



СУ "Св. Климент Охридски",
Факултет по математика и информатика,
Катедра Информационни Технологии

**Разпределена система за управление на
курсове по програмиране с автоматично
оценяване на решения**

Здравко Иванов Гюров, РСМТ, ФН: 26379
Научен ръководител: д-р Стоян Велев

София, 2022

Съдържание

Списък на съкращенията	4
Речник на термините	4
Списък на фигурите	6
Списък на таблиците	6
Увод	6
Структура на дипломната работа	7
Дефиниции	8
Обзор на подобни системи	8
Подобни системи	8
Web-CAT	9
Основни силни страни	10
Опит със системата	11
codePost	14
Основни силни страни	15
Опит със системата	16
GitHub Classroom	18
Основни силни страни	19
Използване на системата	20
Сравнителен анализ	22
Обобщение	23
Проектиране на системата	24
Разрешаване проблемите на конкурентните системи	24
Предоставяне на функционалностите, съществуващи в конкурентните системи	27
Изпълняване на кода във всякаква среда с Docker	28
Какво е контейнер?	28
Предимствата на Docker контейнерите	29
Оркестриране на контейнерите с Kubernetes	30
Какво е Kubernetes?	30
Основни характеристиките	30
Проектиране на базата от данни	32
Проектиране на уеб интерфейса	36
Проектиране на API сървъра	38
Проектиране на компонента за изпълняване на тестове	40
Инсталиране на системата	41
Реализиране на системата	44

Основни работни потоци	45
Реализиране на уеб интерфейса	50
Реализиране на API сървъра	54
Реализиране на компонента за изпълняване на тестове	55
Реализиране на Kubernetes шаблоните	55
Експерименти и анализ на резултатите	55
Заклучение	55
Постигнати резултати	55
Приноси	55
Научни	55
Научно-приложни	56
Приложни	56
Апробация	56
Насоки за бъдеща работа; Перспективи	56
Използвана литература	56

I. Списък на съкращенията

API - Application Programming Interface

WYSIWYG - What You See Is What You Get

REST - Representational State Transfer

MOSS - Measure of Software Similarity

UUID - Universally unique identifier

CRUD - Create, Read, Update, Delete

GCP - Google Cloud Platform

NFS - Network File System

iSCSI - Internet Small Computer Systems Interface

SPA - Single Page Application

SQL - Structured Query Language

NoSQL - Not only Structured Query Language

CAP - data consistency, system availability and partition tolerance

ACID - Atomicity, Consistency, Isolation, Durability

BASE - Basically available, Soft-state, Eventual consistency

JSON - JavaScript Object Notation

HTML - HyperText Markup Language

XML - Extensible Markup Language

II. Речник на термините

Application Programming Interface - проложно-програмен интерфейс

Online Judge - онлайн система за автоматично оценяване на решения на задачи по програмиране

Java Servlet - програма, която работи върху уеб сървър; междинното ниво между заявките, идващи от уеб браузъра и базите от данни или други приложения на уеб сървъра

plug-in - софтуерен компонент, който представлява “приставка”, която се инсталира в допълнение към съществуващо софтуерно приложение, за да предостави допълнителна функционалност

What You See Is What You Get - системи, при които съдържанието по време на редактиране изглежда като крайния резултат

Representational state transfer - стил софтуерна архитектура за реализация на уеб услуги

Measure of Software Similarity - система на Станфорд за откриване на сходен изходен код

Universally unique identifier - универсален уникален идентификатор

Google Cloud Platform - облачната платформа на Google, която представлява набор от облачни услуги

Network File System - мрежова файлова система

Internet Small Computer Systems Interface - стандарт за съхранение на данни, базиращ се на интернет протокола

Single Page Application - уеб сайт съдържащ само една HTML страница, който предоставя динамично съдържание чрез пренаписване на страницата, използвайки JavaScript

Structured Query Language - език за писане на структурирани заявки към бази от данни

Data consistency - последователност на данните

System availability - наличност на системата

Partition tolerance - толерантност на дялове

Atomicity - атомарност

Consistency - последователност

Isolation - изолация

Durability - издръжливост

JavaScript Object Notation - формат на данни, базиращ се на JavaScript, лесен за четене от хора

HyperText Markup Language - маркиращ език, използва се за създаване на уеб страници

Extensible Markup Language - екстензивен маркиращ език

III. Списък на фигурите

Фигура 1 - Диаграма на базата от данни

Фигура 2 - Диаграма на Kubernetes архитектурата

Фигура 3 - Диаграма на потребителските случаи

Фигура 4 - Процес на удостоверяване - диаграма на последователност

Фигура 5 - Процес на създаване на курс - диаграма на последователност

Фигура 6 - Процес на оценяване на решение - диаграма на последователност

Фигура 7 - Процес на присъединяване на курс с вече съществуващи задания - диаграма на последователност

Фигура 8 - Начална страница на СУКАО уеб интерфейса

Фигура 9 - Страница с всички курсове в уеб интерфейса

Фигура 10 - Страница с всички задания в уеб интерфейса

Фигура 11 - Страница с предадените решения в уеб интерфейса

Фигура 12 - Страница с резултат от предадено решение в уеб интерфейса

IV. Списък на таблиците

Таблица 1 - Сравнение на подобни системи

Таблица 2 - Сравнение на ACID и BASE

1. Увод

Обратната връзка за функционалната коректност, ефективност и придържане към конвенции за стил и добри практики е ключова за изграждането на знания и умения в курс по програмиране. Регулярната обратна връзка и особено възможността студентите сами да проверяват и подобряват итеративно решенията си, са възможни единствено чрез автоматизиране на процеса по тестване и оценяване. С увеличаването на броя практически курсове по програмиране и броя студенти през изминалите години, тези практики стават все по-наложителни за подпомагане на преподавателския състав да бъде

по-полезен и да предоставя персонална помощ за разрешаване на по-сложни казуси със спестеното време.

Целта на дипломната работа е да се проектира и реализира разпределена система за управление на курсове по програмиране с автоматично оценяване на решения.

Структура на дипломната работа

В глава **2. Обзор на подобни системи** ще разгледаме няколко приложения, които покриват изискванията ни до някаква степен. След това ще преминем през техните силни страни и ще направим обзор на системите в действие. Най-накрая ще ги сравним и ще съпоставим плюсовете и минусите им, завършвайки с обобщение защо те няма да ни свършват работа.

В глава **3. Проектиране на системата** ще разгледаме функционалните и нефункционалните изискванията на системата и как ще бъдат изпълнени и предоставени.

В глава **4. Реализиране на системата** ще разгледаме конкретните технологии, с които ще бъде имплементирана системата и защо именно те са избрани. Ще видим също и основните API-та, REST ресурсите и обектите, които се връщат, методите за удостоверяване и как може един потребител да я консумира.

В глава **5. Експерименти и анализ на резултатите** ще разгледаме различни тестове, които са били приложени на системата, за да се симулира реална работна среда и ще направим разбор на резултатите.

В последната глава **6. Заключение** ще направим обобщение, ще видим как системата ще влезе в употреба и ще разгледаме възможни подобрения и бъдещо развитие на системата.

Дефиниции

СУКАО/SUKAO - ще наричаме разработваната Система за Управление на Курсове с Автоматично Оценяване - Methodical and efficient never afraid of hard work.

2. Обзор на подобни системи

Подобни системи

В тази глава ще разгледаме едни от най-популярните приложения и уеб услуги, които покриват възможно най-много от изискванията ни. Първо, нека дефинираме кои са тези най-важни за нас функционални и нефункционални изисквания, които ще действат като критерии за сравнение.

1. Системата трябва да ни предоставя възможността да управляваме много на брой курсове (стотици, хиляди, а може и повече, за момента не се интересуваме от конкретен брой).
2. Всеки курс трябва да може да съдържа голямо количество задания (десетки, това отново е приблизително число).
3. Системата трябва да предоставя лесен и интуитивен начин за работа с предадените решения.
4. Системата трябва да дава възможност на преподавателския състав да проверява и оценява решенията по интуитивен начин.
5. Качването на решенията не трябва да е като архив.
6. Системата трябва да може да оценява решенията на студентите автоматично.
7. Написаните тестове от преподавателския състав не трябва да са видими за студентите.
8. Системата трябва да може да улавя признаци на плагиатство у решенията на студентите.
9. Системата трябва да е гъвкава в отношение на технологиите, които се преподават в курса. Целта е да намерим една система, която може да се

използва за всички ситуации, удобна за всички преподаватели и в бъдеще, позната на студентите.

10. Системата трябва да дава свобода на лекторите сами да управляват курсовете си без намеса на администратор.
11. Системата трябва да е възможно най-достъпна финансово както за преподавателите, така и за студентите.
12. Системата трябва да е достъпна 24/7.
13. Системата трябва да е устойчива.
14. Системата трябва да е лесна за използване.
15. Системата трябва да е надеждна.

След като сме въвели тази основа, може да преминем към приложенията.

За този анализ са подбрани както системи, които са били в експлоатация дълго време и са се доказали като едни от най-добрите за времето си, така и новонавлезли системи, използващи модерни и иновативни подходи и технологии, а именно:

- Web-CAT
- codePost
- GitHub Classroom

Web-CAT

Web-CAT е гъвкава и приспособима система за оценяване, предназначена да обработва задания по програмиране.

Web-CAT е програма, която върви на сървър и предоставя възможностите си чрез уеб интерфейс. Всички дейности, свързани с решенията на заданията, обратната връзка, разглеждане на резултатите и оценяване, се извършват в уеб браузър. Всички административни дейности на инструкторите свързани с курсовете, заданията и оценяването също се извършват в уеб браузър, дори системните административни дейности се извършват по този начин.

Преди създаването на Web-CAT са съществували и други системи за автоматично оценяване, но те обикновено са се фокусирали над определяне

дали изходният код на студента произвежда желания резултат. Това са така наречените Online Judges. Web-CAT оригинално е била проектирана като автоматична система за оценяване с общо предназначение, но преди да се завърши първоначалната версия, нейните автори са решили, че искат по-скоро да поддържат дейности по тестване на софтуера на студентите, отколкото да оценяват работата на студентите чрез просто сравнение на резултатите.

Основни силни страни

Сигурност (Security)

На първо място, системата е проектирана да поддържа сигурна работа. Нейният потребителски модел включва подход за удостоверяване, базиран на приставки с отворен API, така че администраторът може да избере една от няколко вградени стратегии за удостоверяване или дори да предостави персонализирана. Услугите на Web-CAT са предпазени едновременно от специфични права на ниво потребител и от система за контрол на достъпа на ниво роля. Разпознаването на злонамерени студентски програми и предпазването от тях идва по подразбиране. Цялостността на данните се поддържа от политики за сигурност в системата и от услугите предоставени от релационна база данни.

Преносимост (Portability)

Web-CAT е приложение написано на Java, което придава висока степен на преносимост. То може да бъде пакетирано и разпространявано като Java Servlet приложение, което ще работи под всеки съвместим сървлет контейнер като Apache Tomcat. Когато е пакетиран като сървлет, Web-CAT може да се разпространява като .war файл (сервлет уеб архив), който включва всички необходими зависимости.

Разтегливост (Extensibility)

Може би най-голямата сила на Web-CAT е вродената гъвкавост и разширяемост, вградени в неговата архитектура. Приложението е проектирано с напълно plug-in базирана архитектура. Основни функционалности могат да

бъдат добавени без промяна на код в съществуващата система, а всички съществуващи възможности на Web-CAT се реализират в няколко плъгина. Приложението е напълно неутрално по отношение на езика - може да се използва за оценяване на решения написани на Java, C++, Prolog и други.

Ръчно оценяване

Web-CAT се справя с всички автоматизирани задачи, които инструкторът иска да се извършват, но приложението също има вградена поддръжка за ръчно оценяване. Преподавателският състав може да пише коментари и предложения на студентите за техните решения, може да добавя или премахва точки директно в HTML изгледа на изходния код на студента по WYSIWYG начин. Студентите се уведомяват автоматично по имейл, когато ръчното оценяване на тяхното решение е завършено, за да могат да го разгледат.

~(Edwards, Stephen. "What is Web-CAT?")

Опит със системата

След разглеждане на документацията на Web-CAT виждаме защо тя е една от най-използваните системи от рода си. Системата е доста близко до това, което търсим, но за да направим цялостен и изчерпателен анализ, трябва също да съобразим как работи в действителност, за да потвърдим, че се предоставят всички тези функционалности.

След употребяването ѝ 8 месеца (4 като студент и 4 като асистент) като кратко обобщение може да се каже, че системата изпълнява голяма част от нужните функционалности, но определено може да се желае доста повече.

Първо, започвайки с това как изобщо един лектор може да се сдобие с приложението, за да го използва за своя курс по програмиране. Тъй като това не е уеб услуга, преподавателят ще трябва сам да се погрижи за поддържането и предоставянето му на други потребители. Алтернативно би могло да има една инстанция на Web-CAT за целия факултет или университет, така може да има системен администратор, който да е посветен на тази дейност. Хостването

на приложението няма да бъде безплатно, а и освен административните дейности в самата система, ще има и усилия за поддържане на самото приложение работещо 24/7 дори при голямо натоварване.

Приемайки, че вече има налична система за използване, следваща стъпка би била регистрация на потребители - администратори, лектори, асистенти и студенти. Това е ръчна дейност на системния администратор. Този процес е доста неприятен, времеемък и склонен към грешки.

При създаване на курсове и задания не може да се пишат дълги описания или условия, така че тази дейност трябва да се извършва в друг инструмент. След създаване на задание, потребителите виждат само името му, което може да се кликне. Това води към друга страница, в която студентите могат да си качат решението като .zip архив. Използването на zip архиви за контрол на версиите на изходен код е много неудобно. Докато решават задачата, студентите постоянно правят промени по решенията си, оправят бъгове и качват нова версия на решението си, което е напълно нормален процес на работа. За една задача те могат да имат много версии, което става трудно за следене. Те трябва да знаят коя версия в кой архив им се намира, какви точно промени са направени в определена версия и още много други усложнения. След като си качат архива с финалната версия в системата не излиза ясно и видимо кода, който се съдържа там, и за да си сигурен, че си качил правилната версия трябва да си изтеглиш решението, да го разархивираш и да го разгледаш цялостно, което е голямо неудобство. В допълнение, от гледна точка на преподавателския състав, когато някой студент иска помощ, неговото решение отново трябва да бъде свалено и разглеждано локално.

При проверка на решения се вижда така наречения WYSIWYG интерфейс, който звучи много удобен и интуитивен, но на практика пълен с бъгове, доста досаден и труден за ползване. Доста неконсистентно, но често явление е да си на един клик от страница за срыв.

Срещат се и сериозни технически ограничения. Примерно при задания свързани с упражняване и тестване на знанията на студентите относно многонишково програмиране, се изпитват затруднения при стартиране на множество нишки.

Накратко, системата може да свърши работа, но си личи, че не е изпитана, не използва модерни подходи и технологии и определено звучи по-добре отколкото изглежда.

codePost

CodePost е система за автоматично оценяване на решения на софтуерни задачи, която също дава възможността за коментиране на кода на студентите и предоставя различни инструменти за даване на обратна връзка. Тя, за разлика от Web-CAT, е уеб услуга и е напълно безплатна за лектори в университети. CodePost се използва в момента от Бостънският университет, Принстънският университет, Университетът на Айова, Университетът Корнел и други.

Това обаче не е просто още един инструмент за оценяване. Процесът за даване на обратна връзка е изграден от нулата с една конкретна цел в предвид и тя е да направи преподавателите отлични в това което правят, а именно да обучават следващото поколение програмисти. Системата е бърза и лесна за използване.

Мисията на codePost е да помогне на преподавателите по компютърни науки да предоставят изключителна обратна връзка на студентите относно техните положени усилия. Оригиналната система е разработена в Принстънския университет от екип от студенти. Тогава всички изпити там са били провеждани и преглеждани изцяло ръчно с лист и химикал. Целта е била този процес да се дигитализира.

Основни силни страни

Коментиране на код

Една от основните цели на codePost е да въведе аотиране на кода с минимални усилия. Могат да се добавят лесни за четене коментари, които могат да са както персонализирани, така и взети от стандартизирани рубрики. Поддържа се също и коментиране на Jupyter тетрадки и .pdf документи.

Автоматично оценяване

Друга важна мисия за създателите на codePost е да предоставят функционалност за лесно писане на тестове и изпълнението им срещу студентските решения. Идеята е ефективно да могат да се откриват грешките дори в курсове с много голям брой студенти. Преподавателският състав може да се възползва от обикновени тестове, които не изискват никакво писане на код. Могат също да се пишат и по-гъвкави тестови сценарии използвайки кратки скриптове. Всички тестове се изпълняват на codePost сървъри като се поддържат основните езици за програмиране.

CodePost е доста гъвкав и мощен инструмент, който предоставя много различни варианти за осъществяването на самото автоматично оценяване. Тестовите могат да се изпълняват директно на сървъри, предоставени от системата, но те също могат да се изпълняват и на сървъри предоставени от потребителя. Резултатите от тестовите пък могат да се показват или като обекти или като прости текстови файлове. При извеждането на резултатите автоматично се добавят или извеждат точки за всеки тест, може да се добавят обяснения и също, ако тестът е пропаднал, да се показват подсказки. Преподавателите, не винаги искат да показват резултатите от тестовите на студентите, примерно, ако искат да ги принудят те да се постарят да си тестват самостоятелно решението максимално. Това също е възможно, както и в допълнение може да се изпълняват тестове при качване на решение, за да се предотвратят ситуации, в които решението на студента дори не се компилира. Системата позволява да се посочват лимити на количеството тестове, които могат да се изпълняват. Могат да се задават и изисквания за имената на файловете, които трябва да се качат. Друга позитивна черта е и факта, че тестовите се изпълняват изолирано, предотвратявайки един неуспешен тест да повлияе на другите.

Отворена и оперативно съвместима

Уеб услугата предоставя доста функционалности, но все пак всички те са предимно базови и очаквани, в контекста на системи за автоматично оценяване. Потребителите обаче могат да пишат скриптове, в които да комуникират с codePost API-то, което позволява интегриране с множество системи като Moss, GitHub, Repl.it, CodePen и други.

~(codePost)

Опит със системата

Разглеждайки документацията добиваме усещане за една доста сериозна и професионална система, която вече се е утвърдила в много от най-добрите университети в света. Модерният и информативен сайт с документацията показва очакванията ни и за самата уеб услуга.

След използване на системата 4 месеца като асистент, основните придобити впечатления са, че това е една доста съвременна система, която използва най-новите технологии на пазара, за да предостави максимална сигурност, бързодействие и удобство. В сравнение с Web-CAT характеристиките, които изпъкват най-много, определено са липсата на грешки и интуитивността при работа с уеб интерфейсът. Тези неща обаче далеч не значат, че codePost е перфектен продукт. Местата, в които Web-CAT се проваля, действително са подобрени тук, но пък и обратното важи за повече от една функционалност.

Първото нещо, което трябва се направи от преподавател, който иска да използва системата, е да се регистрира, което става лесно - с имейл и парола. След това той трябва да се свърже с администратор, който поддържа системата, и да докаже, че наистина е лектор в някой университет, за да може да му се дадат учителски права. Това не е автоматичен процес и никога не се знае колко време би отнел, но все пак е разбираем за сигурността на системата. Чак след това ще може да създава курсове и задания.

След сдобиването с учителски профил, лекторът може да създаде курс и ръчно трябва да добави всички студенти в него и да даде асистентски права. Интерфейсът е много прост, минималистичен и красив. Това прави работата доста приятна. Има малко на брой изгледи, което намаля шанса за объркване или "изгубване". В страничната лента на сайта има връзка към документацията и кратки видеа, които показват как се работи със сайта. Лентата присъства във всеки изглед, което е изключително удобно. С по един клик може да се

разгледат всички курсове, всички задания в даден курс и предадените решения.

Студентите могат да качат своето решение като архив. Това е едно от основните неудобства. Има алтернатива да се използва GitHub за съхраняване на кода, но това е функционалност, която не идва по подразбиране, а трябва преподавателя да напише скрипт за интеграция с GitHub, който ще осъществи тегленето на изходния код на студентите. Там обаче ще се появят допълнителни проблеми свързани с видимостта на хранилищата на студентите. Те или ще трябва да са публични и лесно видими за всички или трябва да са лични, но тогава пак ще има допълнителни затруднения с изтеглянето на кода. С решения в .zip архиви идват същите проблеми, които се срещат и при Web-CAT.

Преминавайки към оценяването на решения и писането на обратна връзка за студентите, виждаме може би най-добрата функционалност на codePost, а именно уеб редактора на код, чрез който се пишат директно коментари върху кода на студента. Друга функционалност, която спестява много време и усилия за синхронизиране на преподавателския състав са рубриките от съобщения за обратна връзка. Тя дава възможност да се създадат рубрики представляващи различни видове грешки, примерно грешки свързани с бързодействие или грешки свързани с чист код и други. След това могат да се попълват тези рубрики с конкретни съобщения и точките, които ще бъдат отнети при това нарушение. Тези съобщения са споделени между всички оценяващи заданието. Един минус на този изглед е, че не се дава възможност на студентите да теглят решенията си, а само да ги разглеждат.

Основната негативна характеристика, освен работата със .zip архиви, е фактът, че codePost е уеб услуга и при срыв на системата преподавателите нямат какво да направят, освен да чакат. Това е изключително неудобно, ако се случи в критичен момент като край на срок за предаване на някакъв проект или домашно. Друго затруднение, с което ще се сблъскат лекторите, е извличането на резултатите от тестовите, които са се изпълнили срещу решенията на студентите. Ако се използват тестови пакети, които са по-сложни от семпли входно-изходни тестове, то за самото извличане и показване на резултатите ще трябва да се напише специфичен скрипт за тази цел. Всички тези недостатъци са помислени и избегнати в Web-CAT.

GitHub Classroom

GitHub Classroom е доста подобна система на вече разгледаните системи - Web-CAT и codePost. При нея целта също е да автоматизира курсовете и да позволи на преподавателите да се концентрират върху обучението. Тя позволява лесно управление и организиране на курсове. Също така предоставя възможност за следене и управление на задания в уеб интерфейса, автоматично оценяване на решения и помагане на студентите, когато срещнат някакъв проблем и не могат сами да стигнат до добро решение. Интересното и очевидно качество на GitHub Classroom е, че тя се базира на GitHub - популярен и стандартен инструмент, който се използва от много софтуерни разработчици ежедневно. Git и GitHub автоматично решават всички проблеми, свързани с предаването на решения в .zip формат, тъй като те са създадени точно с тази цел, а именно контрол на версиите на изходен код. Ученето и работата с тази система, ще даде безценен опит на студентите и ще ги научи на добри практики за работния процес, заради нейната същност.

Основни силни страни

Безболезнено оценяване

GitHub спестява време на преподавателския състав като използва автоматично тестване за оценяване на задания. Тестовите се пускат при всяко качване на код и студентите могат веднага да видят как са се справили, позволявайки им бързо да направят нужните промени в кода си, за да стигнат до решението. Преподавателят има специален изглед, в който лесно вижда как се справя всеки студент в курса в даден момент, колко от тестовите минават успешно за всеки един от тях и дали изобщо са качвали някакъв код.

Даване на ценна обратна връзка

Преподавателите имат правата да разглеждат изцяло GitHub хранилищата на студентите, и не само това, те дори са админи в тях. Това означава, че могат да виждат изходния код, история на качването на решения от студента и различни

графики, които показват цялостната активност в това хранилище. Учителите могат да изискват конкретни промени по кода, да оставят общи коментари и дори да дават обратна връзка ред по ред.

Огромна видимост върху студентската работа

GitHub Classroom показва ясен сигнал, когато учениците срещнат проблем, за да може преподавателите да се отзоват на помощ. Системата също предоставя функционалността да се създават както индивидуални задания, така и групови, което е разлика в сравнение с Web-CAT и codePost, където има само индивидуални. Историята към всяко хранилище прави приноса на всеки един от групата пределно ясен. Студенти, които показват извънредно висока активност определено ще бъдат оценени, както и тези, които не са допринесли много за решението на заданието няма да останат незабелязани.

Допълнителни характеристики

Системата, бидейки една от най-популярните инструменти за разработване на софтуер и имайки над 73 милиона потребителя (към март 2022г.), няма никакви проблеми с мащабирането и поддържането на стотици студенти в един курс. Преподавателите могат спокойно да оставят автоматичното разпределяне на задания и автоматичното тестване на решенията да свършат тежката работа за тях.

GitHub Classroom също прави създаването на задания със стартов код и разпределянето им до студентите лекота.

С фокус над честността и почтеността, системата позволява на лекторите да направят хранилищата лични или да ги оставят публични, по тяхна преценка.

~ (GitHub)

Използване на системата

GitHub Classroom е уеб услуга, с която се работи доста лесно. При наличието на опит в GitHub, учителят вече е наясно с над 50% от функционалностите, менютата и бутончетата, с които той ще трябва да борави. Системата се базира на GitHub организации, хранилища, задаване на права в тях и работни процеси,

които са сравнително нови за GitHub, но са нещо доста познато като цяло в сферата на софтуерното инженерство.

За да започне да използва системата, преподавателят трябва да посети уеб страницата на GitHub Classroom, където ще му бъде поискано да се удостовери като за тази цел може да използва вече съществуващ GitHub акаунт или да си направи нов. След влизане в системата, излиза изглед, в който има бутон за създаване на нова класна стая. Преди създаването на нова стая, първо трябва да се направи нова организация, това представлява технически акаунт, чиято цел е да групира хранилища с една и съща цел и предназначение. Тази организация ще групира студентските хранилища като съответно тяхната обща черта ще е това, че принадлежат на един и същи курс или използвайки GitHub термините - класна стая. Преди да премине към създаването на класната стая, учителят има възможност да покани хора в организацията и евентуално да ги направи администратори, така те ще могат да му помагат с управлението на курсовете. Най-накрая учителят може да продължи със създаването на класната стая. Тази операция започва с избирането на асистенти, което не е финално, винаги може да се добавят или премахват в по-късен етап. За да се завърши процеса с присъединяването към организацията или класната стая, администраторите и асистентите трябва да посетят специален линк, който трябва да им бъде разпространен от лектора.

Следваща стъпка би била да се добавят студенти към класната стая. GitHub предоставя множество интеграции с външни системи за управление на обучението, така че това може да стане по много различни начини. Някои от тях са Google Classroom, Canvas, Moodle, Sakai както и други подобни. По-прост начин за добавяне на студентите е, чрез обикновен текстов файл, като примерно може да се напишат факултетните номера на студентите (или друг идентификатор като 3 имена или имейл) по 1 на ред. За завършването на този процес, студентите трябва да влязат в системата и после в класната стая, като тогава трябва да изберат към кой факултетен номер да се свържат. Тук се появява и проблем, защото всеки студент може да се свърже, с който и да е идентификатор и това оплитане трябва да се решава ръчно.

Последната функционалност, която завършва целия процес на провеждане на курс по програмиране с автоматично оценяване, е създаването на заданията. Това е процес от 3 стъпки, който включва задаване на име, крайна дата, определяне дали заданието ще е индивидуално или групово и също задаване на видимост на хранилищата. Втората стъпка е да се зададе шаблонно хранилище, което съдържа изходен код, който ще се репликира автоматично в хранилището на всеки студент. Тази стъпка не е задължителна и ако бъде пропусната, хранилищата ще са празни по начало. Последната трета стъпка е да се добавят тестовите, които ще се изпълняват срещу решенията на студентите. Те могат да са написани на какъвто и да е език. Тук се натъкваме на друг огромен проблем, за да се изпълняват тези тестове като работен процес на GitHub, те трябва да са репликирани във всяко хранилище, тоест студентите ще могат както да ги гледат и да нагласят решенията си по тях без да мислят изчерпателно за проблема, така и директно да ги редактират, което напълно обезсмисля автоматичното тестване, защото след приключването на оценяването и крайния срок на заданието, ще трябва преподавателският екип да провери хранилището на всеки студент за злонамерени активности.



Накратко, GitHub Classroom, точно както предните 2 системи, покрива голяма част от изискваните функционалности, като се справя с доста от проблемите по по-иновативен и впечатляващ начин, но и тази система има 1-2 големи гореспоменати недостатъци, които бихме желали да избегнем.

Сравнителен анализ

	Web-CAT	codePost	GitHub Classroom
Управление на курсове	Да	Да	Да
Управление на задания	Да	Да	Да
Лесна работа с предадените	Не	Да	Да

решения			
Проверка и оценяване на решенията	Работи, но има доста проблеми	Да	Да
Качване на решения не като архив	Не	Не идва по подразбиране, трябва да се пишат специални скриптове, които крият допълнителни трудности	Да
Автоматично оценяване на решенията	Да	Да	Да
Скрити тестове от студентите	Да	Да	Не
Вградена интеграция със система за плагиатство	Не	Не	Не
Работа с много езици	Да	Да	Да
Цена за ползване	Трябва да се плаща за хостинг	Безплатна	Безплатна, с опция за допълнителни платени функционалности
Достъпност	Зависи от хостинга	Висока	Висока
Устойчивост	Не много висока	Висока	Висока
Леснота за ползване	Не е много висока	Висока	Висока
Надеждност	Не много висока	Средна	Висока

Легенда:

-  - Предоставя функционалността
-  - Не удовлетворява напълно изискването

 - Не предоставя функционалността

(Таблица 1 - Сравнение на подобни системи)

Обобщение

Както виждаме в таблица 1, зеленият цвят преобладава при codePost и GitHub Classroom. Не е изненадващо, че най-старата система има най-много проблеми или изобщо липсващи възможности. Най-основните функционалности, а именно управление на множество курсове, задания, автоматично оценяване на решения и поддръжка на работа с много езици са имплементирани и в трите инструмента. Това са качества, които непременно искаме от нашата система. От тук нататък обаче таблицата става по-разнообразна. Лесна работа с предадените решения може да осъществим само codePost и GitHub Classroom. Проверка и оценяване на решенията на студентите може да извършим с всички системи, имайки в предвид че ще се сблъскаме със затруднения при използването на Web-CAT. Качване на решения не във формат на архив може да извършим единствено и само с GitHub Classroom. Това е безценно качество, от което задължително бихме искали да се възползваме. От друга страна, скритите тестове от студентите е значителен недостатък, който отново се среща само в GitHub Classroom. Нито един от тези инструменти обаче, не се интегрира с външни системи против плагиатство. Принуждавайки учителите да извършват тази дейност ръчно или да си пишат собствени програми, които да го правят вместо тях. Откъм цена за ползване се срещат 2 тотално различни модела, по които се предлагат тези системи. От една страна имаме просто приложение, които трябва да се хоства от преподавателите, а от друга имаме уеб услуги. И двете си имат предимства и недостатъци и би било чудесно да се предоставят и двете възможности на потребителите. CodePost и GitHub Classroom предоставят много по-добро потребителско изживяване при ползване на уеб интерфейса.

Изследването показва, че нито една от разгледаните системи не е удовлетворяваща. Web-CAT е без съмнение най-далеч от изградената ни визия за перфектна система. Обединение на позитивните страни на codePost и

GitHub Classroom, в допълнение с вградена интеграция с инструмент за плагиатство и разнообразие в начина на консумиране на продукта е търсената от нас комбинация.

3. Проектиране на системата

Разрешаване проблемите на конкурентните системи

След анализиране на подобни системи бихме желали да извлечем най-доброто от тях и да помислим за решения на основните им проблеми. Проблем на една система може да е решен в друга, но има недостатъци, които се припокриват и в трите.

Първият проблем, който срещаме е при Web-CAT, а именно сложната и не толкова интуитивна работа с предадените решения. CodePost и GitHub Classroom решават проблема по различни начини. Начинът, по който GitHub Classroom отстранява неудобството, и който ние бихме желали да интегрираме, е чрез система за контрол на версиите на изходен код - GitHub. Тук има няколко системи, подходящи за нас - GitHub, GitLab и BitBucket. BitBucket е най-непознатата от 3те, около 50 пъти по-неизвестна от GitHub (според брой резултати в Google) и също е платена за големи екипи, каквито ще са курсовете. Така BitBucket бързо отпада от възможностите. GitHub е грубо 10 пъти по-популярна от GitLab и е вероятно повечето студенти вече да имат акаунт и да са наясно със системата.

Приемайки, че GitHub е избраният инструмент за решаване на проблема, остава да се разучи документацията на API-тата му. За да се имитира функционалността на GitHub Classroom ще трябва да има технически акаунт, в който програматично да може:

- да създаваме организации (аналог на курс)
- да създаваме лични хранилища на всеки студент (работни места)
- да се добавят студенти като сътрудници в тези хранилища (за да имат правата да достъпят работните си места)

- да може да се качват файлове в хранилищата на студентите (директории, които групират различните задания и условия на самите задания).

Повечето от изброените изисквания могат да се постигнат с програмните интерфейси на GitHub, с изключение на едно - създаване на организации. Тази функционалност е достъпа само в GitHub Enterprise, което е платената версия на GitHub, предназначена за бизнеси и компании. Друго затруднение, с което може да се сблъскаме е ограничение на броя допустими участници в GitHub организация. Тук удряме на камък, тъй като на пръв поглед няма как да избегнем тези проблеми.

След старателно преразглеждане на алтернативите изскача перфектно решение на конкретните проблеми. GitLab предоставя същите възможности като GitHub, позволява създаването на организации програмно (групи, според терминологията на GitLab) и поради факта, че е безплатна система с отворен код, освен уеб услугата gitlab.com, също се предоставя и като програма, която всеки може да си хостне самостоятелно.

Вторият проблем, който не е толкова значителен и се среща основно в Web-CAT е лекотата на проверката и оценяването на решенията. CodePost решава това, като са си изградили собствен интерфейс, който е изпитан и тестван и изглежда да работи безотказно. GitHub Classroom позволява извършването на тази дейност чрез уеб интерфейса на GitHub. Тъй като ние ще интегрираме GitLab, тази функционалност ще дойде автоматично.

Третото ни изискване е свързано с управлението на версии на кода, а именно да не се качват архиви съдържащи изходния код. То не е покрито от всички системи, но ние ще го решим отново чрез интеграцията си с GitLab.

Следващото функционално изискване е потенциала за скриване на тестовите от студентите. При интегрирането на GitLab ще се създаде един технически акаунт за системата като в него ще се създават групите (крусовете) и хранилищата. След това студентите ще бъдат добавяни в съответните хранилища като членове с права на разработчици. Така системата ще може да работи с кода на всички студенти, тъй като системният акаунт ще е админ в

поверените им хранилища, а в същото време те ще могат да виждат само своя код. Учителят също ще си има собствено хранилище, в което ще се намират и тестовете за заданията.

Последното функционално изискване е да има интеграция със система за плагиатство без да се налага преподавателите да пишат скриптове или да правят каквито и да е настройки. В учителския изглед на всяко задание в уеб интерфейса на системата ще се намира бутон, който ще инициира процеса по откриване на плагиатство. Сървърът ще изтегли решенията на всички студенти от хранилищата в GitLab и ще ги изпрати на MOSS за проверка.

Относно цената за ползване, системата ще е достъпна по подобен модел на GitLab. Ще има уеб услуга, която ще има ограничения, но ще е безплатна и също ще може да се инсталира с няколко прости команди на собствен хардуер, предоставен от потребителя.

Оставащите нефункционални изисквания - достъпност, устойчивост и надеждност, ще бъдат покрити чрез използването на Docker контейнери и оркестрирането им с Kubernetes. Това ще бъде обсъдено в някои от следващите подглави.

Предоставяне на функционалностите, съществуващи в конкурентните системи

След преминаването през недостатъците в Web-CAT, codePost и GitHub Classroom може вече положителните им страни да ни звучат като даденост или като прости възможности, които очакваме да съществуват. Това със сигурност не са лесни за имплементиране функционалности.

Първо имаме управление на курсове. За съхраняването на всички данни, с които ще работи системата ще се използва база от данни. За един курс ще е важно да се пази неговото име, описание, създател и метаданни като дата на създаване и време на последната промяна. Тъй като никое от тези полета не е

уникално за един курс, ще трябва да имаме изрично поле id, което ще е UUID. Освен това, интегрирайки GitLab с нашата система, ние ще имаме 1 към 1 съотношение от курс в нашата система към група в GitLab. Поради тази причина ще трябва да се пази идентификатора и името на GitLab групата. Сървър на нашата система ще има връзка с базата и ще извършва CRUD операции. Също така ще има входна точка - приложно-програмен интерфейс, който ще се използва от потребителския интерфейс.

Второто изискване, управление на задания, ще бъде изпълнено по същия начин. В базата ще се пазят почти същите полета, като различията ще са име на заданието в GitLab (името на работните директории, които ще се създадат в хранилищата на студентите) и имейл на автора, който е създал заданието(това може да е и лектор, и някой от асистентите).

Предпоследното изискване е да може автоматично да се оценяват решенията на студентите. Това ще бъде постигнато чрез допълнителен компонент, който ще представлява сървър с асинхронен програмен интерфейс, чиято дейност ще е да изпълнява задачи, които ще се добавят в опашка и ще се обработват от работници. Изпълнението на тестове ще представлява една такава задача. Стъпките в нея ще са:

1. Да се изтегли изходния код на студента от неговото лично хранилище;
2. Да се изтеглят тестовете от личното хранилище на лектора;
3. Да се компилира кода;
4. Да се изпълнят тестовете и да се принтират резултатите.

Този скрипт ще се изпълнява в специален Docker контейнер, който пък ще е Kubernetes Pod. На контейнерите може да се слагат ограничения върху използваните ресурси (процесорно време и памет). Те са процеси вървящи на дадена система, напълно изолирани от всички други процеси. Така ще може да се наложат много предпазни мерки, за да се предотврати изпълнението на злонамерен код или грешен код, включващ безкраен цикъл, който използва цялата памет.

Без никакви промени по сървърния компонент или компонента отговорен за изпълнението на задачите може да се предостави последната желана

функционалност - работа с много езици. Единственото нужно нещо което трябва да се направи, за да се добави поддръжка на оценяване на код, написан на друг език, е да се напише нов скрипт за компилиране на кода, написан на този език, и да се стартират тестовите. Това позволява разширяването функционалността на системата със само няколко линии код.

Изпълняване на кода във всякаква среда с Docker

Какво е контейнер?

Контейнерът е стандартна единица софтуер, който пакетира код и всички негови зависимости, така че приложението да работи бързо и надеждно независимо от средата, в която се намира. Едно Docker изображение е лек, самостоятелен и изпълним пакет от софтуер, който включва всичко нужно за изпълнението на едно приложение - код, среда за изпълнение, системни инструменти и системни библиотеки.

Контейнерните изображения се превръщат в контейнери при изпълнение. Това се случва чрез Docker Engine - среда за изпълнение на контейнери, която подsigурява, че контейнеризираният софтуер винаги ще работи по един и същи начин независимо от инфраструктурата.

~(Docker)

Предимствата на Docker контейнерите

Изпълнението на приложения в контейнери носи много предимства.

Преносимост

След като веднъж е тествано едно контейнеризирано приложение, то може да се изпълнява на каквато и да е система с Docker и е сигурно, че то ще работи по същия начин както когато е било тествано.

Производителност

Въпреки че виртуалните машини са алтернатива на контейнерите, фактът, че контейнерите не съдържат операционна система (докато виртуалните машини имат), означава, че контейнерите са много по-леки, по-бързи за създаване, за унищожаване и за стартиране.

Изоляция

Докер контейнерите са просто процеси, които са ограничени чрез пространства от имена и други технологии за изолация - промяна на основната директория, смяна на видимостта на процесите и забрана на излизане от тези рамки. Така може да се изключат притесненията, че един контейнер процес ще вземе ресурсите на друг или ще попречи на работата му по какъвто и да е начин.

Мащабируемост

Може бързо да се създават нови контейнери, ако има голям трафик и натоварване на приложението. За тази цел обаче е хубаво да се използват специални инструменти, които много улесняват тази дейност. Те извършват и още много други, свързани с управлението на контейнери.

~(Micro Focus)

Оркестриране на контейнерите с Kubernetes

Какво е Kubernetes?

Kubernetes, или още познат като K8s, е безплатна система с отворен код, създадена за автоматизиране на внедряването, мащабирането и управлението на контейнеризирани приложения. Тя е създадена от Google, като първата версия излиза на 7ми юни 2014 г.и в момента им помага с изпълнението на милиарди контейнери всяка седмица. Идеята е да групира контейнери, които съставляват едно приложение, в логически единици за лесно управление и откриване. Използвайки принципите, които позволяват на Google да поддържат огромен брой контейнери, Kubernetes дава възможността приложенията да се мащабират без нуждата да се увеличава екипът отговорен за операциите на

разработчиците като непрекъсната интеграция и непрекъснато развитие. Независимо дали се използва само за локално тестване или за управляването на глобално предприятие, гъвкавостта на Kubernetes нараства, за да достави приложенията лесно, с постоянство и без грешка, независимо колко сложна е нуждата. Тъй като системата е с отворен код, това дава свободата да се възползваме от локална, хибридна или облачна инфраструктура, което ни позволява без усилие да преместим работното натоварване там където има най-много смисъл и е най-лесно за нас.

Основни характеристики

Основните силни страни на Kubernetes са:

Автоматично пускане на актуализации или връщане назад

Kubernetes постепенно въвежда промени в приложенията или техните конфигурации, като същевременно наблюдава здравето им, за да гарантира, че няма да убие всички техни реплики в един и същ момент, което би означавало време, в което приложението няма да е достъпно. Ако нещо се обърка, Kubernetes ще върне промяната вместо нас. Има различни стратегии за внедряване - постепенно актуализиране, пресъздаване, синьо-зелено внедряване и внедряване "канарче". Пресъздаването е най-простият вариант, който представлява тотално разрушаване на всички реплики със старата версия и пресъздаването им с нова версия. Това не е много добър вариант, тъй като предизвиква период, в който приложението не може да се достъпи. По-добър вариант е постепенното актуализиране - при наличието на няколко реплики на приложението, те се ъпдейтват една по една докато всички не са на новата версия, ако при някоя реплика се появи грешка, то всички се връщат на старата работеща версия. Друг вариант е синьо-зеленото внедряване, което означава, че репликите са разделени на две групи - сини и зелени, едните са на новата, а другите на старата версия и временно те работят заедно за да се подсигури, че новата версия работи безпроблемно. Последният тип внедряване - "канарче" е подобна на синьо-зеленото, но тук групите не са разделени на равен брой реплики, а постепенно репликите с новата версия се увеличават,

като започват от много малък брой - примерно първо само 10% от репликите са на новата версия, после се увеличават на 20% и т.н.

Оркестрация на хранилището

Kubernetes позволява автоматично монтиране на система за съхранение по избор, като може да се използват локални хранилища, публични облачни доставчици като GCP или мрежови системи за съхранение като NFS, iSCSI и други.

Самолечение

Kubernetes автоматично рестартира контейнери, които се провалят, заменя и насрочва наново контейнери, когато възлите, на които са вървели умрат. Той също убива контейнери, които не отговарят на дефинирана от потребителя проверка за състоянието и не рекламира новите контейнери като достъпни, докато те не са готови да обслужват клиенти.

Хоризонтално мащабиране

Kubernetes позволява лесно мащабиране с проста команда, потребителски интерфейс или автоматично въз основа на метрики, следящи използването на процесора и паметта.

Управление на конфигурации

Внедряването и актуализирането на конфигурации на приложения и чувствителни данни нужни за тяхното изпълнение става без почти никакви усилия. Този процес не е обвързан с построяването на контейнерното изображение - при нужда от малка промяна по конфигурацията е нужен само да се рестартира приложението.

~ (The Kubernetes Authors)

Проектиране на базата от данни

Една система за управление на курсове се нуждае от база от данни за съхранение на информацията, която се изпраща от клиентския интерфейс и се обработва от сървъра. Потребителите трябва да могат да се удостоверяват по някакъв начин, което означава, че трябва да се пазят данни за тях. След това те ще трябва да бъдат упълномощени, за да бъдат допуснати до определени функционалности - трябва да се съхранява ролята на учителите, студентите и администраторите. В допълнение, трябва да може да складира информация за курсовете и заданията - имена, описания, създатели и автори и как са свързани те с GitLab алтернативата си. Това е голямо количество данни, с които трябва да може да се работи бързо. Те трябва да останат последователни след изпълнението на заявки с различните обекти и самото хранилище трябва да е издръжливо и устойчиво на грешки.

При избор на инструмент за съхранение, надеждно решение, което повечето системи са ползвали през годините са SQL базите от данни. Използването на нерелационни бази от данни, още наречени NoSQL, е алтернативно решение, което е характерно за по-модерни и нови системи, създадени в последното десетилетие.

За избирането на най-подходящото средство, авторът е анализирал характеристиките 2те страни.

- NoSQL базите от данни обикновено са разпределени системи, в които няколко машини работят заедно в група. За да предоставят висока наличност и да предотвратят загубата на данни, част от данните се репликират върху тези машини.
- Хоризонталната мащабируемост на NoSQL е огромно предимство. Този тип системи позволяват на потребителите динамично да добавят нови възли без нарушаване на достъпността. Няма горна граница на броя машини, които могат да се добавят.
- Огромно количество данни могат да бъдат обработени лесно и бързо от NoSQL.

- Структурата на данните, които се пазят в тези системи не се дефинира изрично със схема на базата както се прави при SQL инструментите. Поради тази причина клиентите могат да съхраняват данните както си поискат, без да зависят от предефинирана схема/структура.
- NoSQL може да използва нерелационни модели на данни, които позволява по-сложни структури.
- Системите за управление на релационни бази поддържат основно SQL, а NoSQL не го правят. Всяка NoSQL система си има свой собствен интерфейс за заявки. През годините са правени опити за унифициране на тези интерфейси.
- CAP теоремата назовава трите основни свойства на системите за споделени данни, които са последователност на данните, наличност на системата и толерантност към мрежовите дялове ~ (Brewer, 2000). Накратко, теоремата показва, че най-много две от тези три свойства могат да бъдат налични в една споделена система. Това твърдение се отнася за SQL базите от данни и е в полза на NoSQL. Въпреки това, повечето от NoSQL хранилищата правят компромис с последователността на данните в полза на наличността и толерантността на дялове. ~ (Pritchett, 2008). Нужни са ACID свойства, ако система или част от система трябва да бъдат последователни и толерантни към разделяне на дялове. Ако наличността и толерантността към разделянето на дялове се предпочитат пред последователността, получената система като NoSQL може да се характеризира с BASE свойства. На повечето NoSQL хранилища им липсват напълно ACID транзакции ~ (Vanroose et. Thilo, 2014). Този компромис е критична точка за обсъждане, когато се анализират принципните концепции на базите данни.

ACID (SQL)	BASE (NoSQL)
Силна последователност	Слаба последователност
Изоляция	Последното писане побеждава
Има транзакции	Управлявана от програма

Здрава и стабилна база	Проста база
Прост код (SQL)	Сложен код
Достъпна и постоянна	Достъпна и толерантна към разделяне на дялове
Лимитирано мащабиране	Нелимитирано мащабиране
Споделени диск, памет, процеси и др.	Нищо споделено

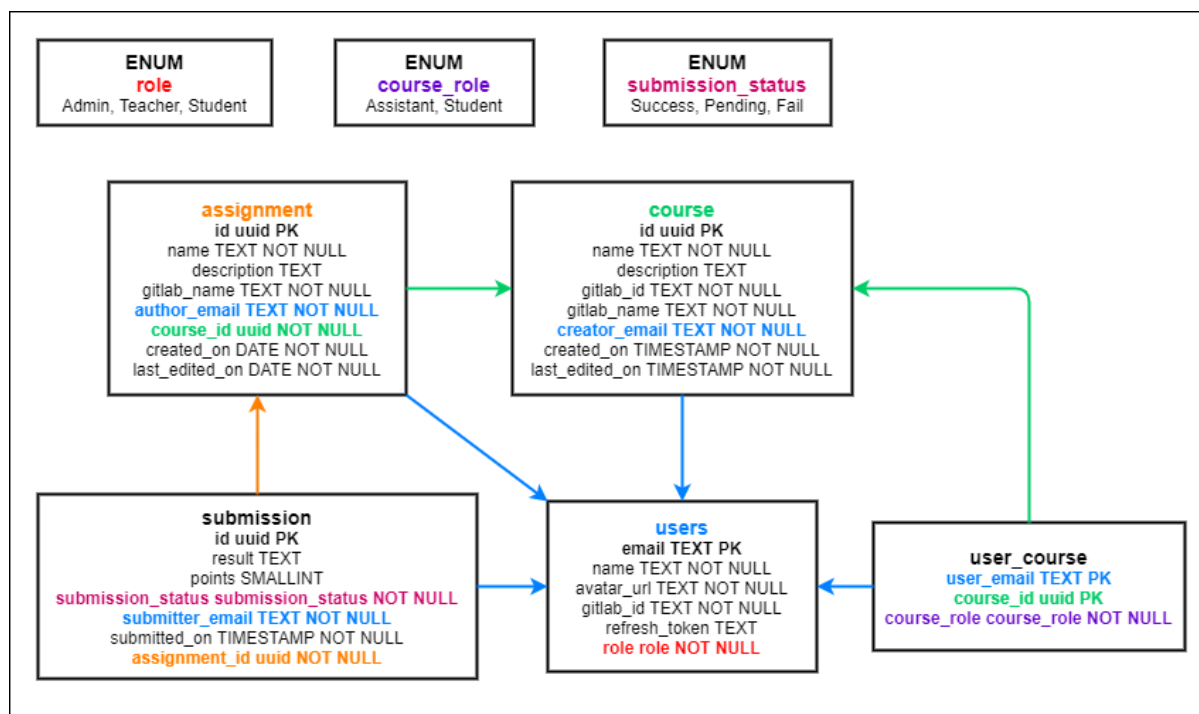
Таблица 2 - Сравнение на ACID и BASE

~(Birgen 4)

Основните характеристики, които са от най-голямо значение за разработваната система за оценяване са:

- Транзакции
- Последователност на данните
- Външни ключове
- Ограничения

Тези черти са присъщи за SQL базите от данни. Една от най-популярните и най-използваните SQL системи, която също е безплатна и с отворен код е PostgreSQL. Именно тя е избрана за разработване на системата за управление на курсове и автоматично оценяване.



Фигура 1 - Диаграма на базата от данни

Диаграмата на базата от данни ни показва всички таблици, полетата, които се пазят в тях, индексите им, първичните и външните ключове.

Първо се виждат три обекта, които не са таблици, а изброени типове, които налагат ограничение на данните, които могат да приемат полетата с този тип. Изброеният тип, който ще представлява ролята на всички потребители в системата се нарича **role** и може да приема стойностите -

Admin(администратор), Teacher(учител) и Student(студент). Типът

submission_status показва, че предадените решения могат да бъдат само в три състояния - Success(успешни), Pending(изчакващи да бъдат тествани) и Fail(провалили се). Последният тип **course_role** може да приема стойностите Assistant(асистент) и Student(студент) и той показва каква е ролята на даден потребител в рамките на някакъв курс, тъй като един потребител може да студент в един курс, но асистент в друг.

Таблиците course(курс), assignment(задание) и submission(предадено решение) имат само един индекс и той е първичният ключ **id**. Уникалното поле в таблицата, в която се пазят данните за потребителите е **email**. Таблицата **user_course** представлява many-to-many(много към много) релация, тъй като един студент може да е в много курсове, но и в един курс може да има много

студенти. В нея уникалното е комбинацията от потребителски имейл и идентификатор на курс.

Една от основните причини за избора на SQL пред NoSQL база от данни е многобройните външни ключове. Както се вижда на диаграмата един submission принадлежи на assignment, което е постигнато с външен ключ **assignment_id** в submission таблицата. Същата релация я има за курс и задание чрез ключа **course_id** в assignment таблицата. Тези три ресурса си имат създател/автор/подател, които са отново ключове към таблицата **users**.

Проектиране на уеб интерфейса

За проектирането на уеб интерфейса, с който ще работят потребителите, авторът е събрал положителните страни на конкурентните системи и е анализирайл бизнес нуждите на приложението.

Съвременните приложения основно се делят на такива с много HTML страници и такива с една HTML страница. По-остарелият начин за правене на приложения е с много страници като неговите основни недостатъци са, че трябва да се правят много на брой статични страници. За зареждането на всяка отделна страница трябва да се прави заявка към сървъра, която може да се забави. В допълнение, с времето се е появила нуждата от динамични уеб приложения, които да си променят съдържанието според ситуацията - за един потребител може да се показва едно, за друг друго или примерно при някакъв онлайн магазин се променят продуктите. Такива приложения все още се използват, в някои специфични ситуации като например за автобиографичен уеб сайт. За изработването на уеб интерфейса на една система за управление на курсове и оценяване на решения, този подход няма да е удачен. Наличието на променящите се данни за курсове, задания, потребители както и желанието за постигане на удобно, приятно и безпроблемно потребителско изживяване, водят до изработването на SPA.

Технологията, избрана за създаване на приложението е React.js, поради огромната популярност и запознатост на автора. Също така предоставя бързо изобразяване на промените по страницата, преизползваеми компоненти и

висока производителност. Относно бъдещото развитие на проекта, голяма част от кода ще може да се преизползва без дублиране при разработването на мобилно приложение, поради наличието на React Native, което е библиотека, базирана на React.js, служеща за създаване на iOS и Android приложения едновременно.

Следвайки примера на codePost и GitHub Classroom, са проектирани 6 основни изгледа:

- Изглед за влизане в системата (единственият достъпен без удостоверяване в системата);
- Изглед с всички курсове, в които членува потребителя (началната страница след вход в системата);
- Изглед с определен курс и заданията в него (достъпен, чрез кликане на някой от курсовете);
- Изглед с определено задание и предадените решения за него от текущия потребител (достъпен, чрез кликане на някое от заданията);
- Административен изглед с абсолютно всички потребители, които са влизали в системата, позволяващ задаването на учителска роля в системата (достъпен само за администратори);
- Административен изглед на даден курс с абсолютно всички негови членове, позволяващ добавянето на студенти и асистенти в него (достъпен само за учители).

Проектиране на API сървър

Приложно-програмният интерфейс на сървърната част на системата трябва да е лесен и прост за ползване. Най-популярният вид архитектура за едно API в днешно време е REST. Това представлява набор от архитектурни ограничения, а не протокол или стандарт. При изпращане на заявка към RESTful API за определен ресурс, сървърът отговаря с едно представяне на състоянието на даден ресурс. Това представяне може да се изпраща в различни формати през

HTTP протокола - JSON, HTML, XML или прост текст. JSON е най-популярният формат, тъй като е независим от езика и е лесно четим за хора и компютри. При комуникация с REST интерфейс, всяка заявка се праща на определен адрес, който включва параметър в пътя, показващ желан ресурс. Има различни видове заявки според HTTP метода - GET заявката служи за вземане на информация за определен ресурс, POST - за създаване, DELETE - за триене и други. В отговора на всяка заявка се включва и код на статуса, който лесно може да покаже дали е била успешна или е възникнала някаква грешка и по-конкретно какъв е бил проблемът - примерно несъществуващ ресурс, забавяне в сървъра и още много. Много важна характеристика на такъв тип уеб услуги е, че те не пазят състояние за заявките. Един потребител изпраща отделните заявки и те нямат никаква връзка една с друга.

Започвайки с удостоверяване в системата, трябва да се установи начин за регистриране и вход в системата. Има много различни подходи, като най-използваните са - с име и парола, с токен, със или без многофакторно удостоверяване. Общата черта, присъща за всички изброени е нуждата да се пазят лични данни за потребителите в базата от данни. Това е много сложен процес. Трябва да се измисли надежден и сигурен начин за съхранение и трябва да се спазват стриктни регламенти и закони като общия регламент относно защитата на данните, приет от Европейския парламент и Съвета на Европейския съюз, в сила от 25 май 2018 г. ~ (Европейски парламент и Съвет на Европейския съюз). Има решения, избягващи всички гореспоменати проблеми и едно от тях е протоколът OAuth. Този протокол е разработен от работната група за интернет инженеринг и позволява защитен делегиран достъп. Той дава възможност на едно приложение, достъп до ресурс, който се контролира от някой друг (краен потребител). Този вид достъп използва токени, които представляват делегирано право на достъп. Приложенията получават достъп, без да трябва да се представят за потребителя, който контролира ресурса ~ (Wallen).

Тъй като се планира системата да има интеграция с GitLab, за да управлява групите, хранилищата и кода на студентите, то тя може да служи също и за управление на акаунтите на потребителите. GitLab се явява доставчик на идентичност, който поддържа OAuth. За интеграцията, системата трябва да

осъществи един от потоците за удостоверяване и оторизация. Основният такъв е потокът на код за оторизация.

При посещение на уеб интерфейса на системата, потребителя се посреща от страница за вход, която не съдържа никакви полета за попълване, а само един бутон. При натискане на този бутон клиентът е пренасочен към страницата за удостоверяване на GitLab, където вече трябва да си попълни личните данни и да даде права на системата за управление на курсове да използва неговия имейл адрес. След съгласие, код за оторизация с кратка валидност, се изпраща на сървъра. В последствие, той може да го обмени за токен за достъп до данните, за които самият потребител е дал позволение. Завършвайки потока, сървърът използва токена и успешно взима имейла на потребителя, считайки го за удостоверен.

След вход в системата, сървърът трябва да предоставя разширени функционалности на потребителите свързани с различните ресурси.

Относно курсовете, трябва да може да се:

- създава курс;
- взима информация за курс;
- взима информация за всички курсове, в които членува един потребител;
- променя името и описанието на курс.

Относно заданията, трябва да може да се:

- създава задание;
- взима информация за задание;
- взима информация за всички задания в курс;
- променя името и описанието на задание;
- трие задание.

Относно предаването на решенията, трябва да може да се:

- предава решение;
- взима информация за предадено решение;
- взима информация за всички предадени решения за определено задание.

Относно административните дейности, трябва да може да се:

- влиза в системата;
- излиза от системата;

- взима информация за текущият удостоверен потребител;
- дават админ права на ниво система;
- дават асистентски права на ниво курс.

Проектиране на компонента за изпълняване на тестове

Проектирайки сървърната част, първата половина от функционалностите на системата за управление на курсове и автоматично оценяване на решения на студентите е покрита. Оставащата половина е частта, която извършва самото оценяване на решенията.

Планирано е да има отделен компонент, който изпълнява тази дейност, следвайки микросервизна архитектура, която е популярна и широко използвана в Docker и Kubernetes средите. Тестването на едно решение е може да е дълъг процес, отнемаш над 30 секунди. Той включва:

1. Изтегляне на решението на студента от GitLab (може да се забави над 5-10 секунди);
2. Изтегляне на тестовете на учителя от GitLab (може да се забави над 5-10 секунди);
3. Компилиране на кода (зависи от размера на проекта, но може да се забави над 5 секунди при малки проекти);
4. Изпълнение на тестовете, анализиране и форматиране на резултатите (също зависи от размера на проекта и може да отнеме над 5 секунди при малки проекти).

Тези минимални забавяния са при оптимални условия, когато няма трафик в системата. Това означава, че при натискане на бутона в уеб интерфейса, който започва този процес, потребителят ще вижда замръзнал екран, без никаква информация, няма да знае какво се случва на фона. След отваряне на страницата отново и отново на слуги, през произволен интервал от време, той рано или късно ще види резултат. Това не е приятно и желано потребителско изживяване. Решението на този проблем е всяка една от тези задачи (оценяване на решения) да бъде асинхронен процес. Така ще може потребителя да види веднага резултат след натискане на бутона, че заявката му е приета, а информацията от самите тестове ще стане достъпна в по-късен

етап. При разработването на този компонент отдаден за изпълнението на асинхронни задачи, ще бъде създадена група от работници. Работниците ще са известен краен брой и ще представляват броя задачи, които могат да се изпълняват едновременно. Ще има опашка от задачи, в която ще се слагат новите задачи, и от която ще се взимат задачи за изпълнение от работниците. Тя също ще е краен брой, за да предотврати пренатоварването на системата и заемане на прекалено много ресурси. При надхвърляне на тази бройка задачите директно ще бъдат отказани.

Инсталиране на системата

Системата за управление на курсове и оценяване на студентски решения е насочена както към индивидуални учители, водещи курсове с не повече от 30 ученика, така и към преподаватели обучаващи в университет или дори едновременно за всички преподаватели от даден университет. Оптималният начин за използване на системата, нейното инсталиране и настройване варира от ситуацията.

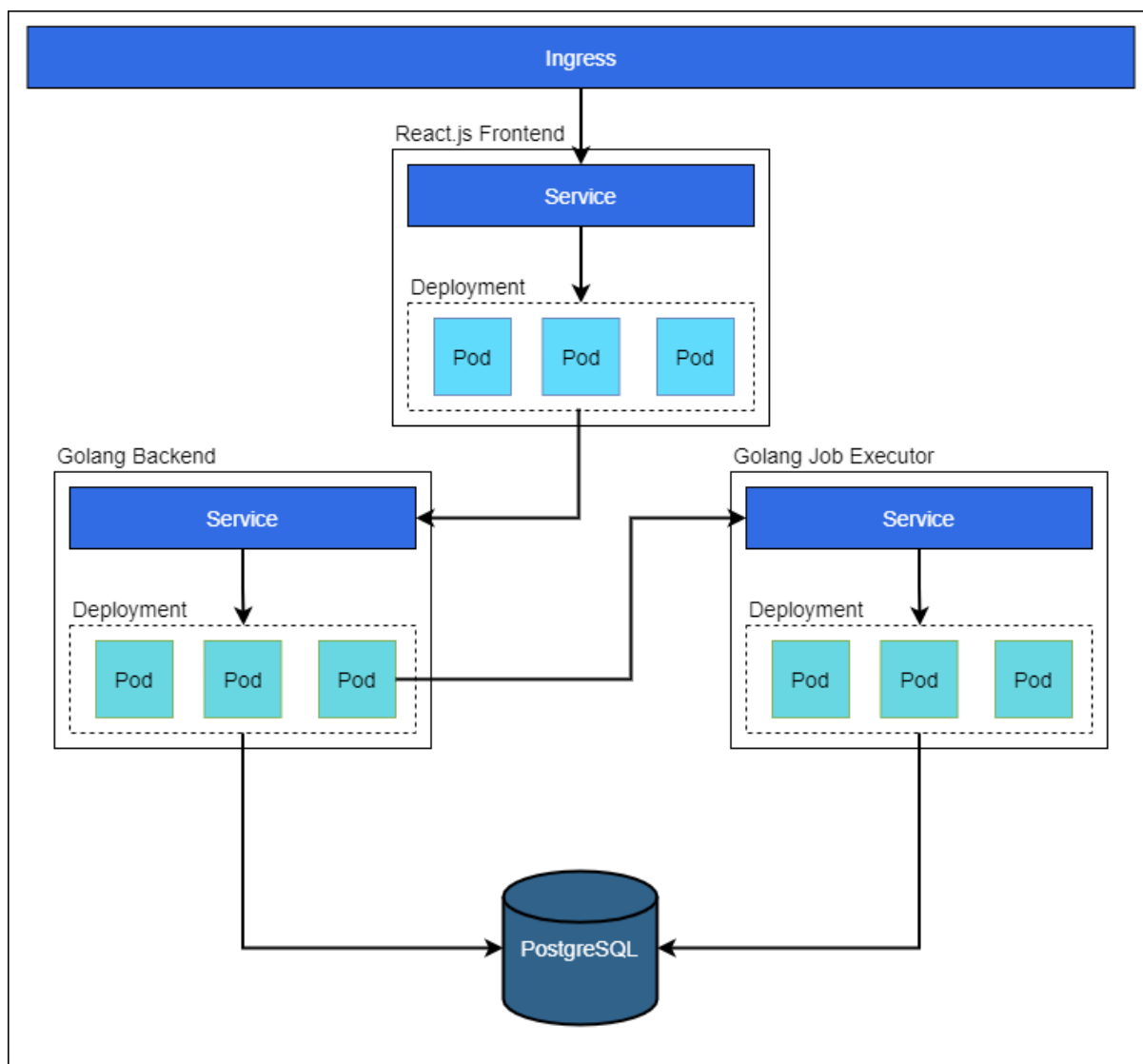
За самостоятелния учител, водещ малък курс, най-лесно би била формата на софтуер като услуга, притежаващ определени щедри ограничения, предотвратяващи пренатоварване и срив на приложението. Така той няма да има нужда да мисли за инсталирането, конфигурирането на системата и още други административни дейности. Той и неговите студенти ще могат бързо и лесно да започнат да я употребяват, заемайки много малко ресурси от цялостната система.

За лектори в университети, по-подходящо може да е програмата да върви на собствен хардуер. В този случай ще е нужен системен администратор, грижещ се за висока наличност на системата. От друга страна, по този начин приложението ще частно и изолирано специално за членовете на курса или университета, за който е предвидено. Така използваните ресурсите ще са напълно независими от алтернативната уеб услуга, а единствените ограничения, които ще съществуват тук, ще зависят изцяло от администратора на системата.

Тези модели са вдъхновени напълно от начините, по които се предлага цялостният продукт GitLab. При него също има безплатна уеб услуга с ограничения - gitlab.com, и отново безплатна програма, която всеки може да инсталира на собствен хардуер, включваща абсолютно всички функционалности.

Нужен е подходящ начин за лесна инсталация с възможност за лесно подаване на конфигурация. Проектирано е системата да върви в Docker контейнери, които да бъдат оркестрирани с Kubernetes. Най-малката градивна единица в Kubernetes се нарича Pod. Тя може да съдържа един или повече контейнери като в случая, всеки един от компонентите на системата ще е в отделен Pod. Тези Pod-ове се менажират от други ресурси наречени Deployment, в които се конфигурират различни черти, включително колко реплики да се създават. Тези Deployment-и без допълнителни настройки, не са достъпни до никого освен до другите ресурси в частната мрежа на Kubernetes. За да се изложат на външния свят са нужни Service-и - друг вид ресурси. В допълнение, всеки компонент от системата - уеб интерфейса, сървър, изпълнителя на задачи и базата от данни се нуждаят от начин за конфигуриране. Това става с ресурсите Secret, използващ се за чувствителни данни, и ConfigMap - за всички останали конфигурации. Има и други ресурси, настройващи мрежовата комуникация с всеки компонент. Всеки един ресурс се създава с един шаблонен файл в .yaml формат. При наличието на 4 компонента и по над 5 различни конфигурационни ресурси за всеки от тях, това става трудно за управление. Трябва някакъв начин, по който всички настройки да са групирани, лесни за промяна и инсталиране. Точно за тази цел е направен и инструментът Helm.

Според авторите, Helm помага за управлението на Kubernetes приложения чрез пакетизирането им в Helm диаграми, които помагат в дефинирането, инсталирането и актуализирането. Характерно за тях е лекотата на промяната на версията, споделянето и публикуването ~ (Helm Authors). Всички от тези качества са нужни за системата и нейното лесно доставяне едновременно като уеб услуга и като лесна за инсталиране програма на хардуера на клиентите.



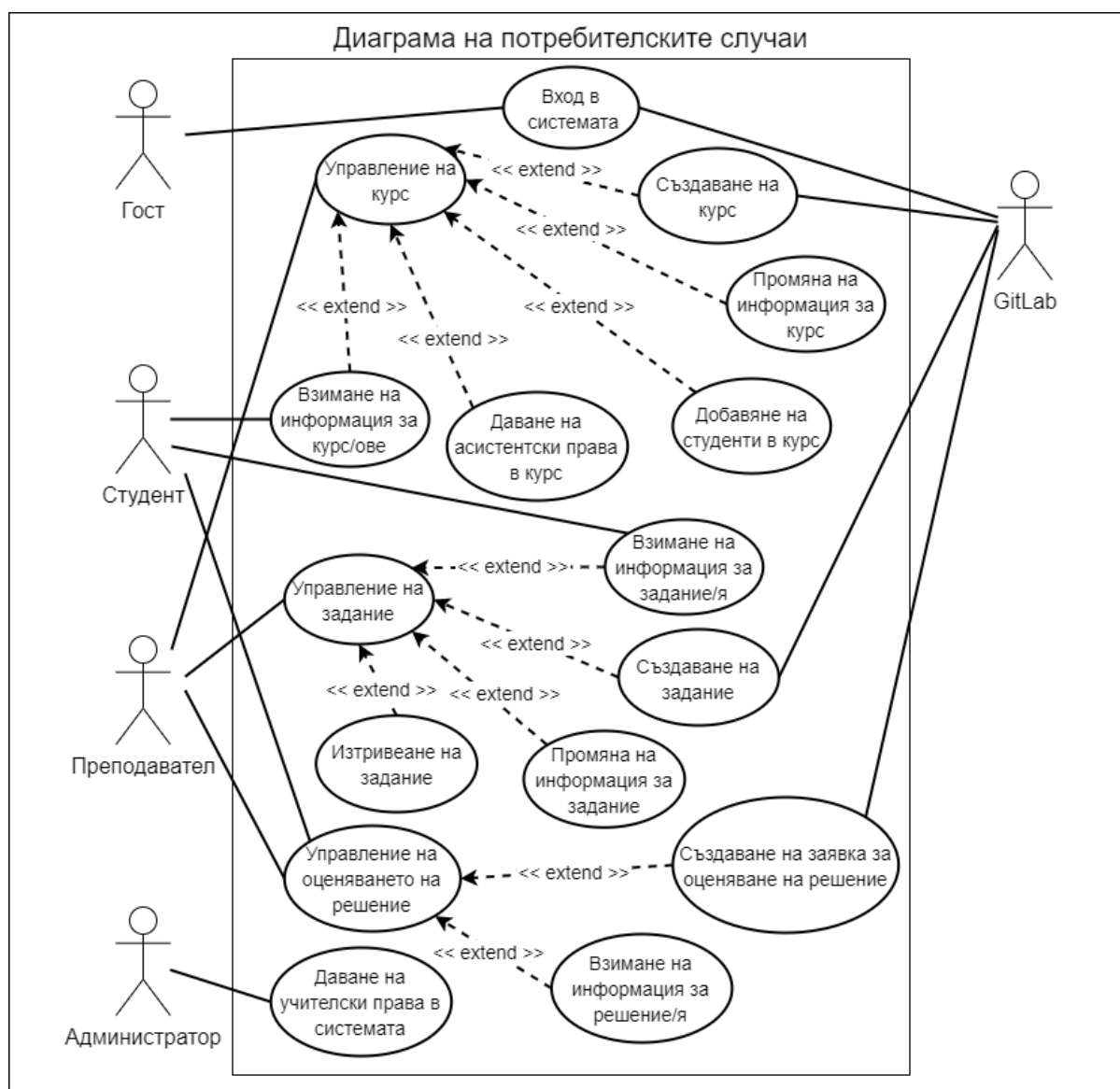
Фигура 2 - Диаграма на Kubernetes архитектурата

Фигура 2 показва архитектурата на системата - всички ресурси и потока на данните между тях. Входната точка на приложението е Ingress, който балансира натоварването и контролира трафика, отиващ към системата. Той прави достъпен до външния свят единствено уеб интерфейс компонента. Самият уеб интерфейс може да прави заявки до сървъра, но не и до компонента - изпълнител на задачи. Сървърът от своя страна може да достъпва базата от данни, за да съхранява нужните данни там и само той може да се обръща до микроуслугата, отговорна за оценяване на решенията, която пък се нуждае от връзка с базата, за да запазва резултатите.

4. Реализиране на системата

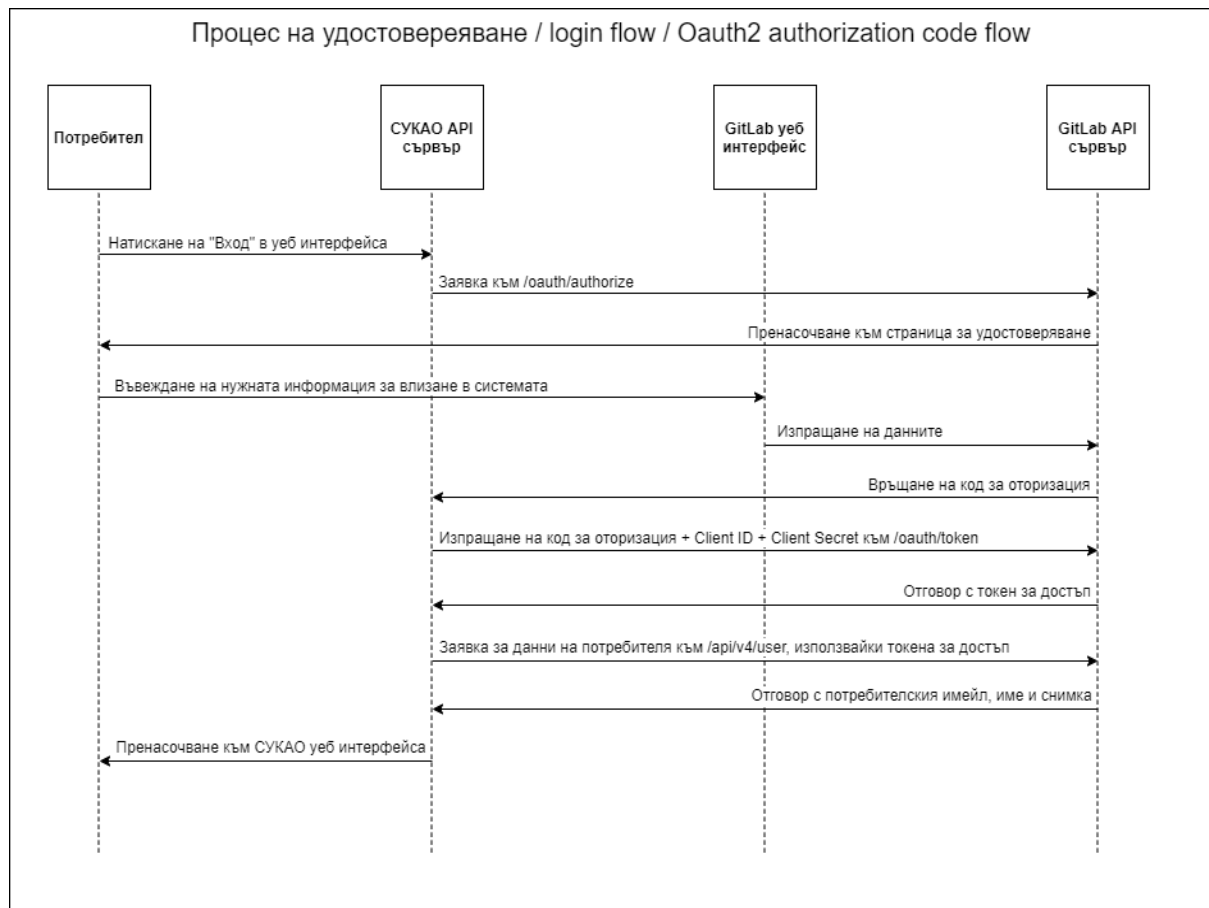
Реализирането на системата следва същите стъпки и се базира на проектирането. В допълнение, ще бъдат разгледани набор от диаграми, както на потребителските случаи, позволяващи по-широк поглед над цялостната система, така и на последователността, показващи в детайли най-сложните процеси, които системните компоненти извършват.

Основни работни потоци



Фигура 3 - Диаграма на потребителските случаи

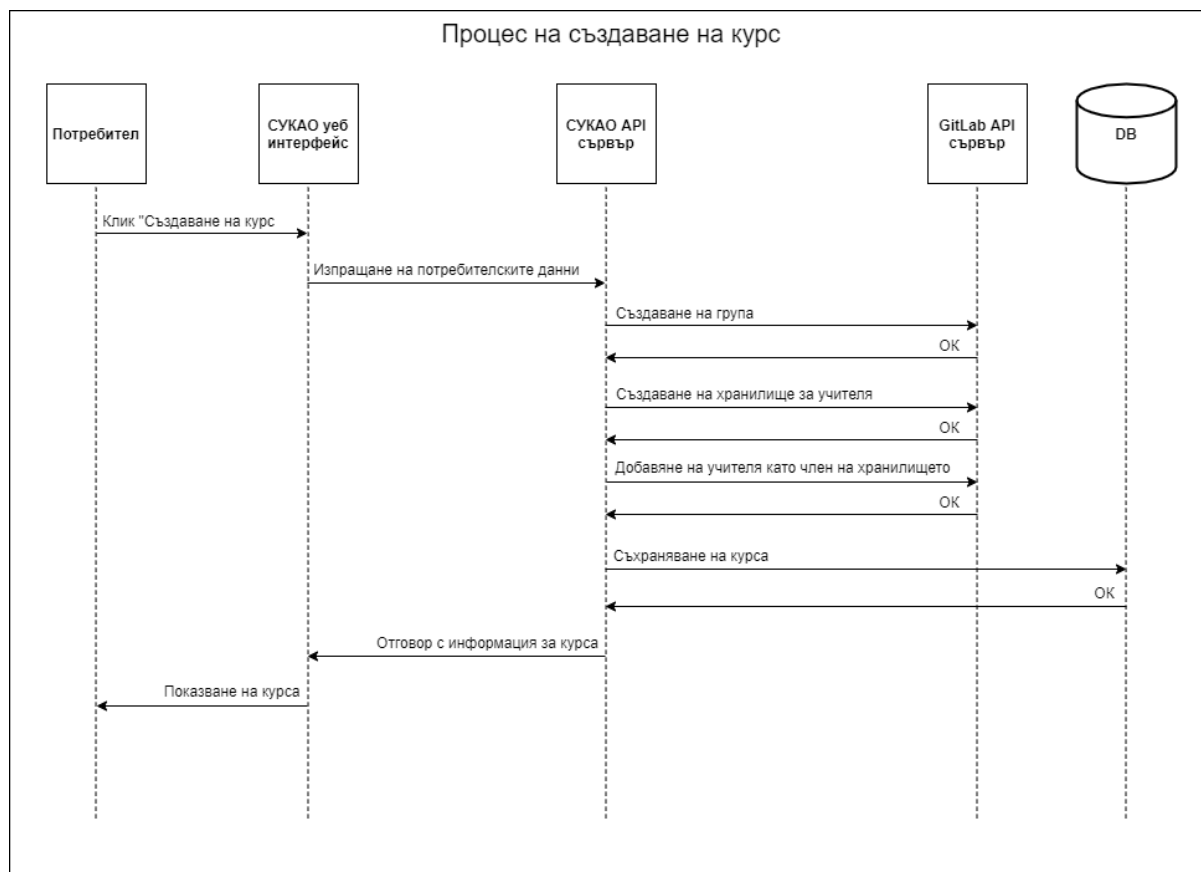
Във фигура 3 се вижда едно обобщение на системата, нейните потребители, наречени актьори, и техните взаимодействия. Реализирани са четири роли - гост, студент, преподавател и администратор. Последният актьор, изобразен на диаграмата е външната система GitLab.



Фигура 4 - Процес на удостоверяване - диаграма на последователност

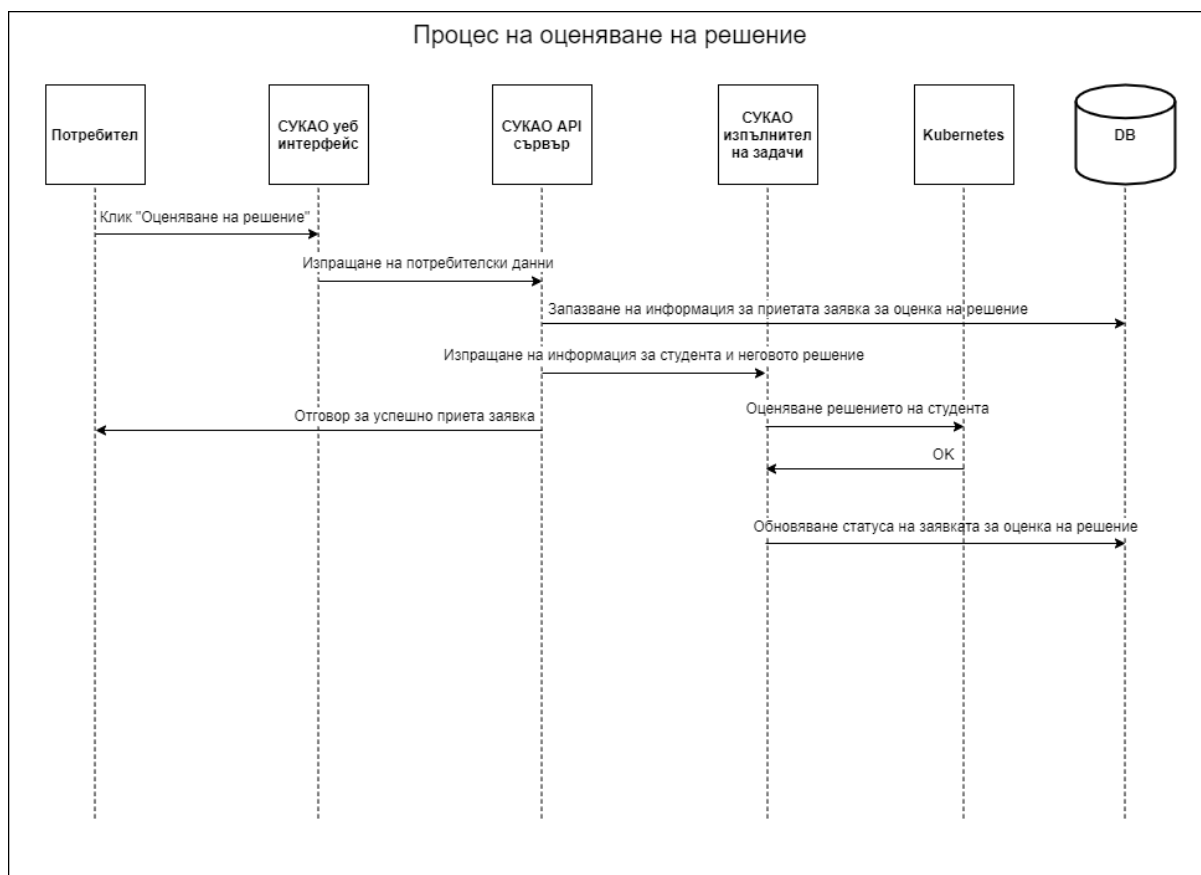
Във фигура 4 се вижда процеса на удостоверяване в детайли. Той следва потока OAuth2 с код за оторизация. Обектите, които си взаимодействат тук са СУКАО уеб интерфейса, СУКАО API сървър, GitLab уеб интерфейса и GitLab API сървър. Започвайки с натискане на бутона "Login", уеб интерфейса на системата прави заявка към сървъра. Той вика сървъра на GitLab с път /oauth/authorize и така потребителя е пренасочен към страницата за удостоверяване на GitLab. След въвеждане на нужната информация, СУКАО API сървър получава специален код за оторизиране, който той след това може да прати обратно на GitLab в комбинация със своите Client ID и Secret. Наближавайки края на процеса, сървър на системата за управление на курсове получава токен, с който може да прави заявки към GitLab от името на

потребителя, за да вземе неговите данни - име, имейл и профилна снимка. Стигайки това състояние, потребителя се счита за удостоверен и се пренасочва към потребителския интерфейс на СУКАО.



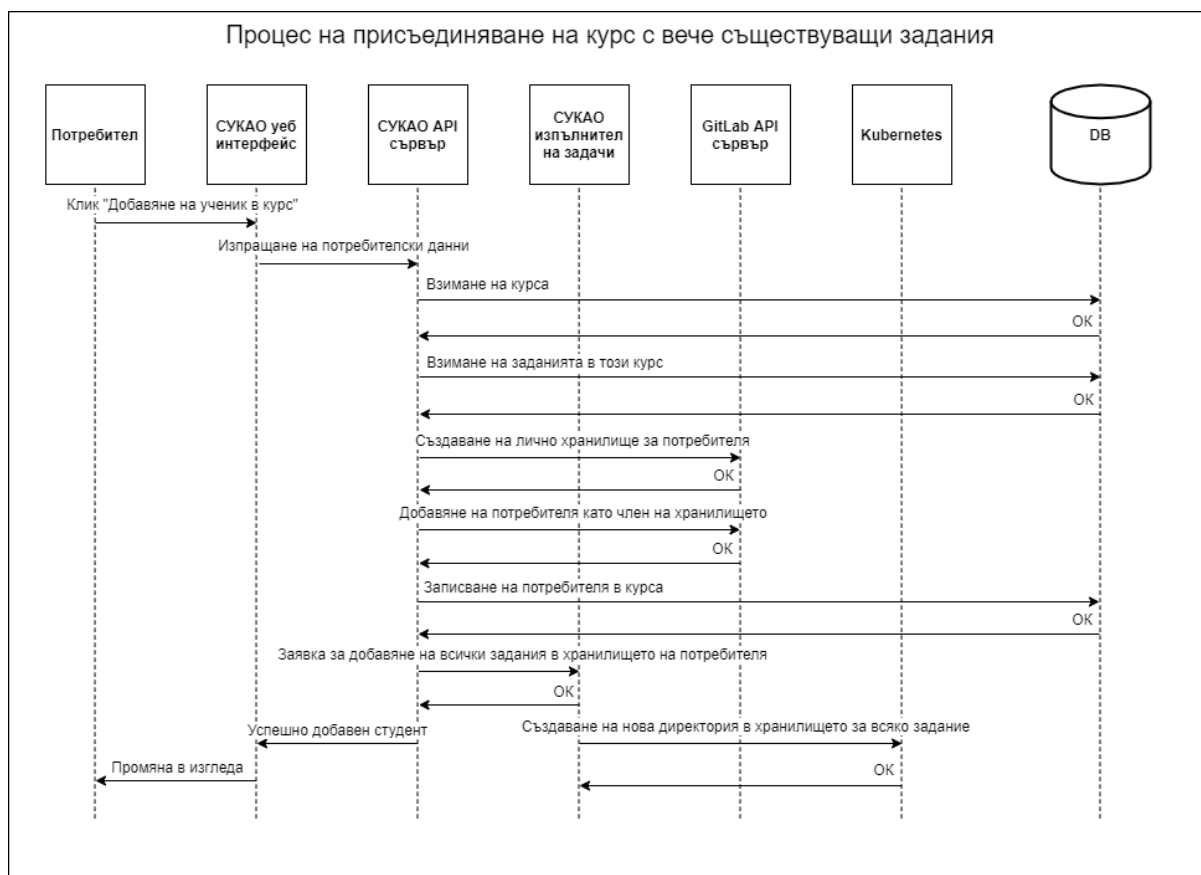
Фигура 5 - Процес на създаване на курс - диаграма на последователност

Следващият процес, изобразен на фигура 5, който си струва да бъде разгледан задълбочено е процесът на създаване на курс. Той започва с клик върху бутон "Създаване на курс" и въвеждане на име и описание. Тези данни се изпращат към API сървъра на системата. Той от своя страна се обръща към GitLab API сървъра, за да създаде група, в която ще се намират хранилищата на всички членове на курса. След това прави една заявка за създаване на хранилище за лектора, и друга, за даване на права на учителя да го използва. Получавайки статус код за успех, СУКАО запазва данните за курса в базата от данни и връща съобщение за успех на потребителя.



Фигура 6 - Процес на оценяване на решение - диаграма на последователност

На фигура 6 е описана основната функционалност на СУКАО - автоматичното оценяване на решения. В този процес участват абсолютно всички компоненти на системата, включително Kubernetes API сървър и GitLab API сървър. Той започва с натискането на един бутон в уеб интерфейса, който изпраща заявка за оценяване на решението. Още в началото, в базата се запазва информация, че е предадено решение, но то още е в процес на обработка. След това се вика компонента, отговорен за изпълнението на задачи. Той обработва асинхронно заявката и отговаря веднага, че е приета и на фона добавя задачата в опашката. Задачата включва създаването на Kubernetes Pod чрез Kubernetes API сървър. В него се пуска контейнер, който тегли решението на студента, тестовите на лектора, компилира ги, форматира резултатите и ги принтира на стандартния изход. Изпълнителя на задачи ги извлича и ги запазва в базата от данни и променя статуса на завършен.



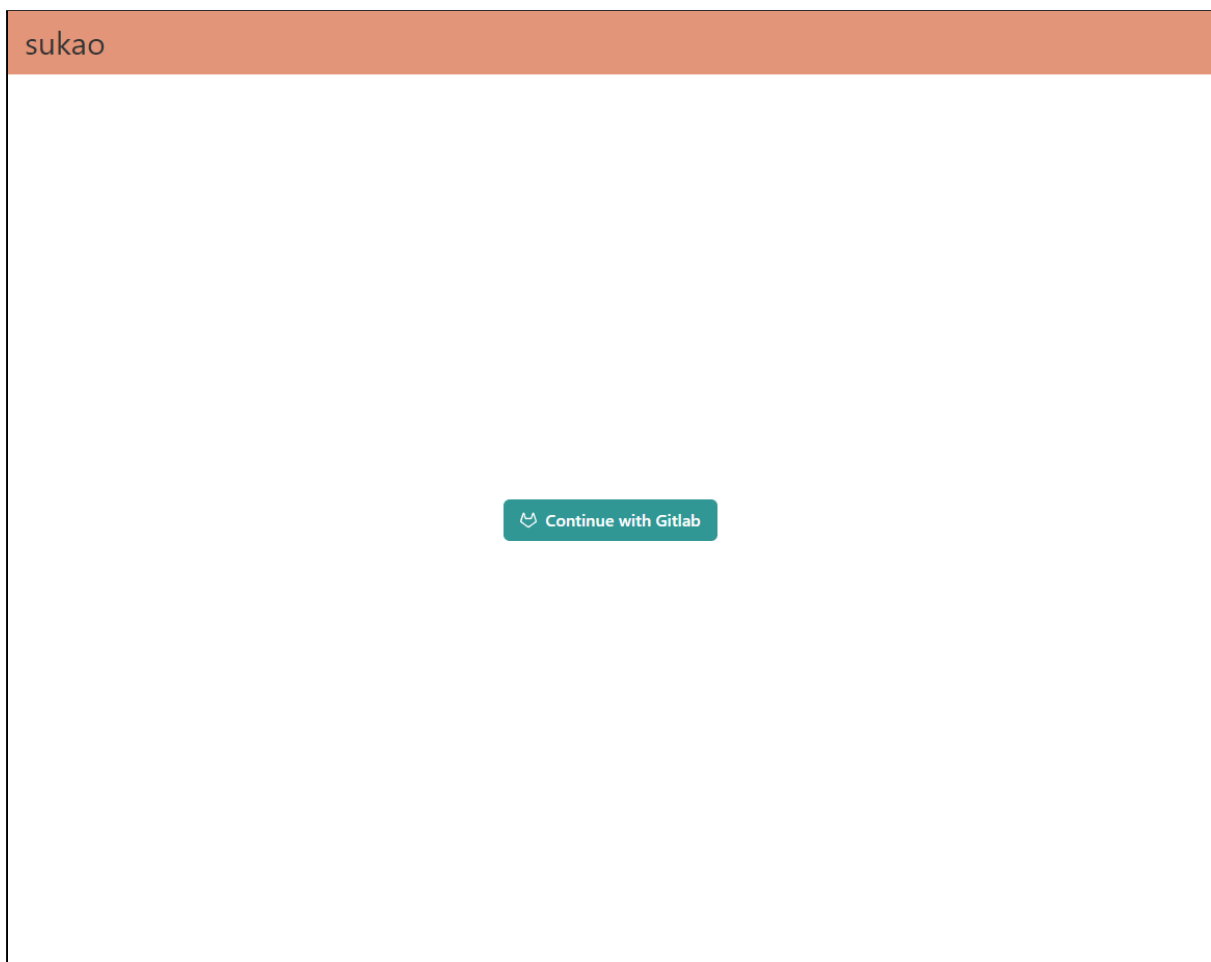
Фигура 7 - Процес на присъединяване на курс с вече съществуващи задания - диаграма на последователност

Друг сложен процес е присъединяването на курс (изобразен на фигура 7), в който вече съществуват задания. Той отново включва всички компоненти както във фигура 6. Тук основният проблем е създаването на всички задания в хранилището на студента, което в началото не съществува. Интересно е как системата се справя с това - използвайки асинхронни задачи. Предпоставка за този процес е съществуването на курс с поне едно задание. Първо, лектора добавя студент в курса от административно-учителския изглед. Тази информация се изпраща на СУКАО API сървъра. Той извлича данни за курса и всички негови задания от базата от данни. След това той има всичко нужно да създаде лично хранилище за студента в GitLab и да го направи член. Протичайки успешно, се запазва в базата информация, че студента се е присъединил в курса. В този момент хранилището му е празно и той може да вижда заданията само в СУКАО, но не и в GitLab. По същия начин, както в процеса от фигура 6, се добавя задача с входни данни - имената на всички задания, в опашката на компонента, отговорен за тяхното изпълнение. Този път

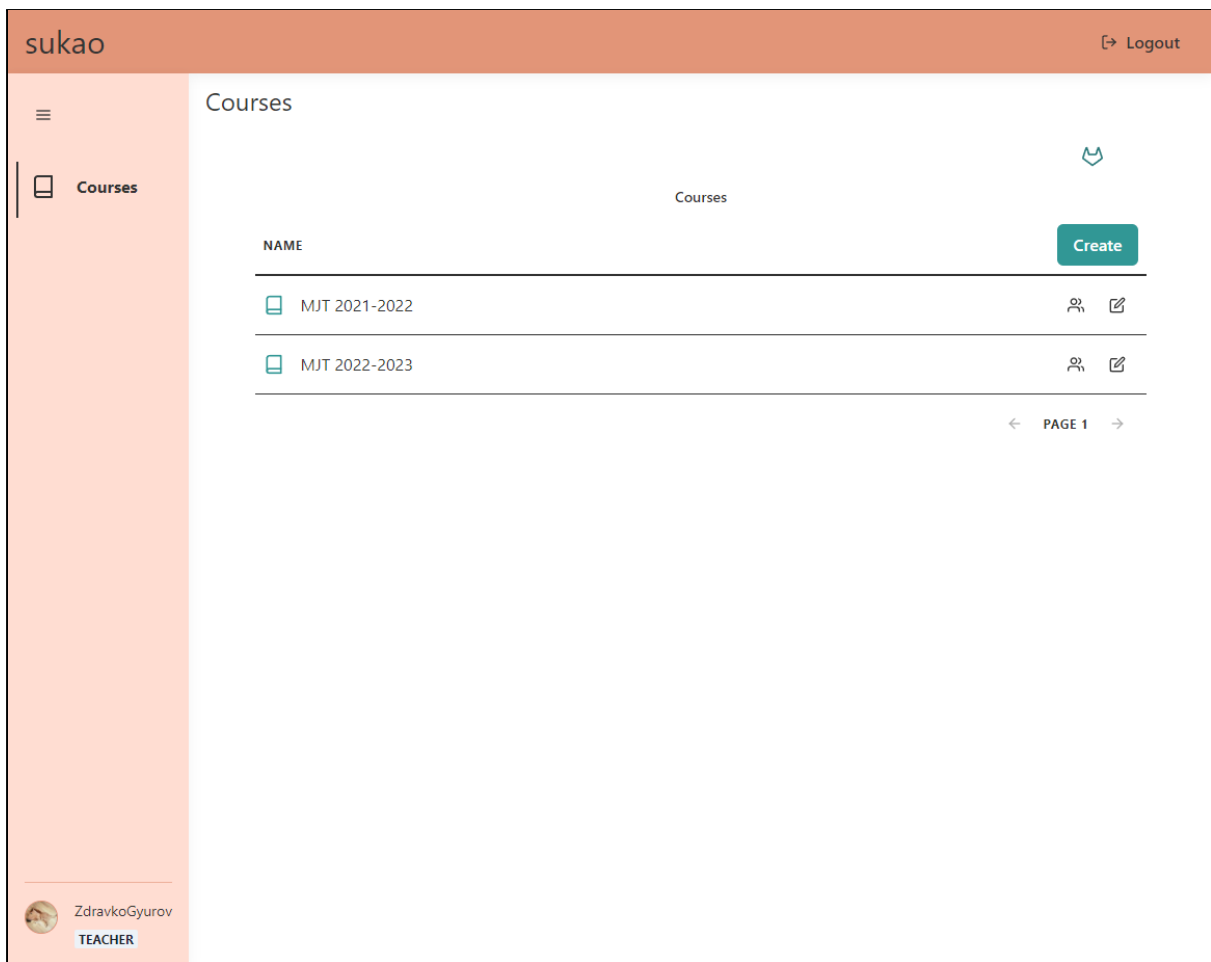
обаче задачата е друга. Притежавайки нужните данни, контейнера може да добави директориите, съдържащи условията на заданията.

Реализиране на уеб интерфейса

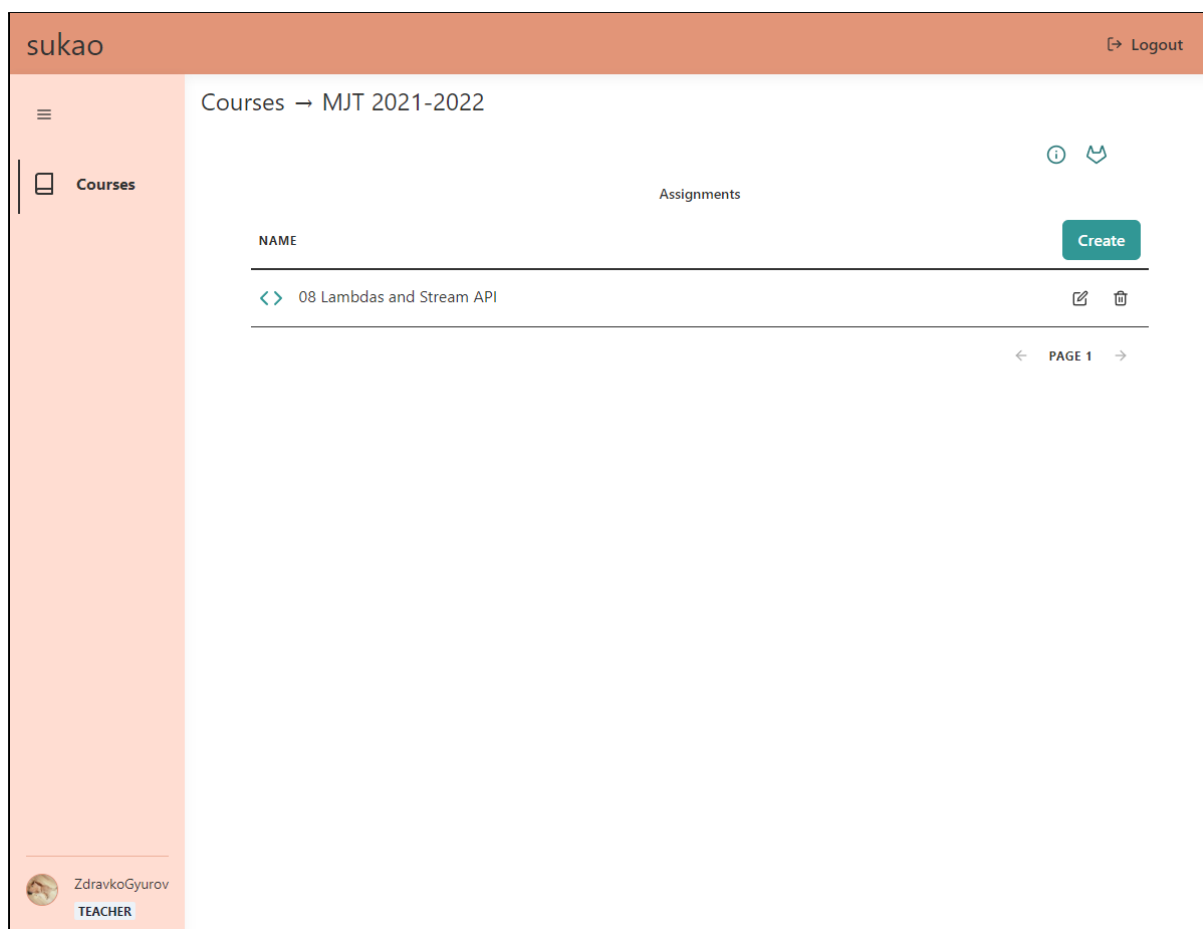
За разработването на уеб интерфейса е използвана JavaScript библиотеката React. За изграждането на красив и ефектен дизайн е използвана библиотеката Chakra UI, която предоставя градивните елементи.



Фигура 8 - Начална страница на СУКАО уеб интерфейса



Фигура 9 - Страница с всички курсове в веб интерфейса



Фигура 10 - Страница с всички задания в веб интерфейса

sukaoLogout

Courses

Courses → MJT 2021-2022 → 08 Lambdas and Stream API

Submissions

Submit

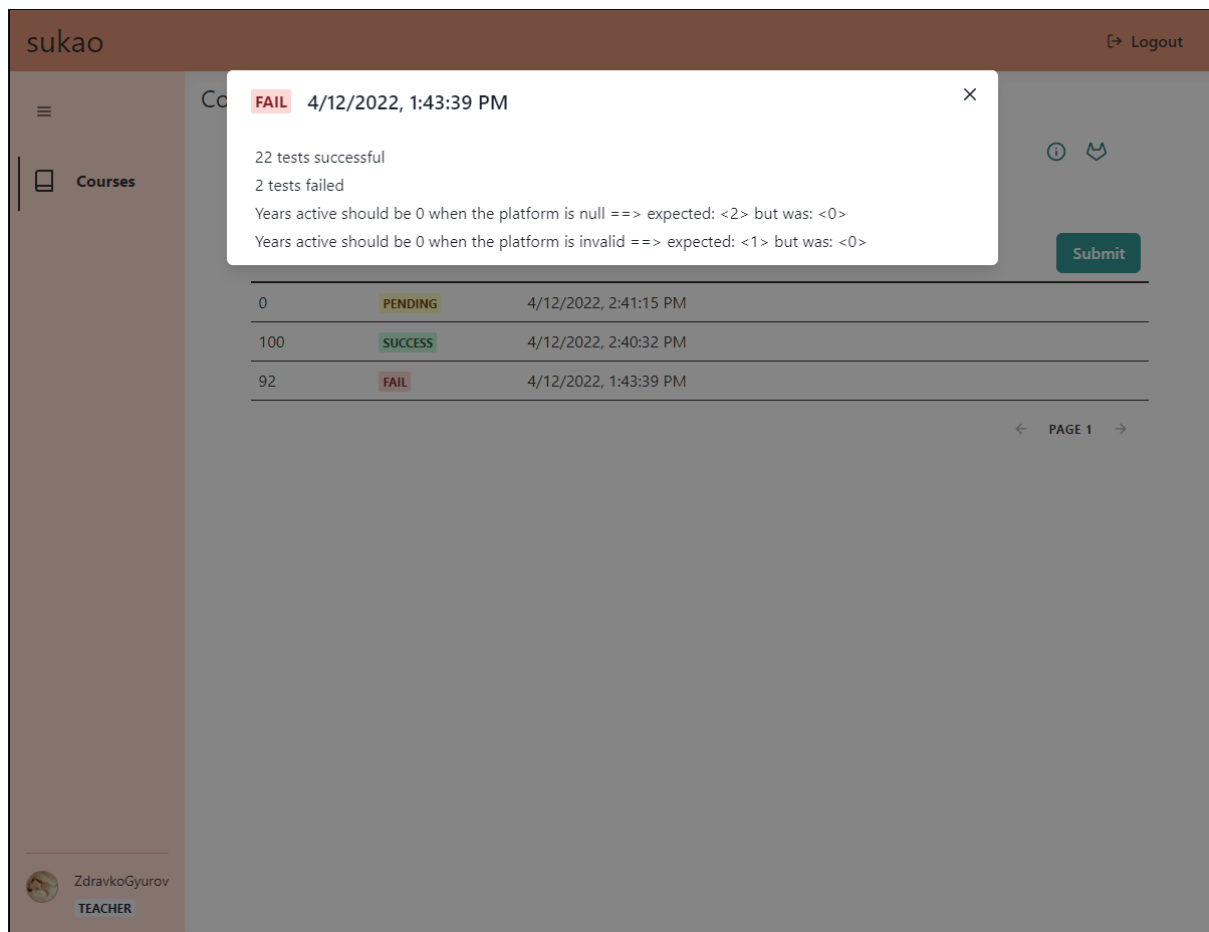
POINTS	STATUS	DATE
0	PENDING	4/12/2022, 2:41:15 PM
100	SUCCESS	4/12/2022, 2:40:32 PM
92	FAIL	4/12/2022, 1:43:39 PM

← PAGE 1 →

Zdravko Gyurov

TEACHER

Фигура 11 - Страница с предадените решения в веб интерфейса



Фигура 12 - Страница с резултат от предадено решение в уеб интерфейса

< снимка на учителски изглед на потребителите в курс >

< снимка на админски изглед на потребителите >

React components

Снимки на ui-a

Реализиране на API сървъра

Go project structure, apis, input, output

Jwt auth

Реализиране на компонента за изпълняване на тестове

Go project structure, apis, input, output

Реализиране на Kubernetes шаблоните

K8s templates

Helm values file

5. Експерименти и анализ на резултатите

Todo

Use apache benchmark to test rate limit

Spam jobs

6. Заключение

Постигнати резултати

Проучени са системи за ...

Проектирана е система за ...

Реализирана е система ...

Приноси

Научни

Сравнителен анализ на съществуващи системи

Научно-приложни

Проектиране

Приложни

Имплементация

Апробация

системата ще се ползва за ...

Насоки за бъдеща работа; Перспективи

rabbitMQ

Използвана литература

Европейски парламент и Съвет на Европейския съюз. “Регламент (ЕС) 2016/679 на Европейския парламент и на Съвета.” *EUR-Lex*, 27 April 2016, <https://eur-lex.europa.eu/legal-content/BG/TXT/?uri=CELEX%3A32016R0679>. Accessed 6 April 2022.

Birgen, Cansu. “SQL vs. NoSQL.” 8 December 2014, p. 42, https://www.researchgate.net/publication/339883071_SQL_vs_NoSQL.

Brewer, E. A. “Towards robust distributed systems. In PODC.” July 2000.

codePost. “codePost.io.” *codePost: Autograder and code review for computer science courses*, <https://codepost.io/>. Accessed 24 March 2022.

Docker. “What is a Container?” *Docker*, <https://www.docker.com/resources/what-container/>. Accessed 1 April 2022.

Edwards, Stephen. "What is Web-CAT?" *Web-CAT*, 1 October 2020, <https://web-cat.org/projects/Web-CAT/WhatIsWebCat.html>. Accessed 22 March 2022.

GitHub. "Basics of setting up GitHub Classroom." *GitHub Docs*, <https://docs.github.com/en/education/manage-coursework-with-github-classroom/get-started-with-github-classroom/basics-of-setting-up-github-classroom>. Accessed 24 March 2022.

GitHub. "GitHub Classroom." *GitHub Classroom*, <https://classroom.github.com/>. Accessed 24 March 2022.

Helm Authors. "What is Helm?" *What is Helm?*, <https://helm.sh/>. Accessed 7 April 2022.

The Kubernetes Authors. "Kubernetes." *Kubernetes.io*, <https://kubernetes.io/>. Accessed 4 April 2022.

Micro Focus. "Benefits of Using Docker." *Micro Focus*, <https://www.microfocus.com/documentation/enterprise-developer/ed40pu5/ET-S-help/GUID-F5BDACC7-6F0E-4EBB-9F62-E0046D8CCF1B.html>. Accessed 1 April 2022.

Pretchett, D. "Base: An acid alternative." 28 July 2008.

Wallen, Dave. "OAuth 2.0: What Is It and How Does It Work? | Spanning." *Spanning Backup*, 12 February 2020, <https://spanning.com/blog/oauth-2-what-is-it-how-does-it-work/>. Accessed 6 April 2022.