблоните не запазват състоянието на маршрутите (т.е. DOM елементите се пресъздават и ефектите се повторно синхронизират).

**Сървърно приложение**

Сървърната страна е междуплатформено, т.е. може да се внедри във всяка операционна система. Избраният език за писане е Java със Quarkus фрамелурка за разработване на REST API service, да може да обработва клиентските заявки съответно.

Софтуерната му архитектура е принципът на Model-View-Controller (MVC) моделът, където:

1. Model: представлява структурата от данни, която се организира в зависимост от релациите с други структури от данни (примерно модел потребител има релация с тип на потребителя, типът риск с почви, земетресения и т.н.).
2. View: интерфейс с който комуникира с крайният потребител
3. Controller: междинен компонент между Model и View, който взима клиентската заявка от View, обработва ги с помощта на Model и връща крайният резултат обратно към потребителя

В сървърното приложение се прилага принципът: *,,dependency injection’’* (DI), който е начин за описване на модули от кои други външни модули той зависи, за да може да обработи съответната задача за който е предназначен. Примерно при стандартна обектно-ориентирана система, начина на описване на зависимости между модули се описва когато първо се създават обекти в дефиницията на класа и след това се предава създадения обект към зависещия модул, докато с DI зависимите модули не се създават вътрешно в класа, а цялата тази отговорност се пренася на зависимия модул чрез използването на контейнер, или още наречен: *,,inversion of control’’* (IoC) .

Контейнерът е отговорен за управлението на жизненият цикъл на всички зависими модули и тяхната инициализация, според конфигурацията им. По отношение на реализация на проекта се използва Quarkus Framework на сървърният Java.

Има 3 различни имплементации за имплементиране на DI:

1. Constructor injection: зависимите модули се предават през конструктора

2. Setter injection: зависимите модули се предават през публичен метод за модификация на обект

3. Method injection: зависимите модули се предават през публични методи, които извършват и се нуждаят от външни зависими модули, за да могат да обработят услугата си.

## **3.2 Дефиниция на модулите на системата**

Модулите на системата са разделени между клиентското и сървърното приложение.

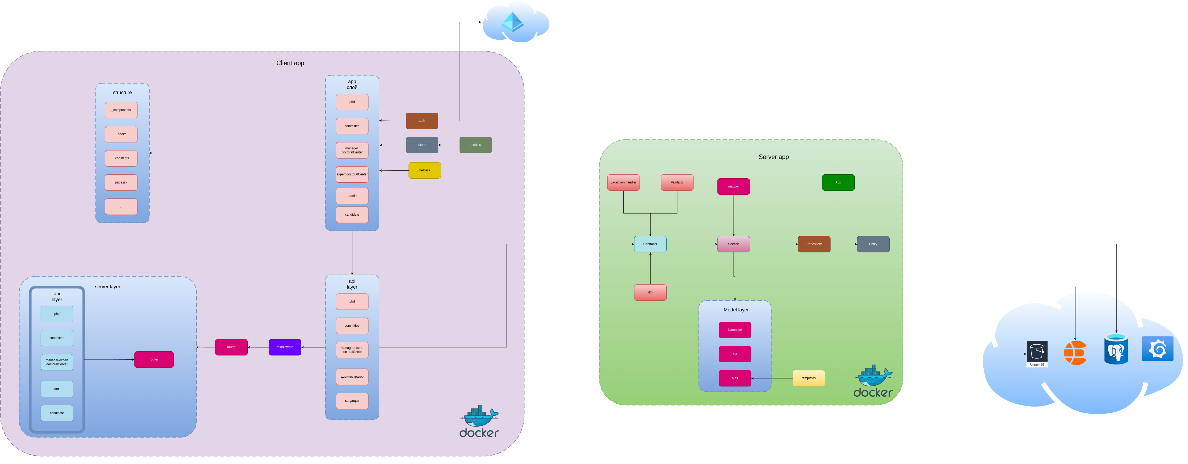
**Клиентско приложение**

* app: главната директория за съхраняване на всички маршрути, които потребителя може да ги достъпи
* Public - публична директория за съхраняване на медии
* Docs - директория за съхраняване на документи, свързано с техническата част на проекта
* Components – reactjs компоненти, които служат като базов интерфейс за приложението. Основната полза е оформление на компонентите (календар, бутон, т.н)
* Hooks - този компонент автоматично се извиква при настъпило се събитие (пример когато потребителят се опита да достъпи главната страница, да се извика функция за удостоверение и да го пренасочи в страницата за влизане)
* Lib - набор от функции за подпомагане на нормалното поведение на приложението
* Models - набор от класове за дефиниране на всички обекти за приложението ( докторант, изпит, член на докторантски център и др.)
* Middleware.js - файл за модифициране на клиентската http заявка преди да бъде обработена от сървърът. Служи основно за пренасочване на адресната линия на потребителя, в случай ако той се опитва да достъпи маршрут/ресурс, която той няма права да я достъпи

**Сървърно приложение**

* app: модул, който е стартираща точка на сървърът
* Controller: модул, който описва всичките API класове, чрез който се прави комуникация с клиента
* Entity: модул, който описва всички структури с данни, от който приложението работи над тях.
* Exception: модул за обработка на всички възможни грешки от страна на REST API.
* Mapper: вид помощен модул, който преобразува/филтрира една структура към друга
* Repository: модул, който обработва entity структурите от данни и ги предава на контролера
* Service: модул, който дефинира бизнес правилата на приложението и предава крайният резултат към контролера.
* Model: модул за дефиниране на микросервис, който служи за взаимодействие с други външни системи (съхраняване на медии в AWS S3 storage service, правене на зявки в PostreSQL изпращане на имейл съобщения)
* Templates – html документи, които служат да бъдат изпратени във формата на имейл съобщение към потребителят
* Dto (Data Object Transfer) - набор от класове, които служат за сериализация на заявките в Controller модула
* Validator - служи за ‘бърза’ валидация на данните веднага след сериализация на dto класовете в Controller модула

## **3.3 Проектиране на отделните модули**



Фигура 17 Архитектура на отделните модули

## **3.4 Прототип на потребителския интерфейс**

В случая на изискванията за UI/UX дизайн за прием и обучение на докторанти, приложението трябва да покрива създаването на интуитивна, ефективна и удобна среда за потребителите (докторанти, членове от комитета, ръководители и експерти на докторантски център и администратори), за да се навигират безпроблемно и да използват платформата ефективно.

По подробност можем да разделим изискванията на самостоятелни под-точки:

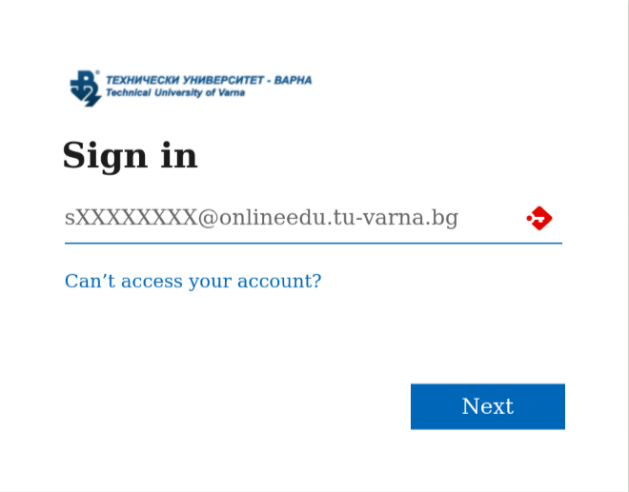
* + Простота и яснота:
    - интерфейсът трябва да е изчистен и без излишни елементи.
    - дизайнът трябва да се фокусира върху използваемостта, като всеки елемент служи за отделна цел.
    - подсказките и помощните секции трябва да са лесно достъпни за нови потребители и по-напреднали.

Отзивчивост и бързина:

* + - приложението трябва да е отзивчив и бърз, като гарантира, че потребителите могат бързо да навигират през различни секции, едновременно имайки обратна връзка (напр. анимация за зареждане)
    - желателно е дизайнът да се адаптира към различни устройства и размери на екрана
* Навигация и достъпност:
  + навигационните менюта и структури трябва да бъдат интуитивни и универсални в цялата платформа, за да улесни потребителите да се движат между различни страници
  + всички функции трябва да са лесно достъпни, в идеалния случай с 2-3 кликвания.
* Табло за управление и визуализация на данни:
  + визуално и информативно табло (*dashboard*) за изпити, неудостоверени и удостоверени потребители, събития, кандидати и т.н. е важно изискване, особено за приложение което служи за платформа за обучение на докторанти.
  + приложението трябва да предлага цялостен поглед върху статуса на изпита, на коя комисия е назначена, на каква оценка са гласували и (оптимално) ключовите показатели за ефективност (KPI), всичко това на едно място.
  + инструментите за визуализация на данни трябва да бъдат включени, за да представят данните в едно-страничен формат.
* Управление на неудостоверени и удостоверени потребители:
  + администраторите трябва да могат да назначават неудостоверени потребители роли, да ги премахват в системата и да пренасят отговорността на член на докторантски център да ги назначи като докторанти с леснота
  + могат да бъдат включени визуални елементи като кръгова диаграма или стълбовидна диаграма за по-добра визуализация на задачите.
* Филтриране на колони от таблици:
  + потребителите трябва да могат да персонализират изгледи, чрез филтри, базирано на колоните от съответната таблица, за да отговарят на техните специфични нужди.
* Функции за сътрудничество:
  + интерфейсът трябва да поддържа сътрудничество с функции като коментари, споделяне на файлове, маркиране и актуализации в реално време.
* Интегриране с други инструменти:
  + софтуерът трябва да бъде проектиран да се интегрира лесно с други често използвани бизнес инструменти.
  + софтуерът НЕ трябва да зависи от външен софтуер и трябва да бъде независим (модулиран)
* Стандартизация:
  + елементите на дизайна, като бутони, икони, цветови схеми и т.н трябва да имат еднакъв дизайн и да подобрява потребителското изживяване.
* Обратна връзка и обработка на грешки:
  + потребителският интерфейс трябва да предоставя ясна, конструктивна обратна връзка и да обработва грешките по подобаващ начин
  + потребителите трябва да бъдат уведомени за успешни действия и напътствани по подходящ начин, когато нещо се обърка.

Като се има в предвид изискванията на потребителския интерфейс, разработени са следните прототипи на дизайна .

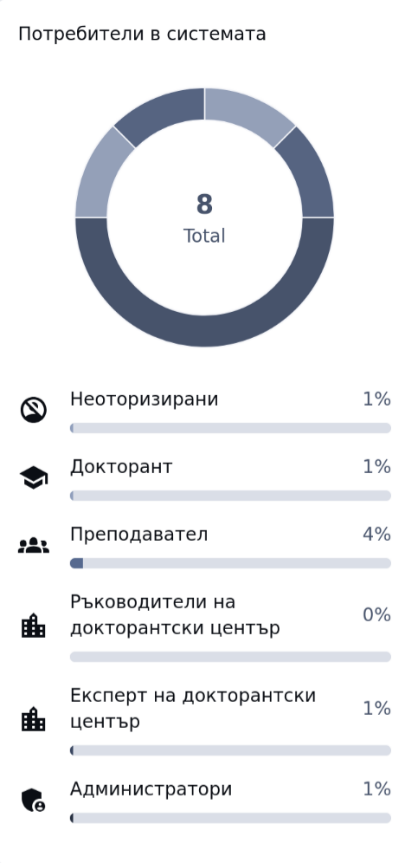
\*Прототиповете също се използват в продуктовата версия на приложението\*



Фигура 18 Страница за вход Microsoft Sign-in

Тази страница е предназначена за вход на потребител.

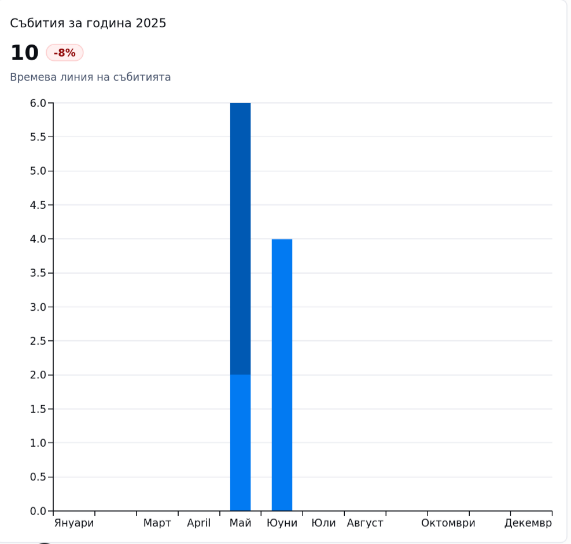
* потребителя трябва да си въведе имейл адресът за потвърждение
* потребителя трябва да си въведе парола (след екрана за парола)
* ако потребителят не може да си влезе в профила си поради някаква причина (или забравена парола), тогава той може да натисне линка ‘*Can’t access your account ?*’



Фигура 19 Кръгова диаграма на администратора

Този компонент е предназначен за визуализация на всички потребители за администратора в приложението във формата на кръгова диаграма.

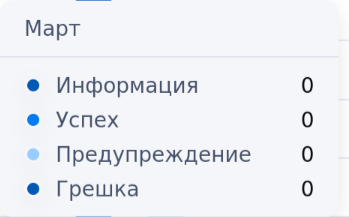
* визуализира се общият брой потребители в кръговата диаграма
* на всяка роля на потребител (неоторизирани, докторант, преподавател, ръководител/експерт на докторантски център, администратор) се показва бройката им
* Това табло може само да се достъпи чрез главната страница на администратора, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



Фигура 20 Стълбовидна диаграма на всички събития

Този компонент е предназначен за визуализация на всички събития, които са били генерирани от всички потребители в платформата (с изключение на администраторите).

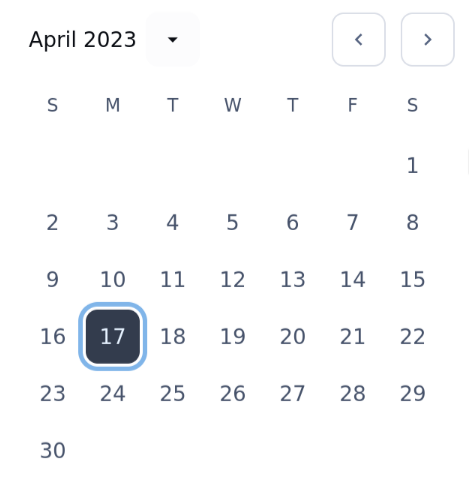
* Визуализират се всички събития, подредени за всеки месец в годината
* Администратора може да премине в следваща или предходна година, за да прегледа следващите събития през годината
* Показва се общият брой събития, генерирани през избраната година
* Това табло може само да се достъпи чрез главната страница на администратора, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



Фигура 21 Контекстно меню на стъпаловидната диаграма

Този компонент е предназначен да покаже допълнителна информация за всички събития, които са били генерирани от всички потребители в платформата (с изключение на администраторите).

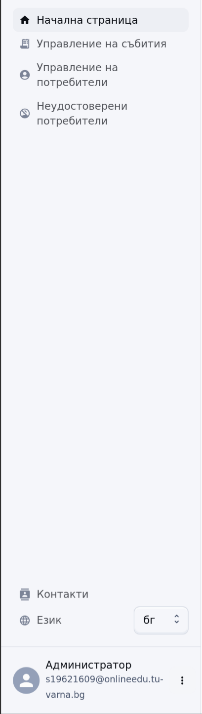
* При преминаване на курсора върху диаграмата се показва това контекстно меню за всички събития, подредени за всеки месец в годината
* Информацията за менюто се променя при преминаване на следваща или предходна година
* Показва се общият брой събития, генерирани през избраната година
* Това меню може само да се достъпи чрез главната страница на администратора, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



Фигура 22 Календар в приложението

Този компонент е предназначен да покаже всички събития, които ще бъдат известени на потребителя във формата на календар.

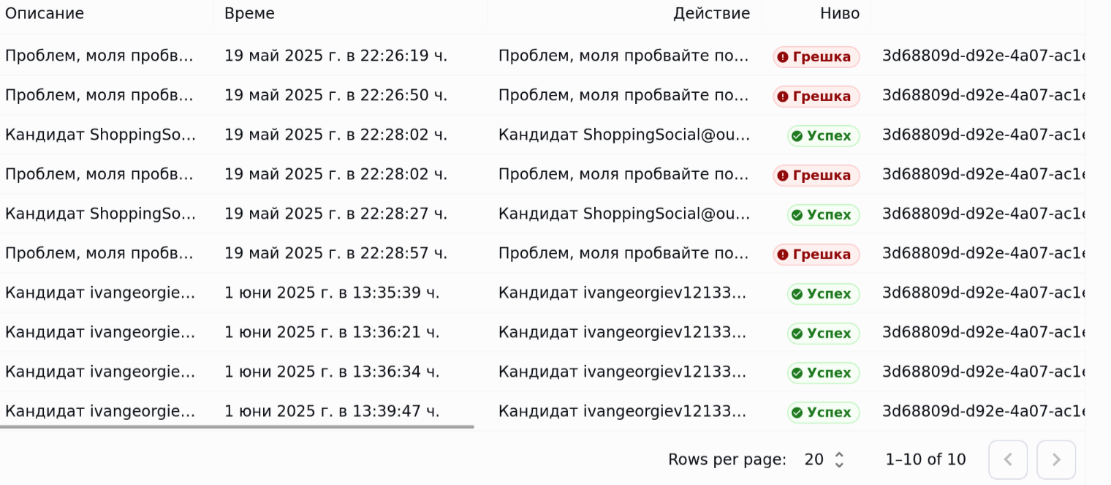
* При натискане на определен ден се показва информация за определеното събитие - титла, описание, степен на важност
* Потребителят може да преминава през следващите и предходните месеци/години чрез натискане на ‘<’, ‘>’ и върху текста на текущия месец (в този пример ‘April 2025’)
* За докторанта и членовете на комитета, календарът служи за дати на изпити



Фигура 23 Панел за навигация на опциите за потребителя

Toзи панел служи за навигация към различните страници и за смяна на езика.

* Навигационните бутони отгоре са различни според назначената роля на удостовереният си потребител. Те са следните
  + Докторант - начална страница, изпити, отчети, статус
  + Член от комитета - начална страница, изпити, комитети, отчети (за изминали изпити)
  + Член от докторантски център - начална страница, неудостоверени потребители, кандидатури, изпити
  + Администратор - начална страница, управление на събития, управление на потребители, неудостоверени потребители
  + Кандидат - начало, обявяване на конкурси, кандидатствай
* С натискане на долно-дясната част на панела се появява контекстно меню с опциите: профил изход
* В зависимост дали потребителят си е персонализирал профилът, в кръглата фигура ще се покаже неговата профилна снимка



Фигура 24 Таблица за всичките събития в таблица

Този компонент е предназначен да се покажат всичките събития във формата на таблица.

* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.
* Тази таблица може само от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



Фигура 25 Търсачка на филтриране на всичките събития

Този компонент е предназначен да се филтрират всичките събития, които са изведени под таблица.

* При въвеждане на тестовото поле с текста ‘Търси...’ впоследствие се филтрира таблицата на всяка колона, която има съвпадение с въведения текст
* При необходимост за по-детайлно и ограничено филтриране на събитията, потребителят може да ограничи търсенето си според колоната в таблицата, като натисне някой от бутоните под търсачката (текстовото поле)
* Този компонент може да бъде използвана само от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



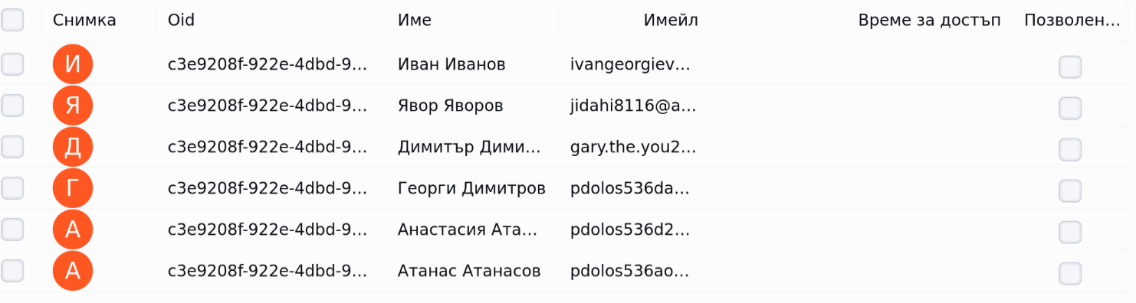
Този компонент е предназначен да упъти потребителят на коя страница той се намира в момента.



Фигура 26 Таблица за всички удостоверени потребители

Този компонент е предназначен да се покажат всичките удостоверени потребители във формата на таблица.

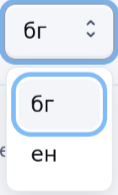
* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.
* При натискане на реда на удостоверен потребител в колоната на ‘Действия’, се появяват опциите ‘Премахни’, в което потребителят може да изтрие натиснатият от реда потребител от таблицата. За да може той отново да се удостовери, този потребител трябва първо да влезе в системата отново, да се покаже
* Тази таблица може само от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



Фигура 27 Таблица за всички неудостоверени потребители

Този компонент е предназначен да се покажат всичките неудостоверени потребители във формата на таблица.

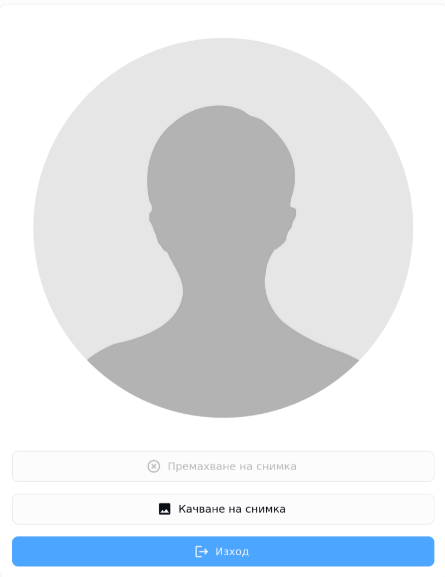
* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.
* При натискане на реда на удостоверен потребител в колоната на ‘Позволи’, се премества потребителят в друга таблица, която вече позволява член на докторантски център да назначи този потребител като докторант или член от комитета
* Тази таблица може само от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘администратор’



Фигура 28 Контекстно меню за промяна на езика

Този компонент е предназначен да сменя целят текст (динамичен и статичен) на потребителската страница.

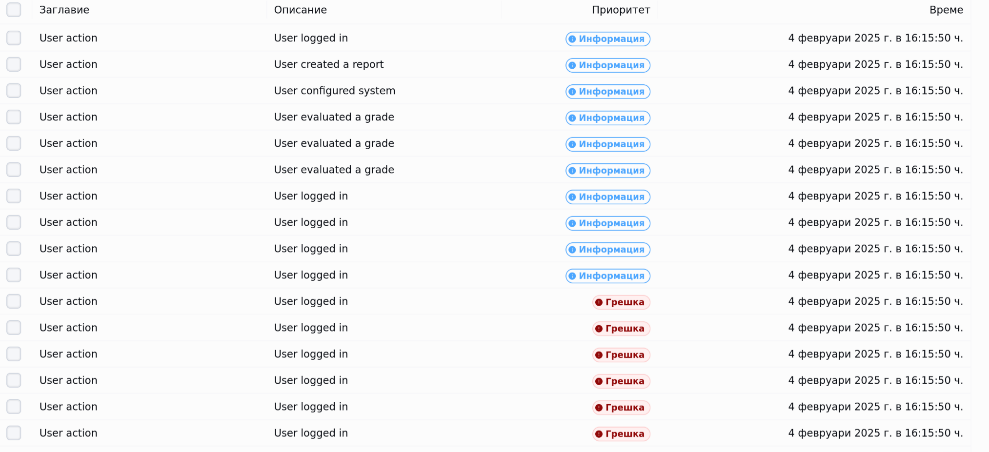
* Валидните опции за смяна на езика са: бг/bg (Български) и аг/en (Английски)
* Запазва се състоянието на избраният език между навигацията на различни страници



Фигура 29 Профилна страница на потребителя

Този компонент е предназначен да визуализира профилната снимка на удостовереният се потребител.

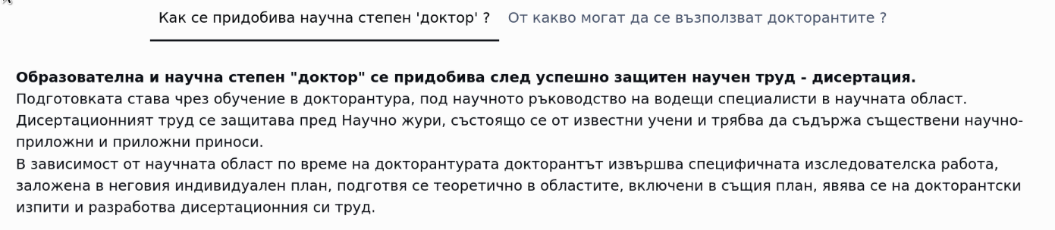
* Потребителят може да смени снимката си или да я премахне (в който случай снимката по подразбиране е тази илюстрация)
* При промяна на снимката, тя след това веднага се качва в S3 bucket и се подновява неговият запис в базата данни с новата снимка, за да може следващият път, когато потребителят влезе в системата, да се покаже вече качената снимка



Фигура 30 Таблица за всички нотификации (известия)

Този компонент е предназначен да се покажат всичките нотификации, предназначени на удостовереният потребител във формата на таблица.

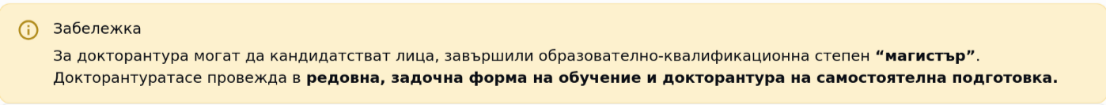
* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 нотификации се натиска долните навигационни бутони ‘<’ и ‘>’.



Фигура 31 Визуализация на основната информация за кандидатстване за докторант

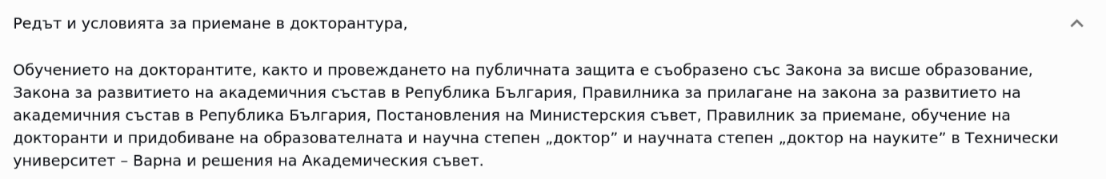
Този компонент е предназначен да покаже основната информация за кандидатстване за докторантура

* За най-често срещани въпроси може компонента да се смени според бройката им - при 2-3 въпроса компонента може да остане като ‘таб’, а при повече - като падащо меню



Фигура 32 Информационен компонент за кандидатстване за докторант

Този компонент е предназначен да покаже важна информация на кандидата и да му обърне вниманието.



Фигура 33 Ред и условия за кандидатстване за докторант

В случай че има нужда да се визуализира по-подробен абзац и този абзац да може да се групира съвместимо съдържанието му - визуализират се правилата за кандидатстване на докторантура.



Фигура 34 Таблица за учебните програми

Този компонент е предназначен да се покажат всичките публични учебни планова на кандидата във формата на таблица.

* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.
* При натискане на името на учебната програма, потребителят след това се навигира в съответната страница на натиснатата учебна програма



Фигура 35 Таблица за обявяване на конкурс

Този компонент е предназначен да се покажат всичките обявени конкурси на кандидата във формата на таблица.

* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.



Фигура 36 Компонент за показване на всичките стъпки на кандидата

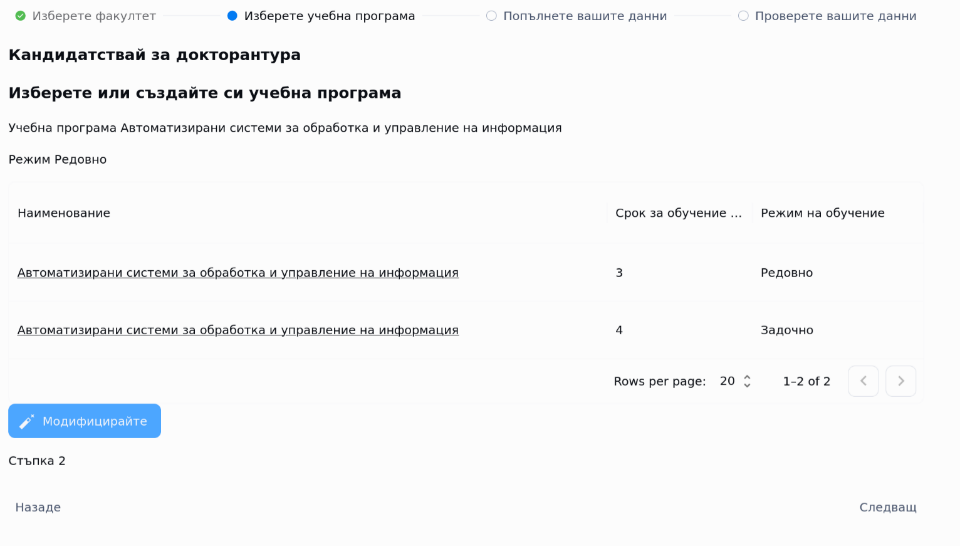
Този компонент е предназначен да се покажат всичките стъпки на кандидата за кандидатстване на учебната програма. Кандидатът може да се върне в предходната стъпка за коригиране на неговите данни.



Фигура 37 Таблица за избор на всички факултети

Този компонент е предназначен да се покажат всичките факултети, които кандидата може да избере за кандидатстване на докторантска програма във формата на таблица.

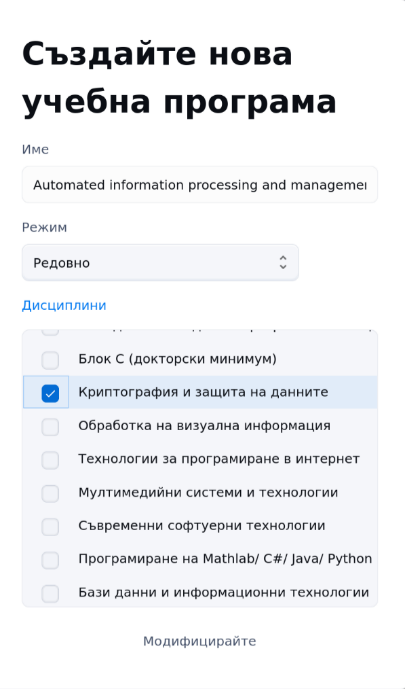
* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.
* При натискане на името на факултета ще се появи името на избрания факултет и потребителят след това ще може да премине в следващата стъпка за избор на учебна програма



Фигура 38 Таблица за избор или създаване на учебна програма

Този компонент е предназначен да се покажат всичките учебни програми, които кандидата може да избере за кандидатстване на докторантурска програма във формата на таблица.

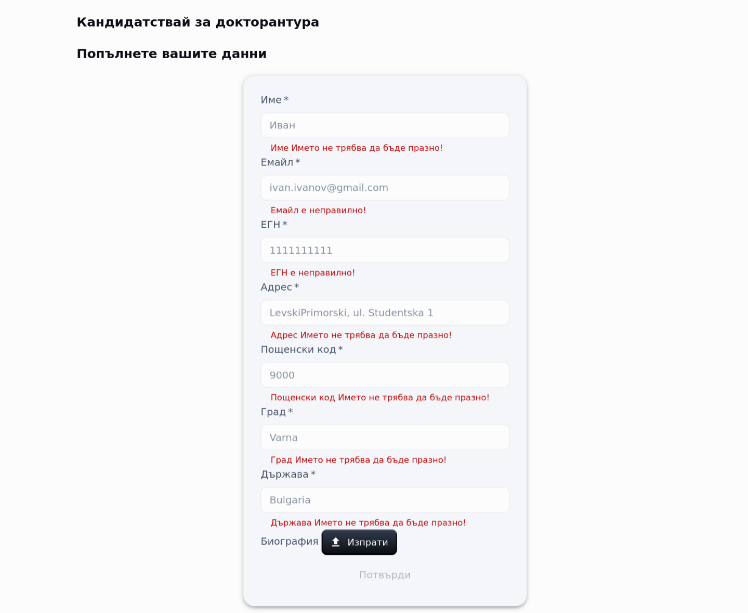
* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 събития се натиска долните навигационни бутони ‘<’ и ‘>’.
* Ако потребителят не желае да избере някои от пресъздадените учебни програми, тогава той може да си създаде своя собствена
* При натискане на името на учебната програма ще се появи името на избраната програма и потребителят след това ще може да премине в следващата стъпка за избор на неговите лични данни



Фигура 39 Формуляр за създаване на учебна програма

Този компонент е предназначен да се предостави на кандидата опцията да си създаде собствен учебен план

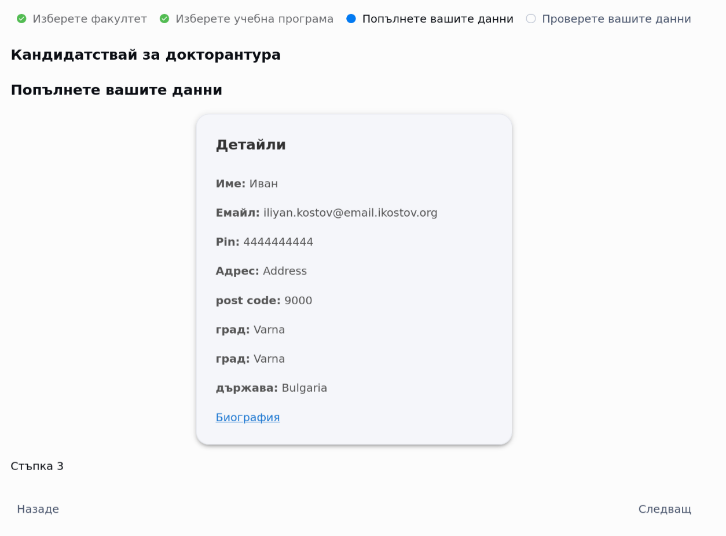
* Въвежда се името на учебният си план
* Избира се режим на обучение - редовно или задочно. От това ще зависи колко години кандидата ще се обучава
* В таблицата на дисциплини на потребителят са му избрани 3 дисциплини вече по подразбиране и той не може да ги премахне от учебната си програма, тъй като тези дисциплини са задължителни за провеждане. Те са - английски език, методи за изследване и разработване на дисертация
* Кандидатът може да си избере още 3 дисциплини, но те не са задължителни и не спира кандидата да продължи само с избраните дисциплини



Фигура 40 Формуляр за попълване на личните данни за кандидат

Този компонент е предназначен да се предостави на кандидата форма за въвеждане на личните си данни.

* Въвежда си пълното име
* Въвежда си личният имейл адрес за обратна връзка, свързано с кандидатурата му
* Въвежда си ЕГН по лична карта
* Въвежда си адресът на местоживеене по лична карта
* Въвежда си пощенския код
* Въвежда си града
* Въвежда си държавата
* Качва си автобиографията
* След въвеждане на данните си, кандидата трябва да натисне бутонът ‘Потвърди’



Фигура 41 Компонент за визуализация на попълнените лични данни за кандидат

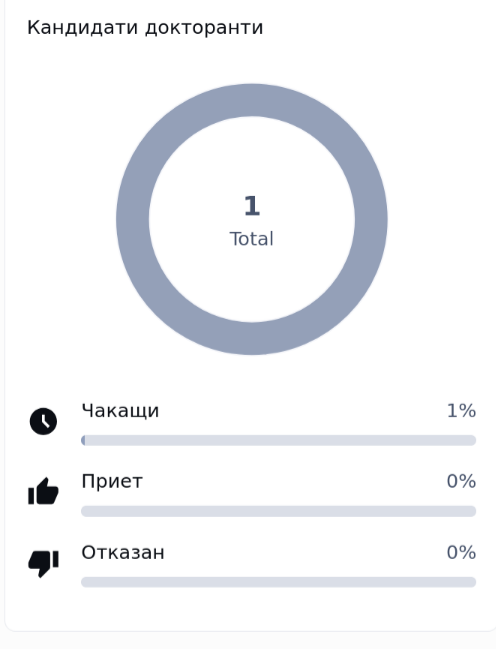
Този компонент е предназначен да покаже на кандидата въведените от него данни за допълнително потвърждение преди продължение

* Ако кандидатът би искал да си провери автобиографията, той може да натисне линка на текста ‘Биография’



Фигура 42 Станция са потвърждение за избраната кандидатура за докторант

Този компонент е предназначен да покаже на кандидата въведените от него данни (лични данни, учебна програма + дисциплини) за допълнително потвърждение преди кандидатстване.



Фигура 43 Кръгова диаграма за статуса на всички кандидати

Този компонент е предназначен да се покажат всичките кандидати с техният статус във формата на кръгова диаграма.

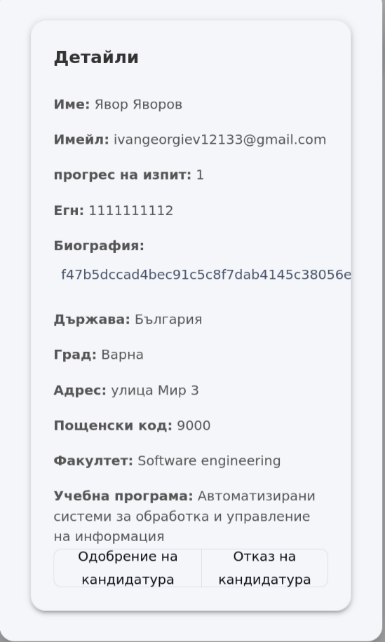
* визуализира се общият брой кандидати в кръговата диаграма
* на всеки вид статус на кандидата (чакащ, приет, отказан) се показва бройката им
* Тази таблица може само от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘член от комитета’ или ‘член от докторантски център’



Фигура 44 Таблица за всички кандиадти

Този компонент е предназначен да се покажат всичките кандидати във форма на таблица.

* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 нотификации се натиска долните навигационни бутони ‘<’ и ‘>’.
* При натискане на кандидата се показва прозорец с неговата информация и опция за одобряване или отказ на кандидатурата му
* Тази таблица може само от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘член от докторантски център’



Фигура 45 Компонент за визуализация на избраният кандидат

Този компонент е предназначен да се покажат всичките данни на избрания кандидат.

* При натискане на името на автобиографията се изтегля автоматично файла
* При натискане на бутонът ‘Одобрение на кандидатура’, записът на кандидата се превръща на ‘одобрен’ и той преминава в следващата стъпка на кандидатстване. Общо стъпките са 3. при стъпка 1-2 се генерират автоматично изпити на кандидата според неговата стъпка. При стъпка 3 (последната) кандидатът се изтрива от таблицата, създава се нов запис за докторант и се изпраща имейл съобщения на администраторите, че е необходимо да се създаде профил на новият докторант в Azure Active Directory. Също така се праща имейл съобщение на кандидата че е приет в системата.
* При натискане на бутон ‘Отказ’, статуса на кандидата се сменя на ‘отказан’



Фигура 46 Таблица за всички изпити

Този компонент е предназначен да се покажат всичките изпити (кандидатски и докторантски) във формата на таблица.

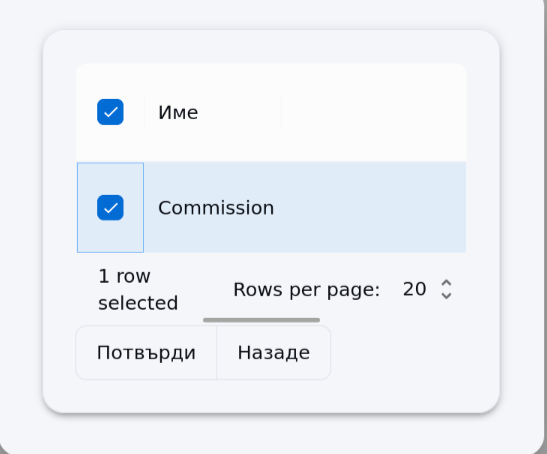
* Таблицата може да се сортира по азбучен ред на всяка колона, като се натисне върху нея
* За преминаване на следващите 20 нотификации се натиска долните навигационни бутони ‘<’ и ‘>’.
* При натискане на квадратното поле се появява прозорец с пълната информация на кандидата
* Тази таблица може само да се достъпи от потребители, където той трябва да влезе с предназначения си профил, на която е назначена ролята ‘член от докторантски център’



Фигура 47 Компонент за избраният изпит

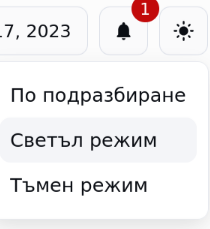
Този компонент е предназначен да се покажат всичките детайли за изпита.

* При натискане на бутон ‘задай комисия’ се извежда друг прозорец за задаването на всички комисии.



Фигура 48 Таблица за всички комисии

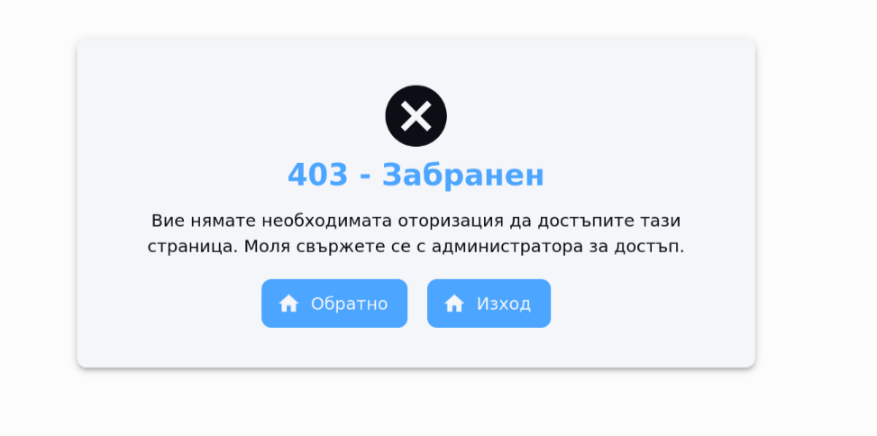
Този компонент е предназначен да се предостави на на член на докторантския център опцията да зададе комисия на избраният вече изпит.



Фигура 49 Контекстно меню за светъл/тъмен режим в приложението

Този компонент е предназначен да променя темата на уеб приложението за потребителя.

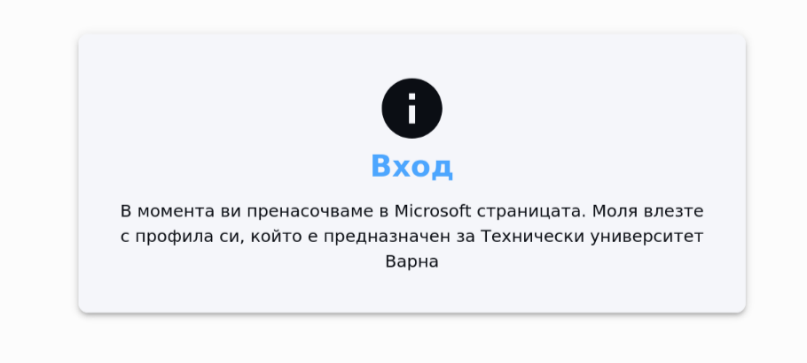
* състоянието на избраната тема се запазва при навигация към други страници



Фигура 50 Страница за известяване на потребителя че му е забранено да достъпи страница

Този компонент е предназначен да информира потребителя, че се опитва да достъпи страница, на която той няма достатъчно права в назначената си роля да я достъпи.

* адресните линии са структурирани по такъв начин, където всяка страница, която тясно зависи от ролята и се намира след домейна на ролята
* потребителят може да се върне обратно в началната страница като натисне бутона Обратно или да излезе от профила си



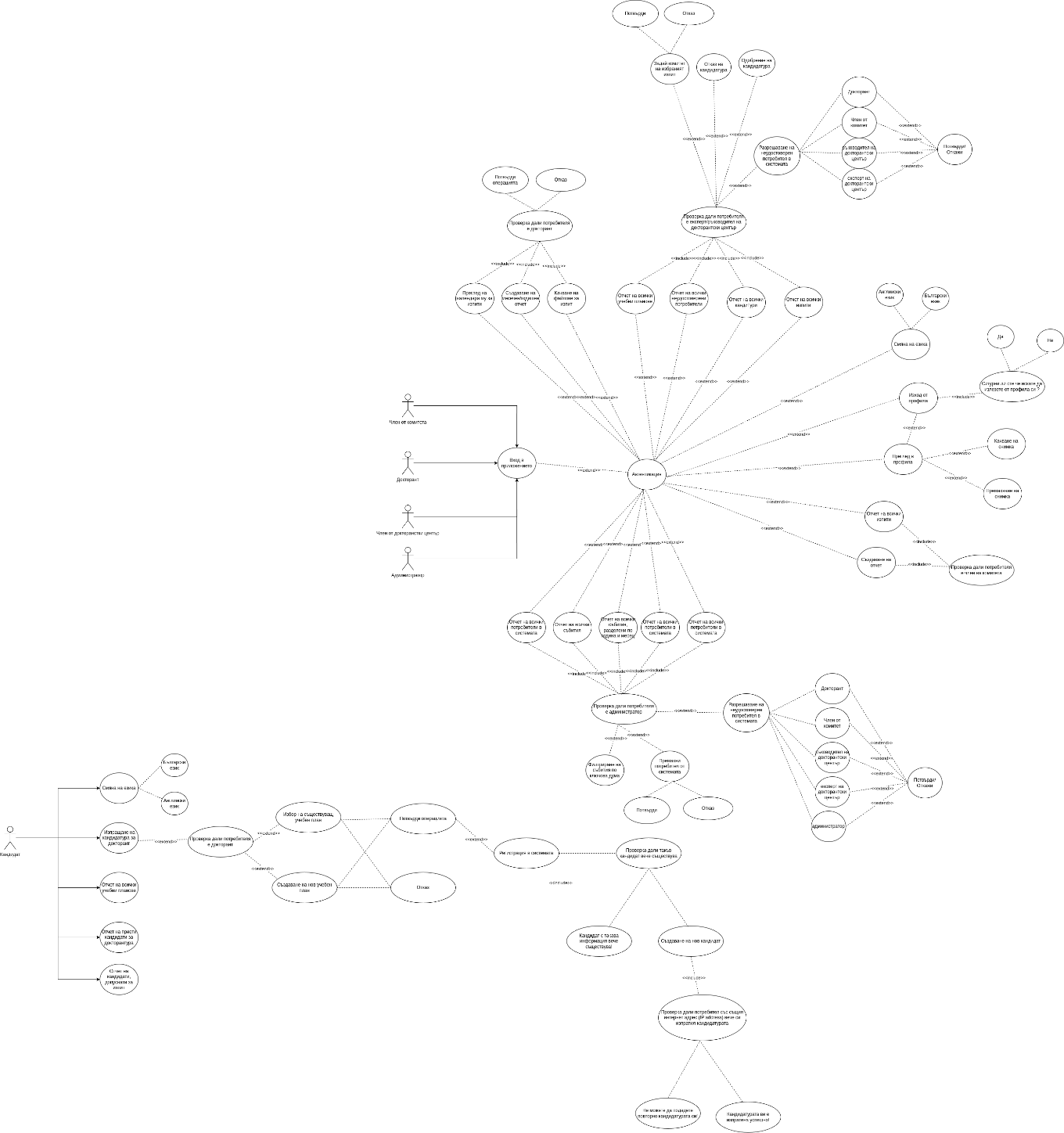
Фигура 51 Страница за известяване на потребителя че се пренасочва към друга страница за вход

Този компонент е предназначен да информира потребителя, че не може да уеб приложението, докато не се удостовери чрез Microsoft Sign-in страницата.

* този компонент се появява само когато потребителят не се е удостоверил в системата

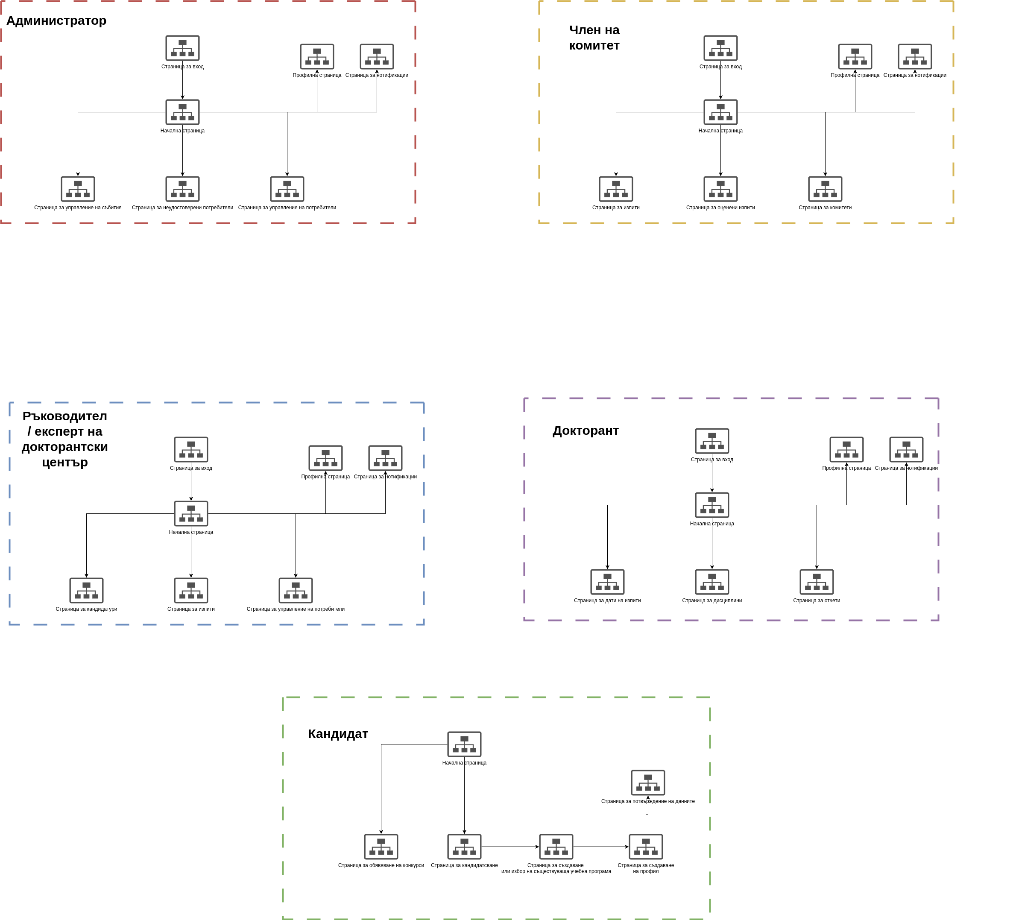
## **3.5 Архитектура на високо ниво**

### **3.5.1 UML диаграма**



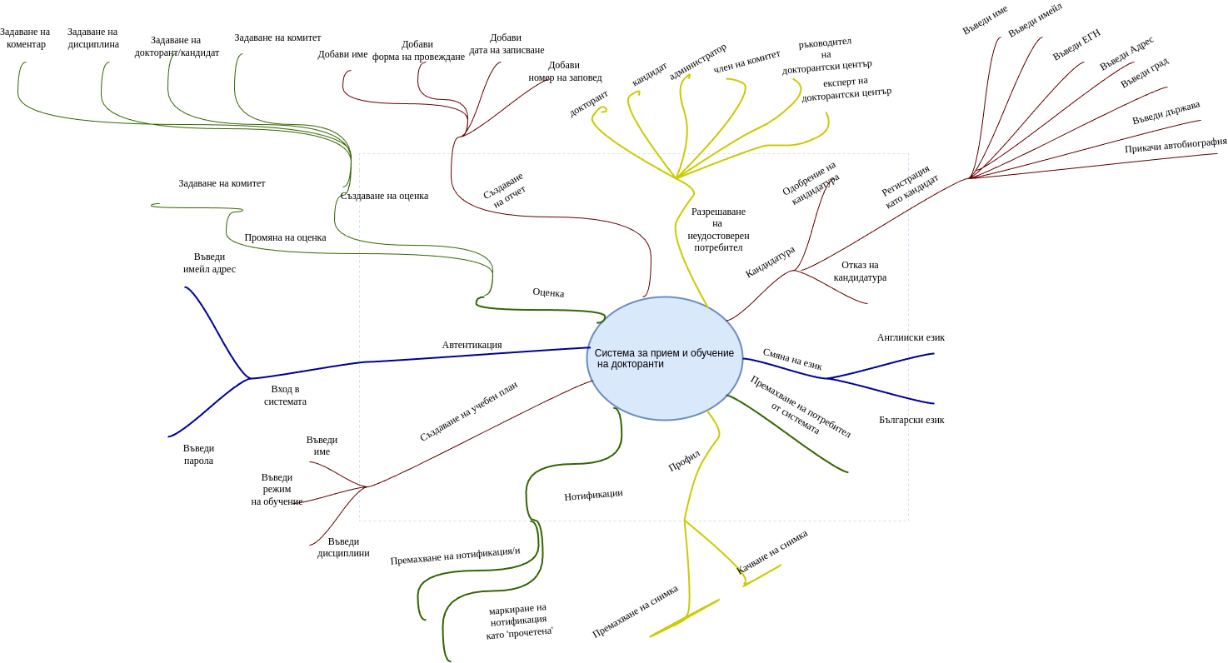
Фигура 52 UML диаграма

### **3.5.2 Site-map диаграма**



Фигура 53 Site-map диаграма

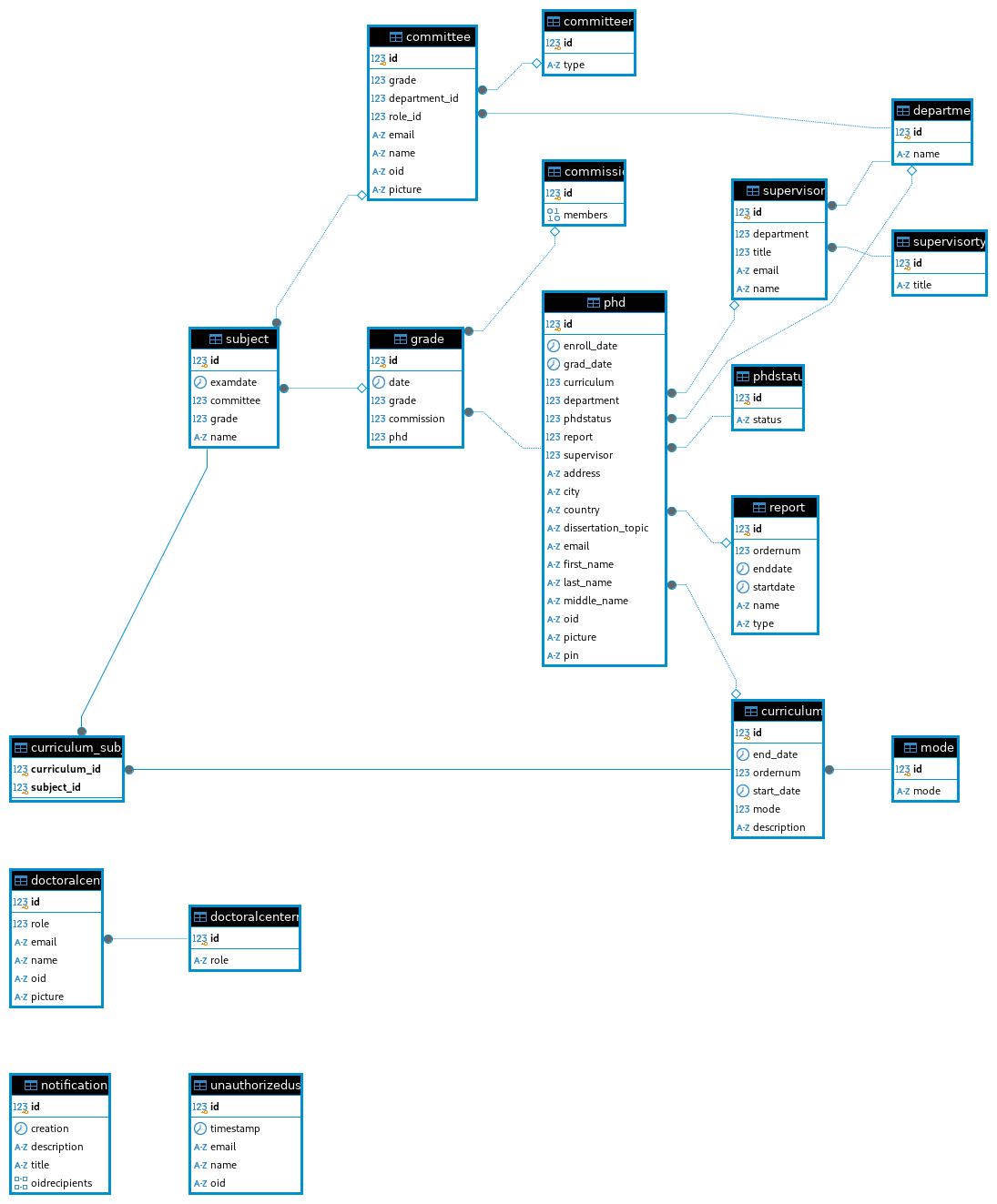
### **3.5.3 Mind-map диаграма**



Фигура 54 Mind-map диаграма

### **3.5.4 Концептуален модел на базата от данни**

Таблиците на приложението са разпределени, според изискванията на заданието.



Фигура 55 Концептуален модел на базата данни

Таблиците са следните:

1. Докторант (Phd)

* id – идентификатор на докторанта, тип цяло число
* name – пълното име на докторанта, тип низ
* email – имейл адрес на докторанта, тип низ
* oid – уникален идентификатор на докторанта в Azure Active Directory (Entra ID), тип низ
* pin – ЕГН на докторанта, тип низ
* addresss – постоянен адрес на докторанта , тип низ
* city – град на докторанта, тип низ
* country – държава на докторанта, тип низ
* dissertation – тема на дисертацията на докторанта, тип низ
* picture – уникално име на файла за личната снимка на докторанта, тип низ
* post\_code – пощенски код на докторанта, тип низ
* enroll\_date – начална дата на започване на обучението на докторанта, тип дата
* graduation\_date – дата на завършване на докторантурската програма, тип дата
* curriculum - идентификатор на таблица ‚ “Учебна програма“ с релация едно-към-много (1:M) , тип цяло число
* faculty - идентификатор на таблица ‚ “Факултет“ с релация едно-към-много (1:M) , тип цяло число
* status - идентификатор на таблица ‚ “PhdStatus“ с релация едно-към-много (1:M) , тип цяло число
* supervisor - идентификатор на таблица ‚ “Supervisor“ с релация едно-към-много (1:M) , тип цяло число
* report - идентификатор на таблица ‚ “Report“ с релация едно-към-много (1:M) , тип цяло число

1. Статус на докторанта (PhdStatus)

* id – идентификатор на статус на докторанта, тип цяло число
* status – идентификатор на статус на докторанта, тип цяло число, тип низ. Валидни стойности:
* записан (enrolled)
* завършен (graduated)
* терминиран (terminated)

1. Ръководител (Supervisor)

* id – идентификатор на ръководителя, тип цяло число
* name – пълното име на ръководителя, тип низ
* email – имейл адрес на ръководителя, тип низ
* picture – уникално име на файла за личната снимка на ръководителя, тип низ
* oid – уникален идентификатор на ръководителя в Azure Active Directory (Entra ID), тип низ
* title - идентификатор на таблица ‚ “SupervisorTitle“ с релация едно-към-много (1:M) , тип цяло число
* faculty - идентификатор на таблица ‚ “Faculty“ с релация едно-към-много (1:M) , тип цяло число

1. Титла на ръководителя (SupervisorTitle)

* id – идентификатор на титлата на ръководителя, тип цяло число
* title – име на титлата на ръководителя, тип цяло число, тип низ. Валидни стойности:
  + доцент (professor)
  + асистент (assistant)

1. Факултет (Faculty)

* id – идентификатор на факултета, тип цяло число
* faculty – име на факултета, тип низ.

1. Отчет (Report)

* id – идентификатор на отчета, тип цяло число
* name – описание на отчета, тип низ
* enrollment\_date – начална дата на отчета, тип низ.
* order\_number – дата на зачисляване на отчета, тип низ
* conduct – форма на провеждане на отчета, тип низ

1. Много докторанти към много отчети (M:M) (PhdReports)

* phd\_id – идентификатор на докторанта, тип цяло число
* report\_id – идентификатор на отчета, тип цяло число

1. Много докторанти към много оценки (M:M) (PhdGrades)

* phd\_id – идентификатор на докторанта, тип цяло число
* grade\_id – идентификатор на оценката, тип цяло число

1. Учебна програма (Curriculum)

* id – идентификатор на учебната програма, тип цяло число
* name – име на учебната програма, тип низ
* is\_public – указател дали учебната програма може да бъде избрана от други докторанти, тип булева стойност.
* faculty - идентификатор на таблица ‚ “Faculty“ с релация едно-към-много (1:M) , тип цяло число
* mode - идентификатор на таблица ‚ “Mode“ с релация едно-към-много (1:M) , тип цяло число

1. Режим на обучение (Mode)

* id – идентификатор на режима на обучение, тип цяло число
* year\_period – период години за провеждане на режима на обучение, тип цяло число.
* mode – име режима на обучение, тип низ. Валидни стойности:
  + regular (редовно)
  + part\_time (задочно)

1. Много учебни програми към много дисциплини (M:M) (CurriculumSubjects)

* curriculum\_id – идентификатор на учебната програма, тип цяло число
* subject\_id – идентификатор на дисциплината, тип цяло число

1. Дисциплина (Subject)

* id – идентификатор на дисциплината, тип цяло число
* name – име на дисциплината, тип низ
* exam\_date – дата за изпита на дисциплината, тип дата.
* teacher - идентификатор на таблица ‚ “Teacher“ с релация едно-към-много (1:M) , тип цяло число

1. Оценка (Grade)

* id – идентификатор на оценката, тип цяло число
* grade – число на общата оценка, тип двойно число
* eval\_date – дата за изпита за оценката, тип дата
* eval\_date – кратко описание за оценката, тип низ
* attachments – прикачени файлова за оценката, тип низ
* commission - идентификатор на таблица ‚ “Commission“ с релация едно-към-много (1:M) , тип цяло число
* subject - идентификатор на таблица ‚ “Subject“ с релация едно-към-много (1:M) , тип цяло число

1. Член от комитета (Committee)

* id – идентификатор на член от комитета, тип цяло число
* name – пълното име на член от комитета, тип низ
* email – имейл адрес на член от комитета, тип низ
* picture – уникално име на файла за личната снимка на член от комитета, тип низ
* grade – число на оценка на член от комитета, тип двойно число
* oid – уникален идентификатор на член от комитета в Azure Active Directory (Entra ID), тип низ
* faculty - идентификатор на таблица ‚ “Faculty“ с релация едно-към-много (1:M) , тип цяло число
* role - идентификатор на таблица ‚ “CommitteeRole“ с релация едно-към-много (1:M) , тип цяло число

1. Комисия (Commission)

* id – идентификатор на комисията, тип цяло число
* name – име на комисията, тип низ

1. Роля на член на комитета (CommitteeRole)

* id – идентификатор на роля на член на комитета, тип цяло число
* role – име на комисията, тип низ. Валидни стойности:
  + прецедател (chairman)
  + член (member)

1. Много комисии към много членове на комисии (M:M) (CommissionCommittees)

* commission\_id – идентификатор на комисията, тип цяло число
* committee\_id – идентификатор на член на комитета, тип цяло число

1. Кандидат (Candidate)

* id – идентификатор на кандидата, тип цяло число
* name – пълното име на кандидата, тип низ
* email – имейл адрес на кандидата, тип низ
* pin – ЕГН на кандидата, тип низ
* addresss – постоянен адрес на кандидата , тип низ
* biography - автобиография на кандидата, тип низ
* city – град на кандидата, тип низ
* country – държава на кандидата, тип низ
* post\_code – пощенски код на кандидата, тип низ
* exam\_step – текущ прогрес на изпит на кандидата, тип низ
* year\_accepted – година на приемане за докторантурска програма4 на кандидата, тип цяло число
* curriculum - идентификатор на таблица ‚ “Curriculum“ с релация едно-към-много (1:M) , тип цяло число
* faculty - идентификатор на таблица ‚ “Faculty“ с релация едно-към-много (1:M) , тип цяло число
* status - идентификатор на таблица ‚ “CandidateStatus“ с релация едно-към-много (1:M) , тип цяло число

1. Статус на кандидата (CandidateStatus)

* id – идентификатор на статус на кандидата, тип цяло число
* status – име на статус на кандидата, тип низ. Валидни стойности:
  + в чакане (waiting)
  + приет (approved)
  + отказан (rejected)
  + в преглед (reviewing)

1. Много кандидати към много оценки (M:M) (CandidateGrades)

* candidate\_id – идентификатор на кандидата, тип цяло число
* grade\_id – идентификатор на оценката, тип цяло число

1. Член на докторантски център (DoctoralCenter)

* id – идентификатор на член на докторантски център, тип цяло число
* name – пълното име на член на докторантски център, тип низ
* email – имейл адрес на член на докторантски център, тип низ
* oid – уникален идентификатор на член на докторантски център в Azure Active Directory (Entra ID), тип низ
* picture – уникално име на файла за личната снимка на член на докторантски център, тип низ
* role - идентификатор на таблица ‚ “DoctoralCenterRole“ с релация едно-към-много (1:M) , тип цяло число

1. Роля на докторантски център (DoctoralCenterRole)

* id – идентификатор на роля на докторантски център, тип цяло число
* status – име на роля на докторантски център, тип низ. Валидни стойности:
  + администратор (admin)
  + експерт (expert)
  + ръководител (manager)

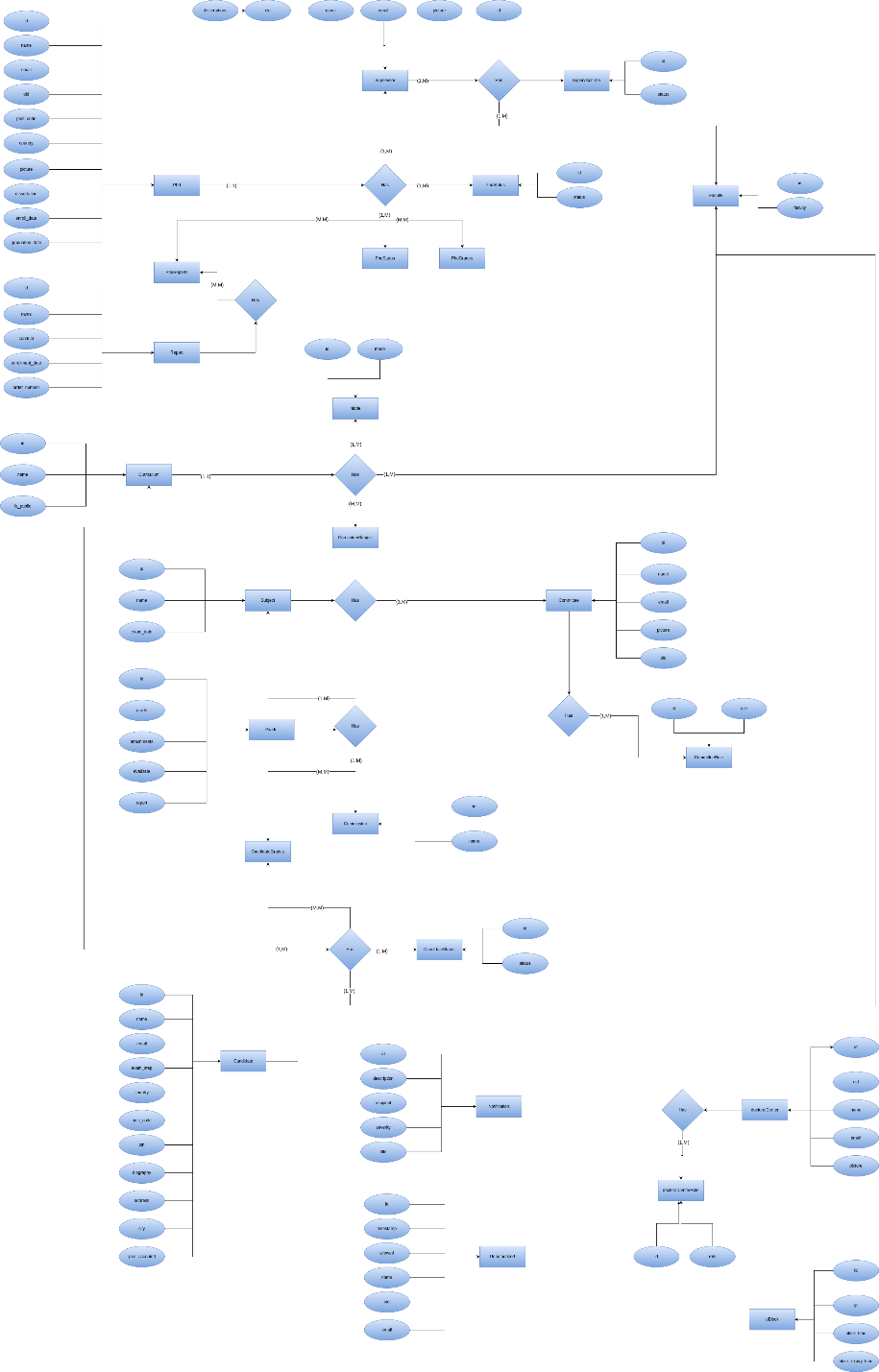
1. Неоторизиран (Unauthorized)

* id – идентификатор на неоторизираният потребител, тип цяло число
* name – пълното име на неоторизираният потребител, тип низ
* email – имейл адрес на неоторизираният потребител, тип низ
* oid – уникален идентификатор на неоторизираният потребител в Azure Active Directory (Entra ID), тип низ
* allowed – дали този потребител е позволен да бъде назначен от член от докторантски център, тип булева стойност
* timestamp – дата на забравянане на неоторизираният потребител, тип дата

1. Нотификация (Notification)

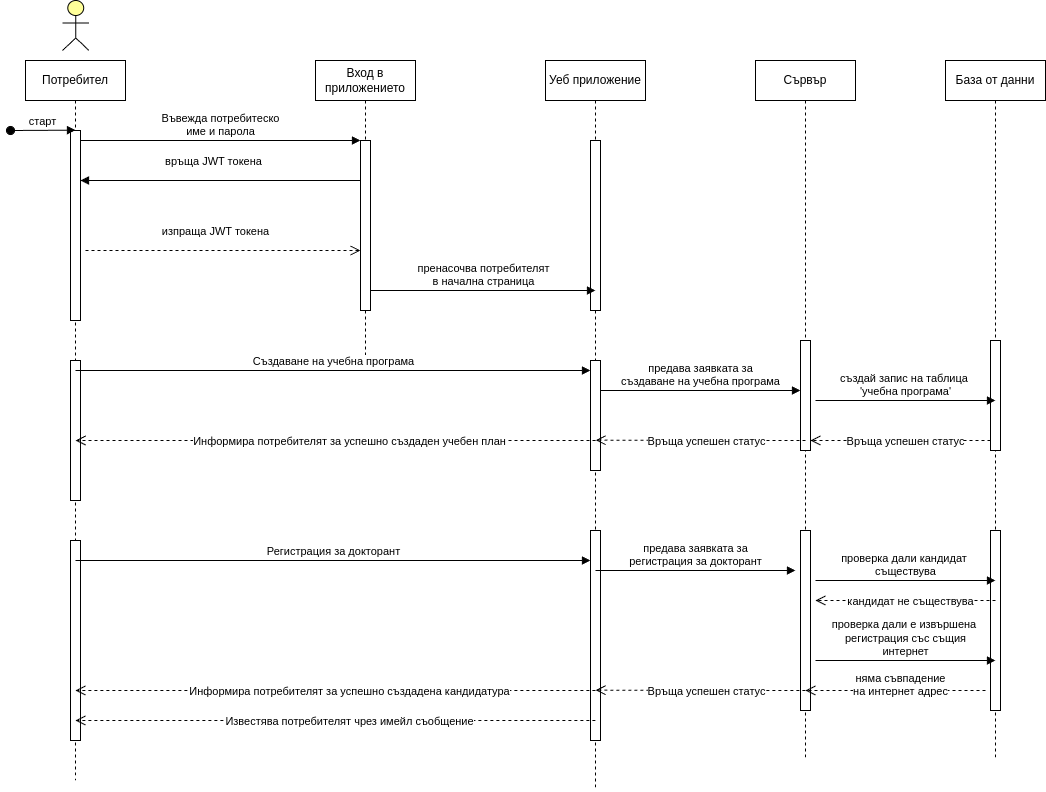
* id – идентификатор на нотификацията, тип цяло число
* title – титла на нотификацията, тип низ
* description – описание на нотификацията, тип низ
* recipient – получател на нотификацията, тип низ
* severity – важност на нотификацията, тип низ. Валидни стойности:
* информация (info)
* успех (success)
* грешка (error)
* предупреждение (warn)
* creation – дата на създаване на нотификацията, тип дата
* Блокиране на интернет адрес (IPBlock)
* id – идентификатор на блокиране на интернет адрес, тип цяло число
* block\_expiry\_time – дата на изтичане на блокиране на интернет адрес, тип дата
* block\_time – дата на блокиране на интернет адрес, тип дата
* ip – стойност на интернет адрес, тип интернет адрес

### **3.5.5 Модел на Чен диаграма**



Фигура 56 Модел на Чен диаграма

### **3.5.6 Sequence диаграма**



Фигура 57 Sequence диаграма

# **4.0 Реализация на системата**

## **4.1 Първоначална настройка на програмната локална и облачна среда**

### **4.1.1 Сървърно приложение**

#### 4.1.1.1 Windows

За изпълнение на сървърното приложение, първо трябва да имате инсталиран jdk 21 (този инструктаж използва WSL2).

* Изтегляне на JDK инсталационната програма чрез URL :

wget <https://download.oracle.com/java/21/archive/jdk-21.0.6_windows-x64_bin.exe>

* инсталиране на JDK в локалната машина чрез :

start [jdk-21.0.6\_windows-x64\_bin.exe](https://download.oracle.com/java/21/archive/jdk-21.0.6_windows-x64_bin.exe)

* Тъй като приложението силно зависи от Java фреимлурка Quarkus, ще бъде необходимо също да се инсталира.
* изтегляне на инсталационната програма чрез URL :

wget <https://nodejs.org/dist/v16.10.0/node-v16.10.0-x64.msi>

curl -Ls https://sh.jbang.dev | bash -s - trust add <https://repo1.maven.org/maven2/io/quarkus/quarkus-cli/>

curl -Ls https://sh.jbang.dev | bash -s - app install --fresh --force quarkus@quarkusio

* за да може да съхраняваме потребителските данни, заедно с зависимите данни за взаимодеиствие с приложението, ще бъде необходимо да инсталираме PostgreSQL 17
* изтегляне на инсталационната програма чрез URL :

curl <https://www.enterprisedb.com/postgresql-tutorial-resources-training-1?uuid=69f95902-b451-4735-b7e4-1b62209d4dfd&campaignId=postgres_rc_17>

#### 4.1.1.2 Linux

За най-ефективната инсталация на всички зависимости, е силно препоръчително да се използва Nix.

Преди всичко уверете се че имате Nix инсталиран в системата ви с командата:

nix --version

За инсталация, уверете се че имате инсталиран direnv

programs.direnv = {

enable = true;

silent = true;

nix-direnv.enable = true;

};

* след това активирайте direnv с командата:

direnv allow

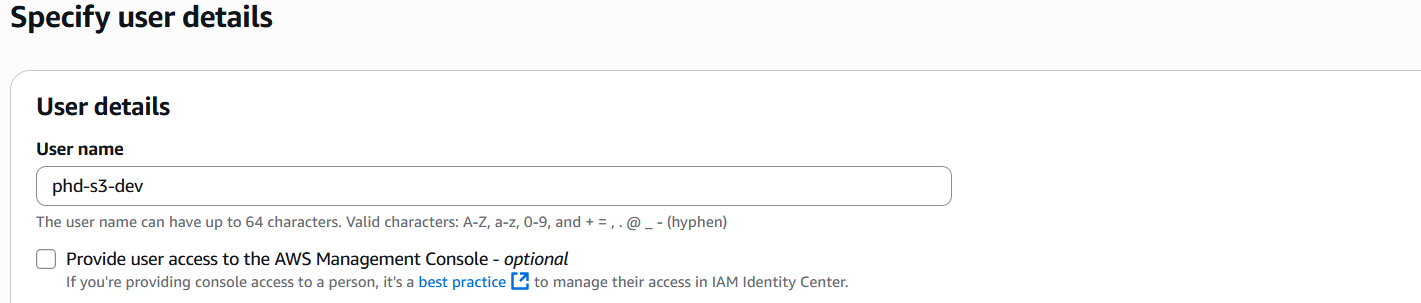
* Създаване на нов AWS акаунт, за да можем да се възползваме от ресурсите, които ще ги използваме в нашето приложение
* следвайте стъпките за създаване на акаунт в : <https://docs.aws.amazon.com/SetUp/latest/UserGuide/setup-AWSsignup.html>
* За да можем да взаимодействаме с AWS S3,ще ни трябва да инсталираме AWS CLI версия 2 (команден интерпретатор) чрез командата:

msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi

* Конфигурация на AWS CLI
* за настройка на AWS CLI,напишете командата :

aws configure

* в терминала ще ви пита да си добавите тайният ключ и идентификация номер в AWS конзолата. С цел да не използвате удостоверенията си от главния акаунт (root) е силно препоръчително да се създаде индивидуален профил за взаимодействие с AWS CLI
* Придвижете се до страницата за създаване на IAM потребител [https://console.aws.amazon.com/iamv2/home#/users/create](https://console.aws.amazon.com/iamv2/home" \l "/users/create)
* Въведете потребителско име и изберете Напред. Можете да именувате потребителя произволно, например „phd-s3-dev“.



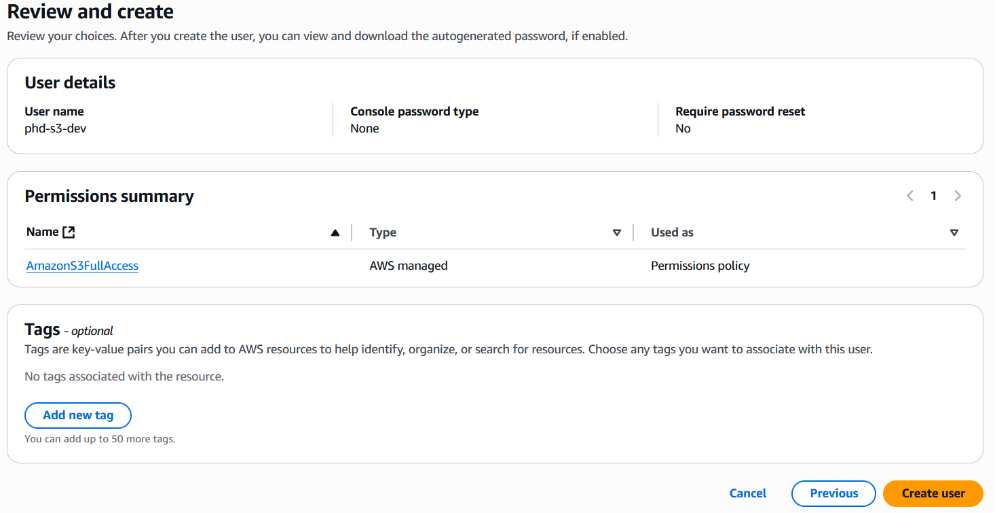
Фигура 58 Пример за създаване на нов IAM потребител

Изберете Директно прикачване на политики и изберете AmazonS3FullAccess като политика за разрешения.



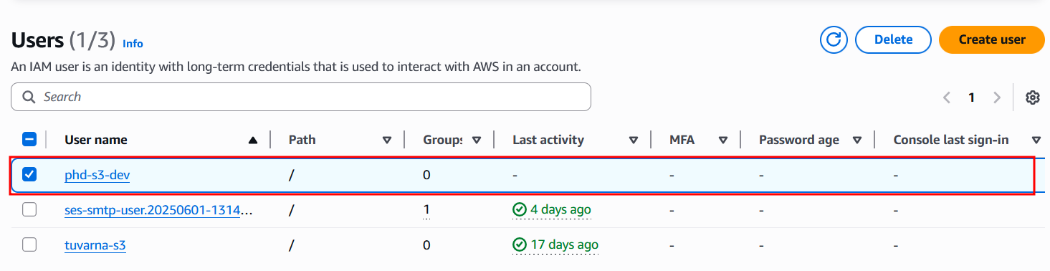
Фигура 59 Пример за добавяне на политика към IAM потребител

* На страницата за преглед проверете дали всичко изглежда добре и изберете Създаване на потребител.



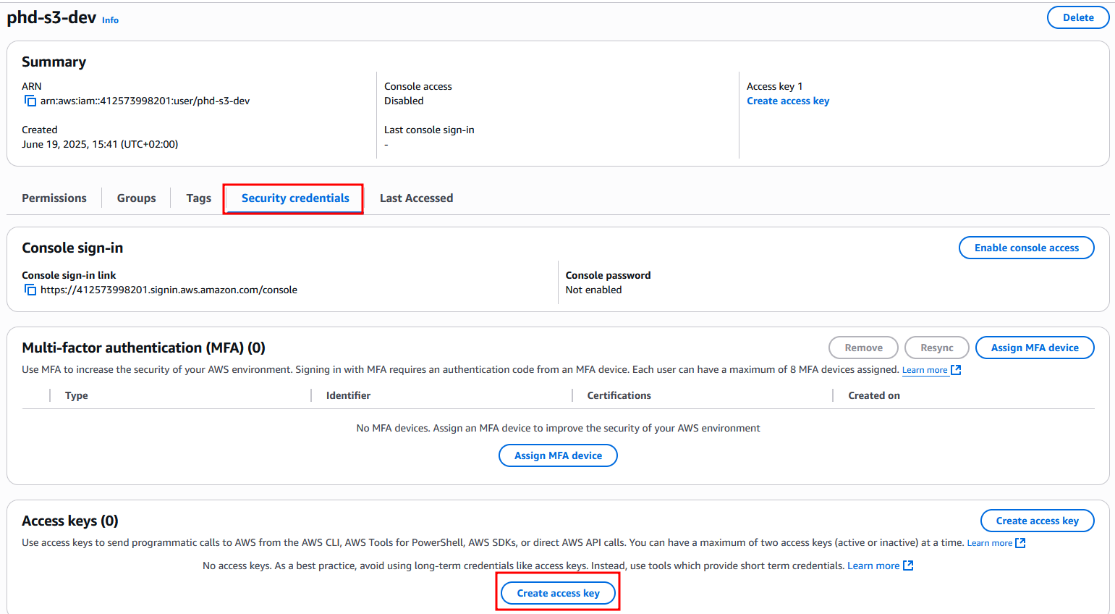
Фигура 60 Пример за потвърждение на новия IAM потребител

* Това ще ви пренасочи към страницата със списък с потребители. Изберете потребителя, който току-що сте създали.



Фигура 61 Пример за изглед на новия IAM потребител в конзолата

* На страницата с подробности за потребителя отидете до раздела Идентификационни данни за защита, превъртете надолу до Ключове за достъп и изберете Създаване на ключове за достъп.



Фигура 62 Пример за навигация на тайните ключове за IAM потребител

* Изберете в опцията за Command-Line Interface (CLI)

A screenshot of a computer

Description automatically generated with medium confidence, Picture

Фигура 63 Пример за създаване на тайни ключове, използвани само програмно

След това копирайте тайните ключове, които ще ги използвате за автентикация към AWS .

A screenshot of a computer

Description automatically generated with low confidence, Picture

Фигура 64 Пример за копиране на тайните ключове

Въведете стойностите, които току-що сте копирали, в съответният CLI.

* Създаване на ново Quarkus приложение
* Изпълнете следните команди за създаване на нов проект:

quarkus create && cd code-with-quarkus

Това създава ново приложение Java в директория, наречена code-with-quarkus и след това отивате в новата директория.

* От директорията code-with-quarkus стартирайте приложението, като използвате следната команда:

quarkus dev

* Това ще ви стартира сървъра локално, и можете да го достъпите на http://localhost:8080

### **4.1.2 Клиентско приложение**

#### 4.1.2.1 Windows

За изпълнение на клиентското приложение, трябва да имате инсталиран pnpm 9.14.4.

* Изтегляне на pnpm чрез URL:

Invoke-WebRequest https://get.pnpm.io/install.ps1 -UseBasicParsing | Invoke-Expression

* Тъй като приложението е базирано на NextJS фреимлурка, ще трябва да инсталираме пакета като използваме npx.

npx create-next-app@latest

След изпълнението на тази команда, командния интерпретатор ще ви попита списък с въпроси, свързано със каква средата да ви създаде - само потвърдете за избора на Typescript да потвърдите с ‘No’.

What is your project named? my-app

Would you like to use TypeScript? No / Yes

Would you like to use ESLint? No / Yes

Would you like to use Tailwind CSS? No / Yes

Would you like your code inside a `src/` directory? No / Yes

Would you like to use App Router? (recommended) No / Yes

Would you like to use Turbopack for `next dev`? No / Yes

Would you like to customize the import alias (`@/\*` by default)? No / Yes

What import alias would you like configured? @/\*

* От директорията my-app стартирайте приложението, като използвате следната команда:

cd my-app && npm run dev

* Това ще ви стартира сървъра локално, и можете да го достъпите на http://localhost:3000

#### 4.1.2.2 Linux

Отново както и при Java инструкциите за Nix, ще ви необходимо същият атрибут и накрая да изпълните командата:

direnv allow

## **4.2 Функционалности на приложението**

Основните функции на проекта са предназначени на 5 различни типа потребители: докторанти, кандидати, администратори, член на докторантски център и член на комитет. Функционалностите изисквания са както следва:

1. Докторант

* влизане в системата
* изход от системата
* промяна на профилна снимка
* премахване на профилна снимка
* премахване на известия
* създаване на месечен/годишен отчет
* качване на файлове за изпит
* Смяна на езика
* Преглед на календара му за изпити

1. Кандидат

* Изпращане на кандидатура за докторант
* Смяна на езика
* Създаване на учебен план

1. Администратор

* влизане в системата
* изход от системата
* промяна на профилна снимка
* премахване на профилна снимка
* премахване на известия
* Смяна на езика
* Премахни потребител от системата
* Филтриране на събития по ключова дума
* Разрешаване на неудостоверен потребител в системата
* Отчет на всички потребители в системата
* Отчет на всички събития
* Отчет на всички събития, разделени по година и месец
* Отчет на всички потребители в системата
* Отчет на всички потребители в системата

1. Член на докторантски център

* влизане в системата
* изход от системата
* промяна на профилна снимка
* премахване на профилна снимка
* премахване на известия
* смяна на езика
* разрешаване на неудостоверен потребител в системата
* задай комитет на избраният изпит
* отказ на кандидатура
* одобрение на кандидатура
* Отчет на всички учебни планове
* Отчет на всички неудостоверени потребители
* Отчет на всички кандитури
* Отчет на всички изпити

1. Член на комитет

* влизане в системата
* изход от системата
* промяна на профилна снимка
* премахване на профилна снимка
* премахване на известия
* смяна на езика
* разрешаване на неудостоверен потребител в системата
* задай комитет на избраният изпит
* отказ на кандидатура
* одобрение на кандидатура
* Отчет на всички изпити

## **4.3 Нефункционалности на приложението**

Нефункционалните изисквания на приложението са главно предназначение за поддръжка на приложението от администраторите.

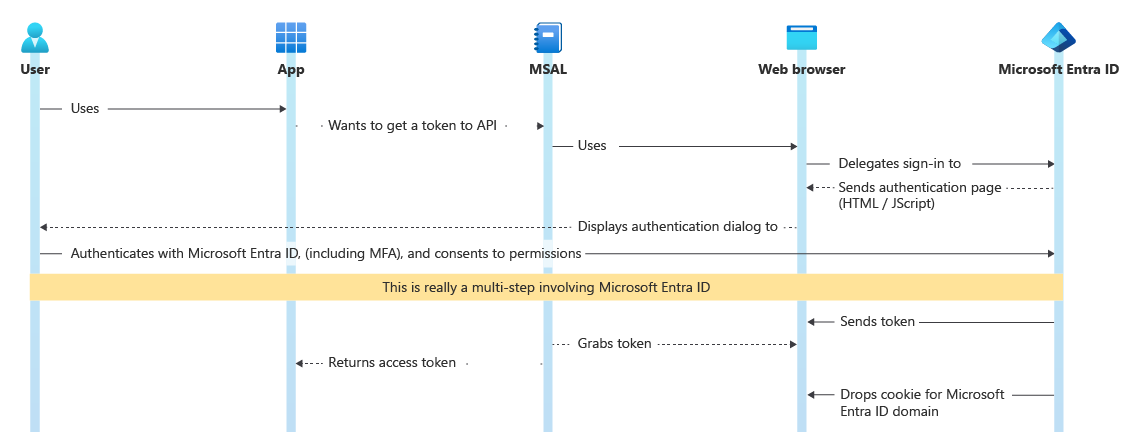
Това включва следните изисквания:

* Потребителите си подновяват JWT токена на всеки 30 минути
* Всички действия извършени от потребителите (с изключение на администраторите) трябва да бъдат изпратени като одит от събития. Успешни и неуспешни
* При настъпване на грешка от сървъра, приложението трябва да информира потребителя че не може да се свърже със сървъра и да опита по-късно. Тези грешки трябва да се изпратят като одит на събития, за да могат да бъдат прегледани от разработчиците, които са отговорни да поддържат приложението.
* При всяка заявка към сървърът, той проверява дали потребителят има необходимите права за да получи изисквания ресурс
* На всеки 10 секунди се прави заявка към сървърът за получаване на всички известия за удостовереният потребител
* На всеки 10 секунди се прави заявка към сървърът за подновяване на визуализираната таблица или табло за управление

## **4.4 Удостоверение**

За удостоверение на потребители и обработка на тяхното влизане/регистрация в приложението се използва Azure Active Directory Sign-in.

Azure Active Directory Sign-in е поток за удостоверяване, който предоставя напълно управляван потребителски интерфейс за влизане.

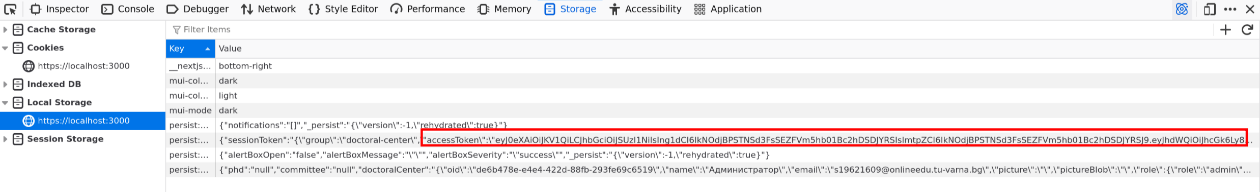


Фигура 65 Sequence диаграма за удостоверение на потребител в Azure Sign-in flow

Това са стъпките на удостоверяване към приложението :

1. **Заявка за удостоверяване**: Когато потребителя се опита да влезе в приложението, MSAL (Microsoft Authentication Library) го насочва към влизане към активната директория на Технически университет Варна чрез пренасочване в [https://login.microsoftonline.com/31886941-8a86-4f93-8f42-d140eaea36ad/oauth2/v2.0/authorize?client\_id=...](https://login.microsoftonline.com/31886941-8a86-4f93-8f42-d140eaea36ad/oauth2/v2.0/authorize?client_id=2e32e834-5b18-4f82-a0b8-32e623d944b4&scope=api%3A%2F%2F4e2b4a7f-3735-4cac-abd8-808c02dbe14d%2FBlazorHostedAPI.Access%20openid%20profile%20offline_access&redirect_uri=https%3A%2F%2Flocalhost%3A3000%2Fauthentication%2Flogin-callback&client-request-id=01978f00-2710-7203-b18f-68d32a1dd5c7&response_mode=fragment&client_info=1&nonce=01978f00-272b-7709-9888-e8d7746f5b4c&state=eyJpZCI6IjAxOTc4ZjAwLTI3MmItNzI0MS04OWQwLTIyOTVjNTViZTRkNCIsIm1ldGEiOnsiaW50ZXJhY3Rpb25UeXBlIjoicG9wdXAifX0%3D&x-client-SKU=msal.js.browser&x-client-VER=4.13.0&response_type=code&code_challenge=APedvpjmCqmHZP05voZw0SjdvsgaSoABTlC64UakYrE&code_challenge_method=S256)
2. **Вход на потребител:** Потребителят въвежда потребителско име и парола.
3. **Генериране на токени**: При успешно удостоверяване (Sends token**)** се генерира три токена: токен за идентификация (ID), токен за достъп (access) и токен за обновяване (refresh) .
4. **Пренасочване**: След това Azure Active Directory пренасочва потребителя обратно към URL адреса за обратно извикване на приложението. URL адресът за пренасочване ще съдържа код за удостоверяване (OAuth 2.0 токен),като параметър на заявката (**Provide tokens and sign in)** .
5. **Размяна за токени**: Приложението ще вземе кода за удостоверяване (OAuth 2.0 токен) и ще го обмени с токени.
6. **Съхраняване на токени**: След като получи токените от Azure Active Directory, приложението ще прочете JWT токените и ще ги съхрани в локалното хранилище на приложението (**Displays authentication dialog to)**.
7. **Удостоверена сесия**: установява се сесия за потребителя. ID-то съдържа информация за самоличността на потребителя. Токенът за достъп след това се използва за предоставяне на достъп до ресурсите в зависимост от ролята на потребителя

Пример :



Фигура 66 Токени на вече удостоверен потребител

1. **Рефреш на токена**: Ако токените са близо до изтичане, приложението ще извика автоматично друга заявка към Azure Active Directory автоматично (background), за да получи нови идентификатори и токени за достъп.
2. **Изход**: Когато потребителят иска да излезе, приложението премахва в локалното хранилище на потребителя токенът и сесията изтича, след което пренасочва потребителя обратно към началната страница за вход.

## **4.5 Автоматизация на внедряването на приложението**

За автоматизация на подновяването на приложението се използва CI/CD pipeline.

CI/CD означава непрекъсната интеграция (CI) и непрекъснато внедряване (CD). То е вид методология в разработката на софтуер, който ‚гарантира‘ качеството на сорс кода, когато разработчиците обединяват техния нов разработен сорс код в главния бранч в хранилището.

Първо се стартират автоматизирани процеси за изграждане (Building) на приложението, след това се стартира тестване на промените и накрая се внедрява кода в продукцията (т.е приложението което потребителите използват).

Основните цели на CI/CD се основават на по-бързото идентифициране и коригиране на софтуерни грешки, подобряване на качеството на кода и намаляване на продължителността, необходима за проверка и внедряване на промените в множествените фази на кода (разработка/тестване/продукция) .

След като са изяснени големите предимства за CI/CD, фазите (още наречени stages), на които новите промени преминават към хранилището (използва се Git в този проект), можем да анализираме какво се случва от промяна на сорс кода в хранилището до финалното приложение:

Фази:

1. **Предоставяне на агент (Set up job) :**

* Задаване на променливи на средата
* Инсталиране на ОС пакети за Docker
* Разпределяне на агент в Github за изчислителна мощност при извършване на останалите фази

1. **Проверка (Checkout)**

* Клонира се сорс кода на приложението - клиентското или сървърно приложение
* Клонирането на pipeline за тестване обикновено се извършва в бранча, който съдържа новите промени и се планира да се слива с главният бранч (master или main)
* Клониране при pipeline за внедряване се извършва в главният бранч

**3. Запазване на хранилището в артефакт (Save repo to artifact)**

- за да може в бъдеще на изпълнение на pipeline да изпълни втората фаза, тази фаза ще запази сорс кода в артефакт, които е хранилище за двоични файлове или сорс файлове. С него няма да бъде необходимо да се изпълни фаза 2, а ще се изтегли сорс кода директно от артефакта

**4. Задаване на JDK 21 (Setup JDK 21)**

* тъй като тестването на сървъра зависи от jdk 21, необходимо е да се инсталира пакета към агента
* За клиентското приложение се инсталира nodejs 23

**5. Направи gradlew за изпълнение (Make gradlew executable)**

* За да може да тестваме сървър приложението с quarkus, трябва да промениме правата на двоичният файл gradlew
* Командата за промяна на правата е:

chmod +x ./gradlew

**6. Тестване (Test)**

* За тестване може да се използва една команда на двоичният файл gradlew. Така всички тестове под директорията *src/test/java/com/tuvarna/phd/* ще бъдат изпълнени автоматично
* Командата за тестване е:

./gradlew test

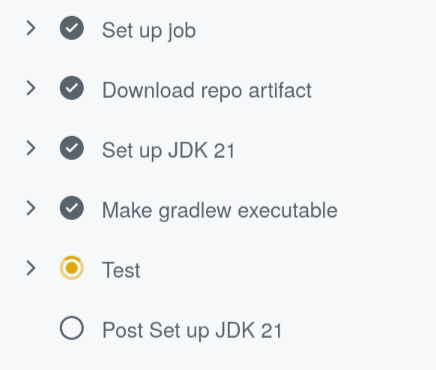
* Също така тестването съдържа сканиране на кода за уязвимости. Инструментите които се използват са GitGuardian за проверка на протичане на тайни ключове в сорс кода и SonarQube за проверка на качеството на кода за използването на библиотеки и функции, които са уязвими

**Проверка за формат на кода (Lint)**

* тази част е необходима за проверка на формата на кода
* Използва се в случайте, когато има повече от 1 разработчик за приложението за създаване на стандартизация на кода
* В Java се използва инструмента checkstyle, а за Javascript е eslint и prettier

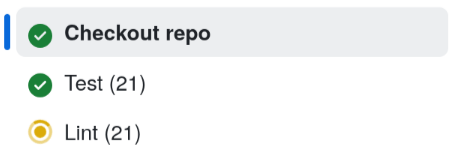
**7. Вграждане (Deploy)**

* Тази част се намира в друг pipeline, когато бранча с новите промени са слети с главният бранч
* Внедряването се случва като се създава нова версия на приложението във формата на Docker image, и след това се качва в Github Packages
* Накрая в Kubernetes, където се хоства приложенията се подновява версията на Docker image и накрая се внедрява новата версия в продукцията



Фигура 67 Стъпки на Тест фазатa

*Фиг.59 Архитектура на CI/CD*



Фигура 68 Фази на Github worklow за тестване

## **4.6 Изисквания към апаратното осигуряване**

**- Процесор: >** AMD Ryzen 7 8845HS w/ Radeon 780M Graphics (16) @ 5.102GHz

- **RAM**: > 32GB

**- Памет: >** 1TB SSD

- **GPU**: > NVIDIA GeForce RTX 4060 Max-Q / Mobile, AMD ATI Phoenix3

## **4.7 Инструкции за поддържане на системата**

За да може приложената да бъдат поддържани според тяхната сложност и следване на най-добри практики в софтуерната индустрия, следните аспекти трябва да се вземат под внимание (особено ако поддръжката ще се извърши от повече от един разработчик).

1. Спазване на структурата на приложенията
   1. В сървър проекта патерна, който се използва е *Model View Controller (MVC),* и той трябва да не се променя към друг патерн и да се поддържа така структура при по-нататъшното развитие на приложението. Разбира се може да има изключение в дадени случай, така че за тази цел е вече създаден индивидуален пакет наречен ‘model’ - то служи за дефиниране на класове за взаимодействие към външни системи.
   2. При клиентското приложение структурата е базирана на NextJS и всички маршрути, които са дефинирани под app директорията трябва да съдържат еднакъв брой на под-директории и page.jsx.
   3. Ако има необходимост да се автоматизира тази проверка на структура на проекта, тогава може да се направи скипт за проверка на новите директории при настъпване на заявка за изтегляне *(Pull request)*, и да се интегрира в продължителното тестване *(Continous integration)*
2. Добавяне на тестове при разработване на функционалности в приложението
   1. Тази част не може да се пропусне, тъй като това ще навреди на потребителското изживяване при настъпване на проблем в приложението.
   2. За сървър проекта тестовете са дефинирани в *src/test/java/com/tuvarna/phd ,* и може да се разширят тези тестове като се използва Mockito и стандартните пакети за тестване на Quarkus
   3. За клиентското приложение може да се доразработят край-до-край (end-to-end, E2E) тестове със Cypress и unit тестване с Jest
3. Тестване на формата на сорс кода
   1. При формата на тест кода, се има предвид тестване дали разстоянието между символите и празното разстояние се спазват
   2. При сървъра е вече автоматизирано с използването на checkstyle, където конфигурацията се намира в checkstyle.xml
   3. При клиента е също автоматизирано вече с използването на prettier и eslint
4. Тестване на сигурността
   1. При него може да се създадат много различни тестове за сигурността - сканиране на уязвимостите на библиотеките, сканиране на мрежата, симулиране на атака за експлоатация на приложенията и т.н. Най-важните тестове които са необходими да се използват са: сканиране на уязвимостите на библиотеките, сканиране на изтичане на тайни ключове в сорс кода и сканиране на методи в сорс кода, които представят уязвимост
   2. За сървър и клиентското приложение са вече разработени такиви тестове - Gitguardian за сканиране на изтичане на тайни ключове в сорс кода, SonarQube за сканиране на методи в сорс кода, които представят уязвимост и Github Dependabot за сканиране на уязвимостите на библиотеките
5. Преглед на таблата за управление с Grafana
   1. Особено при първо пускане на приложенията в продукция, където още не се знае колко наистина са стабилни и дали могат да издържат на заявките от потребителите, монтиронга и следенето на статуса на приложенията е важен компонент, за да може да се видят какви компоненти от приложенията не работят както предвидено, и следователно да се оправят тези проблеми възможно най-скоро
   2. Добра практика е също да се създадат предефинирани панели за графичните диаграми, за да може да се погледне на едно място най-важните части, които не трябва да се провалят. Един от тях е: % използване на процесора и памет, http заявките, преглед на одит на събитията, изпълнение (performance) на базата от данни, честота на генерирането на грешки от приложението, латентност (latency) на отговор на клиентските заявки

## **4.8 Възможности за развитие на програмата**

Макар че приложенията покриват основните потребителски изисквания (плюс някои допълнителни за постигане на по-добро потребителско изживяване), приложението може да се доразвие още - повече за създаване на потребителските функционалности и по-малко за поддръжката на приложенията от страна на разработчика.

* Създаване на функционалност за потребителите да си създадат диаграми със собствена логика за агрегация и избор на вид визуализация на диаграмата
* Създаване на повече графични диаграми
* Позволяване на преместване на панелите на графичните диаграми
* Създаване на функционалност за влизане на повече от един профил едновременно
* Създаване на мобилна версия на приложението
* Създаване на клавишни команди за достъп до функционалностите от панела
* Създаване на re-Captcha при регистрацията на кандидате
* Създаване на функционалност за вход на профила на кандидата
* Създаване на офлайн режим на приложението
* Създаване на функционалност за автоматично създаване на потребител в Azure Active Directory, след когато кандидата е одобрен за изпитна стъпка 3 (т.е последната)
* Миграция на приложенията в облака
* Генериране на известие към разработчиците и потребителите, когато се издаде нова версия (release), на приложенията, за да може всички да бъдат информирани за новата версия
* Създаване на общо известие за всички потребители в приложението, за да бъдат информирани какви нови функционалности са добавени в продукцията
* Създаване на 2 среди - за разработка и за тестване
* Създаване на глас-върху-текст за подобряване на функции за достъпност

# **5.0 Тестове и резултати**

## **5.1** **Тестване на бакенд средата**

За тестване на бакенд средата е използвана Mockito фраймлурк и Quarkus Junit .

Mockito е начин за заместване на обекти за база от данни, API методи с `фалшиви`, за създаване на симулация на логиката на такиви обекти, и следователно потвърждаване на очаквания върнат резултат от тези обекти.

### **5.1.1 Unit тестове**

Направени са тестове на следните service методи :

1. В ролята за кандидат

* кандидатстване за докторантура
* създаване на учебен план

2. В ролята на член на докторантски център

* извличане на всички неудостоверени потребители
* задаване на комисия за изпит

**TestCandidateService.java**

1 @BeforeAll

2 **public** **static** void setup() {

3 CandidateStatusRepository candidateStatusRepository2 =

4 Mockito.mock(CandidateStatusRepository.**class**);

5 CandidateRepository candidateRepository2 = Mockito.mock(CandidateRepository.**class**);

6 ModeRepository modeRepository2 = Mockito.mock(ModeRepository.**class**);

7 CurriculumRepository curriculumRepository2 = Mockito.mock(CurriculumRepository.**class**);

8

9 Mockito.**when**(candidateStatusRepository2.getByStatus("enrolled"))

10 .thenReturn(**new** CandidateStatus(**null**, "enrolled"));

11 Mockito.**when**(curriculumRepository2.getByName("curriculum1"))

12 .thenReturn(**new** Curriculum("curriculum1"));

13 Mockito.**when**(candidateRepository2.getByEmail("john.doe@mail.com"))

14 .thenReturn(**new** Candidate("john.doe@mail.com"));

15 Mockito.**when**(modeRepository2.getByMode("regular")).thenReturn(**new** Mode("regular"));

16

17 QuarkusMock.installMockForType(candidateStatusRepository2, CandidateStatusRepository.**class**);

18 QuarkusMock.installMockForType(curriculumRepository2, CurriculumRepository.**class**);

19 QuarkusMock.installMockForType(candidateRepository2, CandidateRepository.**class**);

20 QuarkusMock.installMockForType(modeRepository2, ModeRepository.**class**);

21 }

22

23 @Test

24 **public** void testApply() {

25 CandidateApplyDTO candidateApplyDTO =

26 **new** CandidateApplyDTO(

27 "John",

28 "john.doe@mail.com",

29 "1111111",

30 "Bulgaria",

31 "Varna",

32 "Primorski, 9002 Varna, Bulgaria",

33 "biography113.jpg",

34 2025L,

35 9000L,

36 "faculty1",

37 "enrolled",

38 **new** CurriculumCreateDTO(

39 "Curriculum123",

40 "regular",

41 "faculty1",

42 Arrays.asList("discipline1", "discipline2")));

43

44 Candidate candidate = **new** Candidate();

45 CandidateStatus candidateStatus = **new** CandidateStatus(1L, "enrolled");

46 candidate.setStatus(candidateStatus);

47

48 **when**(this.candidateMapper.toEntity(candidateApplyDTO)).thenReturn(candidate);

49 **when**(this.facultyRepository.getByName("faculty1")).thenReturn(**new** Faculty(1L, "faculty"));

50

51 **when**(ipBlockService.isClientIPBlocked()).thenReturn(**false**);

52

53 **when**(this.candidateStatusRepository.getByStatus("enrolled")).thenReturn(candidateStatus);

54 **when**(this.candidateRepository.getByEmail("john.doe@mail.com")).thenReturn(candidate);

55

56 assertEquals("enrolled", candidate.getStatus().getStatus());

57 }

58

59 @Test

60 **public** void testCreateCurriculum() {

61 CurriculumDTO curriculumDTO =

62 **new** CurriculumDTO(

63 "curriculum1",

64 "regular",

65 "2025",

66 "faculty1",

67 Arrays.asList("discipline1", "discipline2"));

68

69 Curriculum curriculum =

70 **new** Curriculum(

71 "curriculum1",

72 **new** Mode("enrolled"),

73 **Set**.**of**(**new** Subject("subject1"), **new** Subject("subject2")));

74

75 **when**(this.curriculumMapper.toEntity(curriculumDTO)).thenReturn(curriculum);

76

77 curriculum.setMode(this.modeRepository.getByMode(curriculumDTO.getMode()));

78 curriculum.setFaculty(this.facultyRepository.getByName(curriculumDTO.getFaculty()));

79

80 this.curriculumRepository.**save**(curriculum);

81 }

**TestDoctoralCenterService.java**

1 @BeforeAll

2 **public** **static** void setup() {

3 UnauthorizedRepository unauthorizedRepository = Mockito.mock(UnauthorizedRepository.**class**);

4 GradeRepository gradeRepository = Mockito.mock(GradeRepository.**class**);

5 CommissionRepository commissionRepository = Mockito.mock(CommissionRepository.**class**);

6

7 Mockito.**when**(unauthorizedRepository.getByOid("0000")).thenReturn(**new** Unauthorized("0000"));

8 Mockito.**when**(gradeRepository.getById(1L)).thenReturn(**new** Grade(1L));

9 Mockito.**when**(commissionRepository.getById(1L)).thenReturn(**new** Commission(1L, "commission1"));

10

11 QuarkusMock.installMockForType(unauthorizedRepository, UnauthorizedRepository.**class**);

12 QuarkusMock.installMockForType(gradeRepository, GradeRepository.**class**);

13 QuarkusMock.installMockForType(commissionRepository, CommissionRepository.**class**);

14 }

15

16 @Test

17 **public** void testGetUnauthorizedUsers() {

18 List<Unauthorized> unauthorized = List.**of**(**new** Unauthorized("0000"));

19

20 **when**(this.unauthorizedRepository.getByOid("0000")).thenReturn(unauthorized.**get**(0));

21

22 assertEquals("0000", unauthorized.**get**(0).getOid());

23 }

24

25 @Test

26 **public** void testSetCommissionOnGrade() throws IOException {

27 Long id = 1L;

28 String name = "commission1";

29

30 Commission commission = **new** Commission(1L, name);

31 Grade grade =

32 **new** Grade(

33 id, 6.00, **new** **Date**(**System**.currentTimeMillis()), "report1", **new** Subject("subject1"));

34 grade.setCommission(commission);

35

36 **when**(this.gradeRepository.getById(1L)).thenReturn(grade);

37 **when**(this.commissionRepository.getByName(name)).thenReturn(commission);

38

39 assertEquals(1L, grade.getId());

40 assertEquals("commission1", commission.getName());

41 assertEquals(commission, grade.getCommission());

42

43 doNothing()

44 .**when**(this.mailModel)

45 .send(

46 "Вие сте добавен в изпит",

47 TEMPLATES.COMMITTEE\_ADDED\_TO\_EXAM,

48 "john.doe@mail.com",

49 **Map**.**of**("$EVAL\_DATE", grade.getEvalDate().toString()));

50 }

### **5.1.2 Integration тестове**

Направени са тестове на следните контролер методи :

1. В ролята за кандидат

* кандидатстване за докторантура
* извличане на всички кандидати, които са със статус „в преглед“ на тяхната кандидатура
* извличане на всички учебни програми

2. В ролята на член на докторантски център

* приеми/откажи кандидатура
* задаване на комисия за изпит
* извличане на всички кандидати

1 All 10 tests are passing (0 skipped), 10 tests were run in 3961ms. Tests completed at 10:14:40.

**TestCandidateController.java**

1 @Test

2 void testApply() {

3 doNothing().**when**(candidateService).apply(**any**());

4

5 given()

6 .contentType(ContentType.JSON)

7 .body(candidateApplyDTO)

8 .**when**()

9 .post("/apply")

10 .**then**()

11 .statusCode(200)

12 .body("message", **is**("Candidate application finished successfully!"));

13

14 verify(candidateService).apply(candidateApplyDTO);

15 }

16

17 @Test

18 void testInReview() {

19 List<CandidateDTO> candidateDTOs =

20 List.**of**(

21 **new** CandidateDTO(

22 "John",

23 "john.doe@mail.com",

24 "1",

25 "111111111",

26 "Bulgaria",

27 "Varna",

28 "Primorski, 9002 Varna, Bulgaria",

29 "9000",

30 2025L,

31 "biography113.jpeg",

32 "curriculum1",

33 "faculty1",

34 "enrolled"));

35

36 **when**(candidateService.getCandidatesInReview()).thenReturn(candidateDTOs);

37 Response response = candidateController.getCandidatesInReview();

38 assertEquals(200, response.getStatus());

39

40 ControllerResponse controllerResponse = (ControllerResponse) response.getEntity();

41 assertEquals("Candidates in review retrieved!", controllerResponse.message());

42 assertEquals(candidateDTOs, controllerResponse.**data**());

43 }

44

45 @Test

46 void testGetCurriculums() {

47

48 List<CurriculumDTO> curriculumDTOs =

49 List.**of**(

50 **new** CurriculumDTO(

51 "curriculum1",

52 "part\_time",

53 "2025",

54 "faculty1",

55 Arrays.asList("discipline1", "discipline2")));

56

57 **when**(candidateService.getCurriculums()).thenReturn(curriculumDTOs);

58 Response response = candidateController.getCurriculums();

59 assertEquals(200, response.getStatus());

60

61 ControllerResponse controllerResponse = (ControllerResponse) response.getEntity();

62 assertEquals("Curriculums retrieved", controllerResponse.message());

63 assertEquals(curriculumDTOs, controllerResponse.**data**());

64 }

**TestDoctoralCenterController.java**

1 @Test

2 void testReview() throws IOException {

3 String email = "john.doe@mail.com";

4 String status = "rejected";

5 doNothing().**when**(doctoralCenterService).review(anyString(), anyString());

6

7 given()

8 .contentType(ContentType.JSON)

9 .pathParam("email", email)

10 .pathParam("status", status)

11 .**when**()

12 .patch("/candidate/{email}/application/{status}")

13 .**then**()

14 .statusCode(200)

15 .body("message", **is**("Candidate's applicaton is changed to: " + status));

16

17 verify(doctoralCenterService).review(email, status);

18 }

19

20 @Test

21 void testSetCommissionOnGrade() {

22 Long id = 1L;

23 String name = "commission1";

24 doNothing().**when**(doctoralCenterService).setCommissionOnGrade(id, name);

25

26 given()

27 .contentType(ContentType.JSON)

28 .pathParam("id", id)

29 .pathParam("name", name)

30 .**when**()

31 .patch("/grade/{id}/commission/{name}")

32 .**then**()

33 .statusCode(200)

34 .body("message", **is**("Grade updated with commision: " + name + " !"));

35

36 verify(doctoralCenterService).setCommissionOnGrade(id, name);

37 }

38

39 @Test

40 void testGetCandidates() {

41 String fields = "name";

42 List<CandidateDTO> candidateDTOs =

43 List.**of**(

44 **new** CandidateDTO(

45 "John",

46 "john.doe@mail.com",

47 "1",

48 "111111111",

49 "Bulgaria",

50 "Varna",

51 "Primorski, 9002 Varna, Bulgaria",

52 "9000",

53 2025L,

54 "biography113.jpeg",

55 "curriculum1",

56 "faculty1",

57 "enrolled"));

58

59 **when**(doctoralCenterService.getCandidates(fields)).thenReturn(candidateDTOs);

60 Response response = doctoralCenterController.getCandidates(fields);

61 assertEquals(200, response.getStatus());

62

63 ControllerResponse controllerResponse = (ControllerResponse) response.getEntity();

64 assertEquals("Candidates retrieved!", controllerResponse.message());

65 assertEquals(candidateDTOs, controllerResponse.**data**());

66 }

## **5.2 Тестване на фронтенд средата**

За тестване на фронтенда са направени Unit тестове, Snapshot тестове за провека дали страницата не се презарежда ненужно, и край-до-край тестове E2E със Cypress.

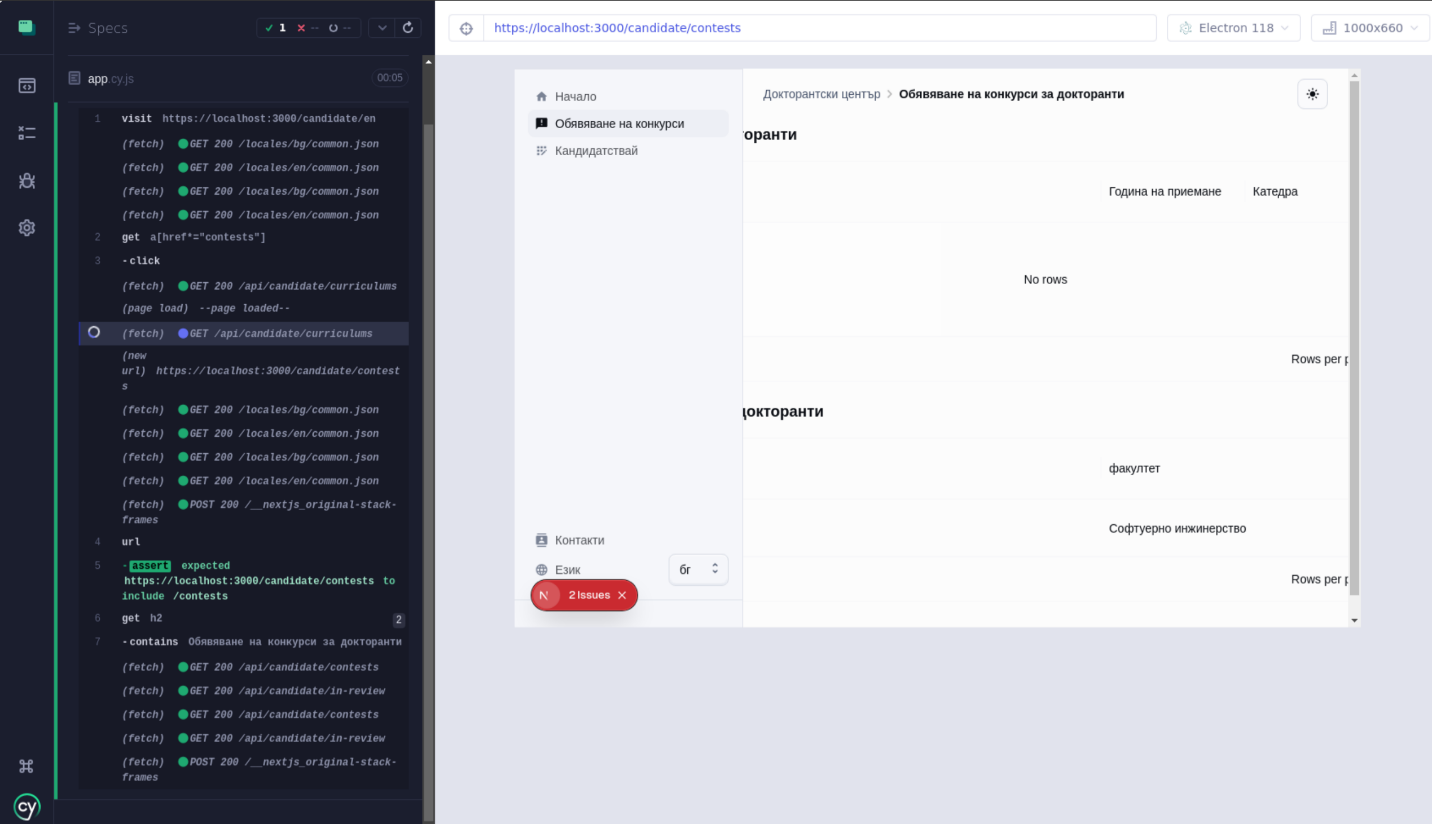
### **5.2.1 Резултати при Unit и Snapshot тестове**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **File** | **% Stmts** | **% Branch** | **% Funcs** | **% Lines** | **Uncovered Line #s** |
| All files | 65.98 | 65.77 | 35.02 | 65.98 |  |
| app | 96.55 | 66.66 | 100 | 96.55 |  |
| loading.js | 96.55 | 66.66 | 100 | 96.55 | 15 |
| app/(common)/\_profile/\_components | 98.68 | 33.33 | 20 | 98.68 |  |
| ProfileGrid.jsx | 98.68 | 33.33 | 20 | 98.68 | 57,64 |
| app/(common)/\_profile/\_hooks | 34.78 | 100 | 33.33 | 34.78 |  |
| ProfileHook.jsx | 34.78 | 100 | 33.33 | 34.78 | 21-55,58-82 |
| app/candidate/(home) | 100 | 100 | 100 | 100 |  |
| page.jsx | 100 | 100 | 100 | 100 |  |
| app/candidate/\_constants | 97.05 | 100 | 50 | 97.05 |  |
| columnsConstant.jsx | 97.02 | 100 | 50 | 97.02 | 26-28 |
| pathConstant.jsx | 100 | 100 | 100 | 100 |  |
| app/candidate/\_hooks | 90.68 | 86.36 | 100 | 90.68 |  |
| ApplyHook.jsx | 84.37 | 75 | 100 | 84.37 | 40-42,60-65,72-77 |
| ContestsHook.jsx | 100 | 100 | 100 | 100 |  |
| app/doctoral-center/admin/\_constants | 100 | 100 | 100 | 100 |  |
| pathConstant.jsx | 100 | 100 | 100 | 100 |  |
| sideMenuConstants.jsx | 100 | 100 | 100 | 100 |  |
| app/doctoral-center/admin/profile | 100 | 100 | 100 | 100 |  |
| page.jsx | 100 | 100 | 100 | 100 |  |
| components/cells-renderers | 84.61 | 100 | 0 | 84.61 |  |
| link.jsx | 84.61 | 100 | 0 | 84.61 | 12-13 |
| components/dialog-box | 90.74 | 100 | 33.33 | 90.74 |  |
| ConfirmDialogYesNo.jsx | 90.74 | 100 | 33.33 | 90.74 | 22-23,43-45 |
| components/main-layout | 92.85 | 40 | 100 | 92.85 |  |
| Layout.jsx | 92.85 | 40 | 100 | 92.85 | 35-36,46,60-61 |
| components/main-layout/common | 70.17 | 95.23 | 60 | 70.17 |  |
| AlertBox.jsx | 100 | 100 | 33.33 | 100 |  |
| AppNavbar.jsx | 98.18 | 100 | 100 | 98.18 | 36-37 |
| CardAlert.jsx | 32 | 100 | 0 | 32 | 9-25 |
| CustomDatePicker.jsx | 37.97 | 100 | 0 | 37.97 | 11-36,57-79 |
| CustomTable.jsx | 100 | 100 | 100 | 100 |  |
| Header.jsx | 27.5 | 100 | 0 | 27.5 | 23-80 |
| HeaderPublic.jsx | 100 | 100 | 100 | 100 |  |
| MenuButton.jsx | 100 | 100 | 100 | 100 |  |
| MenuContent.jsx | 100 | 100 | 100 | 100 |  |
| NavbarBreadcrumbs.jsx | 100 | 66.66 | 100 | 100 | 11 |
| OptionsMenu.jsx | 23.33 | 100 | 0 | 23.33 | 22-90 |
| SideMenu.jsx | 33.33 | 100 | 0 | 33.33 | 28-81 |
| SideMenuMobile.jsx | 100 | 100 | 100 | 100 |  |
| SideMenuPublic.jsx | 100 | 100 | 100 | 100 |  |
| components/main-layout/theme/customizations | 28.31 | 100 | 0 | 28.31 |  |
| charts.js | 42.66 | 100 | 0 | 42.66 | 10-28,34-37,39-45,60-72 |
| dataGrid.js | 23.88 | 100 | 0 | 23.88 | 18-44,46-47,49-64,66-78,80-99,101-118,120-122,124-126 |
| datePickers.js | 26.4 | 100 | 0 | 26.4 | 16-32,39-44,58-94,100-135,141-175 |
| index.js | 100 | 100 | 100 | 100 |  |
| treeView.js | 21.31 | 100 | 0 | 21.31 | 10-28,30-58 |
| components/shared-theme | 70.2 | 57.14 | 57.14 | 70.2 |  |
| AppTheme.jsx | 94.44 | 50 | 100 | 94.44 | 15,36-37 |
| ColorModeIconDropdown.jsx | 92.22 | 60 | 60 | 92.22 | 15-16,18-19,21-23 |
| themePrimitives.js | 61.61 | 100 | 0 | 61.61 | 73-219 |
| components/shared-theme/customizations | 71.17 | 33.33 | 46.87 | 71.17 |  |
| dataDisplay.jsx | 53.21 | 50 | 60 | 53.21 | 74-79,95-197 |
| feedback.jsx | 58.69 | 50 | 33.33 | 58.69 | 9-20,37-43 |
| inputs.jsx | 83.85 | 33.33 | 44.44 | 83.85 | 281-292,298-309,325-364,431-435,441-443 |
| navigation.jsx | 66.3 | 28.57 | 50 | 66.3 | 17-27,123-147,160-170,220-224,230-266,272-276 |
| surfaces.js | 75.22 | 25 | 33.33 | 75.22 | 13-32,38-45 |
| lib/api | 54 | 100 | 30 | 54 |  |
| file.jsx | 52.5 | 100 | 25 | 52.5 | 11-17,20-25,28-33 |
| logs.jsx | 81.25 | 100 | 50 | 81.25 | 9-11 |
| notification.jsx | 45.45 | 100 | 25 | 45.45 | 10-11,14-33,36-37 |
| lib/api/router | 24.63 | 100 | 33.33 | 24.63 |  |
| client.jsx | 24.63 | 100 | 33.33 | 24.63 | 11-52,55-64 |
| lib/auth | 54.09 | 50 | 16.66 | 54.09 |  |
| auth.jsx | 50.68 | 50 | 20 | 50.68 | 21-31,34-38,41-46,51-64 |
| authConfig.js | 59.18 | 100 | 0 | 59.18 | 20-39 |
| lib/features | 100 | 100 | 100 | 100 |  |
| constants.js | 100 | 100 | 100 | 100 |  |
| lib/features/notification/slices | 71.42 | 100 | 0 | 71.42 |  |
| notificationsMemoSelector.jsx | 75 | 100 | 0 | 75 | 10-13 |
| notificationsSlice.jsx | 69.69 | 100 | 0 | 69.69 | 12-14,16-20,23-24 |
| lib/features/sessionToken/slices | 73.46 | 66.66 | 25 | 73.46 |  |
| sessionTokenMemoSelector.jsx | 94.11 | 66.66 | 100 | 94.11 | 12 |
| sessionTokenSlice.jsx | 62.5 | 100 | 0 | 62.5 | 12-19,22-23,25-26 |
| lib/features/uiState/slices | 76.92 | 100 | 60 | 76.92 |  |
| uiStateMemoSelector.jsx | 100 | 100 | 100 | 100 |  |
| uiStateSlice.jsx | 73.91 | 100 | 0 | 73.91 | 12-13,15-18 |
| lib/features/user/slices | 70.9 | 66.66 | 16.66 | 70.9 |  |
| userMemoSelector.jsx | 81.39 | 66.66 | 50 | 81.39 | 15-16,21-24,32,41 |
| userSlice.jsx | 64.17 | 100 | 0 | 64.17 | 19-21,23-25,27-33,35-37,40-41,44-45,48-49,52-53 |
| lib/helpers | 36.03 | 85.71 | 31.25 | 36.03 |  |
| APIWrapper.jsx | 31.42 | 100 | 25 | 31.42 | 12-35,38-44,47-63 |
| Translate.jsx | 75 | 66.66 | 66.66 | 75 | 13-17 |
| utils.jsx | 33.12 | 100 | 22.22 | 33.12 | 21-33,36-41,44-46,56-93,96-123,126-131,147-157 |
| models | 63.91 | 100 | 6.66 | 63.91 |  |
| Candidate.jsx | 100 | 100 | 100 | 100 |  |
| Committee.jsx | 52.63 | 100 | 0 | 52.63 | 5-20,23-24 |
| Curriculum.jsx | 100 | 100 | 100 | 100 |  |
| DoctoralCenter.jsx | 94.28 | 100 | 50 | 94.28 | 13-14 |
| DoctoralCenterRole.jsx | 84.61 | 100 | 0 | 84.61 | 7-8 |
| File.jsx | 100 | 100 | 100 | 100 |  |
| Log.jsx | 28.07 | 100 | 0 | 28.07 | 10-14,17-20,23-41,44-56 |
| Notification.jsx | 76.92 | 100 | 0 | 76.92 | 11-16 |
| Phd.jsx | 50.5 | 100 | 0 | 50.5 | 19-56,59-60,62-63,66-70,73-74 |
| models/auth | 100 | 100 | 100 | 100 |  |
| SessionToken.jsx | 100 | 100 | 100 | 100 |  |
| --------------------------------------------- | --------- | ---------- | --------- | --------- | ------------------------------------------------------- |

Snapshot Summary › 1 snapshot written from 1 test suite.

Test Suites: 4 passed, 4 total Tests: 5 passed, 5 total Snapshots: 1 written, 1 passed, 2 total Time: 2.256 s Ran all test suites.

### **5.2.2 Резултати при E2E тестове**



Фигура 69 E2E тестове

# **6.0 Заключение**

Създаденото приложение за прием и обучение на докторанти позволява на административните и обучаваните потребители да им помогне за управлението на кандидатурите, администрацията в системата, оценяването на кандидати и докторанти, одит на събития и персонализация в приложението.

Някой от най-важните функционалности поддържат:

* Създаване на графични диаграми според нуждите на потребителя, разделени по роля
* Превеждане на английски/български език
* Филтриране на одит на събития според атрибут и ключова дума
* Изпращане на известие за индивидуални и групови (по роля) потребители
* Изпращане на имейл съобщения на потребители
* Позволяване на достъп към приложението само с профил, който съществува в Azure Active Directory
* Качване на файлове в облака за централизиране на файловете през много инстанции на сървъра
* Управление на достъпа на неудостоверени потребители от администратора и член на докторантския център
* Предоставяне на панели за мониторинг на статуса на приложението с Grafana

След доказаните стрес тестове към клиентското и сървърно приложение, финалният им резултат доказва тяхната ефективност за постигане на потребителските изисквания, лесната им поддръжка и адаптиращият се интерфейс за създаване на добро изживяване като краен потребител, администратор и разработчик.

# **7.0 Литература**

[1]Developing a Systems Engineering Concept for Digitalizing Higher Education Institutions, by Iurii Teslia, Nataliia Yehorchenkova, Iulia Khlevna, Yevheniia Kataieva, Tatiana Latysheva, Oleksii Yehorchenkov, Andrii Khlevnyi, Vitaliy Veretelnyk**:** <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3759735>

[2]Digitalization in university education: didactic aspects, by Mariia A. Abysova, Mariia H. Kravchuk, Oksana M. Hurniak: <http://ir.librarynmu.com/bitstream/123456789/8794/1/%D0%A1%D1%82.%20-%20Abysova%20M.A.%2C%20Kravchuk%20M.H%2C%20Hurniak%20O.M.%2C%202023.pdf>

[3] Indicators for determining the effective level of digitalization in higher education, by svetana Stoyanova 1, Kiril Anguelov: <https://www.researchgate.net/profile/Kiril-Anguelov-2/publication/379429806_Indicators_for_determining_the_effective_level_of_digitalization_in_Higher_Education/links/674edfbb359dcb4d9d4efe61/Indicators-for-determining-the-effective-level-of-digitalization-in-Higher-Education.pdf?__cf_chl_rt_tk=cmEdBeXjEYbfXL1PGGdPXjqJhChVkZAMQLrlyr5GMnA-1750708331-1.0.1.1-SMnSOlzwpgbnPpcdUa95LZOEUOVNTO.jVZFGBsp95EE>

[4] Java Documentation: <https://docs.oracle.com/en/java/>

[5] Quarkus Documentation : <https://quarkus.io/guides/>[#](https://quarkus.io/guides/#/)

[6] Javascript: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

[7] ReactJS Documentation: <https://react.dev/>

[8] NextJS Documentation: <https://nextjs.org/docs>

[9] PostgreSQL Docs: <https://www.postgresql.org/docs/>

[10] Docker Documentation: <https://docs.docker.com/>

[11] Kubernetes Documentation: <https://kubernetes.io/docs/home/>

[12] GNU Make Documentation: <https://www.gnu.org/software/make/manual/>

[13] Nix Documention :<https://nix.dev/manual/nix/2.18/>

[14] Woodpecker Documentation: <https://woodpecker-ci.org/>

[15] AWS Documentation: <https://docs.aws.amazon.com/>

[16] Azure Documentation: https://learn.microsoft.com/en-us/azure/?product=popular

[17] Figma Documentation : <https://help.figma.com/hc/en-us>

[18] ElasticSearch Documentation: <https://www.elastic.co/>

[19] MaterialUI Documentation: <https://mui.com/>

[20] O“Relly Practical PostgreSQL: <https://books.google.bg/books?hl=en&lr=&id=G8dh95j5NgcC&oi=fnd&pg=PR4&dq=postgresql&ots=8U53LXCVk3&sig=ojHSxulz0A3t-stC8e9CpmXrltw&redir_esc=y#v=onepage&q=postgresql&f=false>

[21] Active Directory 101: <https://www.serveracademy.com/blog/active-directory-101-a-step-by-step-tutorial-for-beginners/>

[22] File based routing: <https://tanstack.com/router/latest/docs/framework/react/routing/file-based-routing>

## **7.1 Цитати**

1. Европейска комисия 2022

2. „Digital Natives, Digital Immigrants“ – М. Пренски, 2001 г. Терминът се използва за да опише новото поколение на студенти и нуждата от адаптиране към дигитализацията от преподавателите.

3. Еванс, Миклосик и Дю (2023)

## **7.2 Глосар**

1. Production = Живата среда, в която работи финалната версия на система или приложение и се използва от реални потребители.
2. Attack vector = Метод или път, използван от хакер за получаване на неоторизиран достъп до система.
3. Software Development Lifecycle (SDLC) = то е структуриран процес, който изгражда етапите, свързани с разработката на софтуерни приложения, включително планиране, проектиране, програмиране, тестване, внедряване и поддръжка на системата, за да се гарантира качество и ефективност.
4. Micro-architecture = отнася се до вътрешната структура или шаблона за проектиране в рамките на софтуерен модул или компонент. Той определя как са организирани отговорностите, как взаимодействат класовете или функциите и как специфични принципи на проектиране (като SOLID или чиста архитектура) се прилагат в малък мащаб – в рамките на услуга, модул или слой.
5. Loosely-coupled = описва системен дизайн, при който компонентите или услугите са независими и взаимодействат чрез добре дефинирани интерфейси, минимизирайки зависимостите. Това подобрява гъвкавостта, мащабируемостта и поддръжката.
6. Bugs = грешка, недостатък или непредвидено поведение в софтуера, което води до неправилни резултати, срив или функциониране по начин, различен от очаквания. Грешките могат да възникнат поради грешки в програмирането, логически грешки или несъответствия в изискванията.
7. Antipattern = неефективно или контра-продуктивно решение на повтарящ се проблем в дизайна или разработката на софтуера. В началото може да изглежда полезно, но често води до лошо качество на кода, проблеми с поддръжката или проблеми с производителността в дългосрочен план.
8. Document Object Model (DOM) = Това е интерфейс за програмиране за уеб страницата, така че програмите да могат да променят структурата на документите си, стила и съдържанието и т.н. DOM е йерархично представяне на структурата на една уеб страница.