



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	5
Речник на термините	5
Списък на фигурите	7
Списък на таблиците	7
Увод	7
Структура на дипломната работа	8
Дефиниции	9
Обзор на подобни системи	9
Компютърно подпомагано обучение. Предимства и ограничения.	9
Предимства	10
Ограничения	10
Сравнение и анализ на подобни системи	10
Подобни системи	13
Web-CAT	14
Основни силни страни	15
Опит със системата	16
codePost	18
Основни предимства на codePost	19
Опит със системата	20
GitHub Classroom	22
Основни силни страни	22
Използване на системата	24
Сравнителен анализ	25
Обобщение	27
Проектиране на системата	28
Разрешаване проблемите на конкурентните системи	28
Предоставяне на функционалностите, съществуващи в конкурентните системи	31
Изпълняване на кода във всякаква среда с Docker	32
Какво е контейнер?	32
Предимствата на Docker контейнерите	33
Оркестриране на контейнерите с Kubernetes	34
Какво е Kubernetes?	34
Основни характеристики	34
Проектиране на базата от данни	36
Проектиране на уеб интерфейса	40
Проектиране на API сървъра	41

Проектиране на компонента за изпълняване на тестове	44
Инсталиране на системата	45
Реализиране на системата	48
Основни работни потоци	48
Реализиране на уеб интерфейса	53
Реализиране на API сървъра	60
Структура	60
Приложно-програмни интерфейси	60
Вход в системата	60
Продължение на OAuth2 потока за вход в системата, извиква се само от GitLab	61
Вземане на информация за текущия потребител	61
Вземане на информация за всички потребители в системата	61
Промяна на ролята на потребител в системата	62
Обновяване на токена за достъп до системата	62
Изход от системата	63
Създаване на курс	63
Вземане на информация за всички курсове, в които членува текущият потребител	64
Вземане на информация за определен курс	64
Редактиране на име и описание на курс	65
Изтриване на курс	65
Създаване на задание	65
Вземане на информация за всички задания в курс	66
Вземане на информация за конкретно задание в определен курс	67
Редактиране на име и описание на задание	67
Изтриване на задание	68
Предаване на решение за тестване	68
Вземане на резултатите от тестовете на всички предадени решения за определено задание	68
Вземане на резултатите от тестовете на предадено решение за определено задание	69
Добавяне на потребител в курс	70
Вземане на информация за ролята на всички потребители в даден курс	70
Промяна на ролята на потребител в курс	71
Премахване на потребител от курс	71
Реализиране на компонента за изпълняване на тестове	71
Структура	71
Приложно-програмни интерфейси	72
Създаване на заявка за тестване на предадено решение от студент	72

Създаване на заявка за добавяне на работни директории в хранилищата на новонавлезлите студенти в курс	72
Реализиране на Kubernetes шаблоните	73
Експерименти и анализ на резултатите	74
Заклучение	76
Постигнати резултати	76
Приноси	77
Научни	77
Научно-приложни	77
Приложни	78
Апробация	78
Насоки за бъдеща работа; Перспективи	78
Използвана литература	78

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

JWT - JSON Web Token

CAL - Computer Assisted Learning

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 веб сървър; междинното ниво между заявките, идващи от веб браузъра и базите от данни или други приложения на веб сървъра

plug-in - софтуерент 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

JSON Web Token - Това е отворен интернет стандарт за създаване на криптирани и подписани данни във формат JSON, може да съдържа информация за потребител и да се използва като механизъм за удостоверяване

Computer Assisted Learning - компютърно подпомагано обучение

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

Фигура 13 - Страница с всички потребители в даден курс в уеб интерфейса, достъпна само за учители

Фигура 14 - Страница с всички потребители в уеб интерфейса, достъпна само за администратори

IV. Списък на таблиците

Таблица 1 - Развити инструменти

Таблица 2 - Наскоро разработени инструменти

Таблица 3 - Сравнение на подобни системи

Таблица 4 - 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 таргетират описаната проблемна област. След това ще преминем през техните силни страни и ще направим обзор на системите в действие. Най-накрая ще ги сравним и ще съпоставим техните предимства и недостатъци.

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 са избрани именно те. 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 върху системата, за да се симулира реална работна среда и ще анализираме 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

Компютърно подпомагано обучение. Предимства и ограничения.

Компютърно подпомаганото обучение има потенциал да трансформира напълно образователния процес и значително да подобри ефективността на учене чрез осигуряване на голяма мотивация на студентите. CAL предоставя начин, по който може да се използва технологичния достъп, който днешните студенти имат, за тяхно добро. CAL им дава свобода да експериментират с различни опции, получавайки незабавна обратна връзка. Самостоятелното темпо е друга незаменима функция. Тази помощ от технологиите могат да осигурят много допълнително време на учителите да работят с ученици, които се нуждаят от повече внимание. Функциите за поверителност помагат на срамежливите студенти да изпробват нови неща без страх от останалата част от колегите им, узнавайки колко пъти са се опитвали и дали отговорите им са били правилни или грешни. Въпреки многобройните страхотни функции, които CAL може да предложи, прекомерната употреба на мултимедия може да отклони вниманието от съдържанието и ученето става твърде механично.

Предимства

- CAL е индивидуализирано, тоест всеки студент е свободен да работи със собствено темпо, напълно незасегнат от другите студенти.
- Информацията е представена в структурирана форма.
- CAL принуждава активно участие от страна на студентите, което контрастира с по-пасивното алтернативно обучение - четенето на книга или посещаването на лекции.
- CAL използва система за оценяване, която изгражда ясна картина на студента за неговия прогрес. Така студентите могат да идентифицират предметни области, в които са се подобрили и в които се нуждаят от подобрене.
- Чрез предоставяне на практически опит на студентите се намалява нужното време за разбиране на трудни концепции.

Ограничения

- Има реални разходи, свързани с разработването на CAL системи. Скъпо е по отношение на времето на персонала за изработване на ефективна CAL програма.
- Съдържание, обхванато от определена CAL програма може да стане остаряло. Разработката има висока цена, а ако един курс е остарял, ресурсите вложени в разработването му ще бъдат пропиляни.
- Мотивирането и обучението на учители да използват компютри в образованието е предизвикателна задача. Те може да не желаят да отделят допълнително време за подготовка, избор и използване на CAL програма. Може също да го възприемат като заплаха за работата им.

~(Sharma 3)

Сравнение и анализ на подобни системи

Име	Основни функции	Поддържани езици	Начин на работа	Показатели за оценяване
CourseMaker	Мащабируемост, поддржка.	Java, C++	Самостоятелна	Типография. Функционалност.

	Сигурност, конфигурируемост. Откриване на плагиатство. Работа с различни нива на обратна връзка.			Използване на структури. Дизайн на обекти. Отношения между обектите.
Marmoset	Подробна информация. Езикова независимост. Сигурност и мащабируемост на модула за оценяване. Apache 2.0 лиценз.	Всеки език	Самостоятел на	Динамичен и статичен анализ.
WebCat	Разширяемост и гъвкавост на базата на пългини. Сигурност на достъпа. Преносимост. Полу и напълно автоматичен процес. GNU GPL лиценз.	Java, C++, Scheme, Prolog, Standard ML и Pascal. Гъвкавост за всеки език.	Самостоятел на	Коректност на кода. Пълнота. Валидност на тестове. Разширяема чрез пългини.
Virtual Programming Lab	Интеграция с Moodle. Конфигурируем режим за оценяване. GNU GPL лиценз. Откриване на плагиатство. Конфигурируеми дейности. Затворническа среда.	Ada, C, C++, C#, Haskell, FORTRAN, Java, Octave, Pascal, PHP, Prolog, SQL, Ruby, Python, Scheme, Vhdl.	Плъгин за Moodle	Коректност, базирана на тестови сценарии. Отворен за нови методи.

Grading Tool (Магдебургски университет)	Използване на услуги. Конфигурируем процес на оценяване.	Haskell, Scheme, Erlang, Prolog, Python, Java	Разширение на система за управление на обучение	Компилация. Изпълнение. Динамични тестове.
--	--	---	---	--

Таблица 1 - Развити инструменти

Име	Основни функции	Поддържани езици	Начин на работа	Показатели за оценяване
JavaBrat	Използване на услуги. Интеграция със система за управление на обучение.	Java, Scala	Плъгин за Moodle. Самостоятелна.	Коректност.
AutoLEP	Статичен и динамичен анализ за оценяване.	-	Самостоятелна	Статичен анализ. Динамичен анализ.
Petcha	Координация между съществуващи и инструменти за поддръжка на програмиране. Използване на технология за оперативна съвместимост.	Езици, поддържани от Eclipse и Visual Studio	Самостоятелна	Базирани на тестови сценарии.
JAsses	Интеграция с Moodle.	Java	Плъгин за Moodle	Компилиране.
RoboLIFT	Оценяване на мобилни приложения. Оценяване на графичен потребителски интерфейс.	Java	Самостоятелна	Тестване на градивни единици (публични и частни)

Moodle ext. (Словашки технологичен университет)	Ориентирана към дигитални системи. Откриване на плагиатство.	Vhdl.	Плъгин за Moodle.	Компиляция. Статичен анализ. Функционалн ост в сравнение.
--	---	-------	----------------------	--

Таблица 2 - Наскоро разработени инструменти

~(Calza and Del Alamo #7)

Подобни системи

В тази глава ще разгледаме няколко от най-популярните приложения и уеб услуги, които покриват възможно най-много изисквания от описания проблем. Първо, нека дефинираме най-важните функционални и нефункционални изисквания, които ще ползваме като критерии за сравнение.

1. Системата трябва да предоставя възможност да се управляват голям брой курсове (стотици, хиляди).
2. Всеки курс трябва да може да съдържа голям брой практически задания (десетки).
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 максимален брой аспекти на решенията на студентите автоматично.
7. Написаните от преподавателския състав тестове към решенията не трябва да са видими за студентите.
8. Системата трябва да може да открива признаци на плагиатство у решенията на студентите.
9. Системата трябва да е гъвкава по отношение на технологиите, които се преподават в курса. Целта е една и съща система, позната както на преподавателите, така и на студентите, да може да се използва в множество курсове.

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 разглеждане на съществуващи подобни системи.

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 новонавлизащи 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. Тя е създадена през 2006 г. от преподаватели от Техническия университет във Вирджиния.

Web-CAT е система, която се инсталира 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 само върху входно-изходни тестове. т.е. дали кодът на студента при даден вход извежда даден изход. Това са така наречените системи тип Online Judges, които произхождат и са подходящи по-скоро за състезания по алгоритми, отколкото като инструмент в курсове по програмиране. 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 изхода при подадени входни данни.

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 такава. 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 основно 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). Отделно трябва да

се инсталират и конфигурират две зависимости: релационна база от данни (MySQL) и мейл сървър.

Разширяемост (Extensibility)

Може би най-голямата сила на Web-CAT е вградената 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

Разглеждането на документацията на Web-CAT дава отгора, защо тя е една от най-използваните системи от рода си. Системата покрива повечето критерии, но за цялостен и изчерпателен анализ, трябва също предоставяните функционалности да се тестват на практика.

След 8-месечен практически опит със системата (4 като студент и 4 като асистент в курс), авторът на настоящата дипломна работа може да обобщи, че

системата изпълнява голяма част от критериите, но има и определени недостатъци.

Първо, от гледна точка на достъпност на системата за преподавателите, за да се използва от тях в курс по програмиране. Тъй като това не е хоствана уеб услуга, преподавателят трябва сам да се погрижи за поддържането и предоставянето ѝ на други потребители. Алтернативно, би могло да има една инстанция на Web-CAT системата, която да се споделя в рамките на факултет или университет, за да се споделят разходите за инфраструктура и системно администриране, които биха били значителни, предвид изискването за мащабируемост при натоварване и достъпност на услугата 24/7.

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 времеемък и предразполага към грешки.

При създаване на курсове и задания, вградените ограничения за максимален брой символи в описания или условията на заданията, налагат те да се съхраняват в друга платформа и само да се реферират от WEB-Cat. След създаване на задание, потребителите виждат само името му и линк към детайлното описание и към интерфейс, през който студентите могат да качват решенията си като .zip архив. Използването на zip архиви за контрол на версиите на изходен код обаче е много неудобно. Докато решават задачата, студентите постоянно правят промени по решенията си, поправят грешки и качват нова версия на решението си, което е напълно нормален процес на учене и работа. За една задача, те могат да създадат и качат множество версии на решението, което затруднява преглеждането и оценяването. Студентите трябва да знаят, коя версия в кой архив се намира, какви точно промени са направени в определена версия и още подобни детайли. След като се качи архива с финалната версия, в системата не се визуализира ясно кода, който се съдържа там, а за да се провери, дали е качена актуалната версия трябва да се изтегли решението, да се разархивира и разгледа цялостно в подходящ редактор, което е сериозен недостатък. В допълнение, от гледна точка на преподавателския състав, когато някой студент има въпроси или нужда от помощ, неговото решение отново трябва да бъде свалено и разгледано локално.

При ръчна проверка и оценяване на решения, се използва WYSIWYG интерфейс, който обаче като потребителско изживяване не е оптимален, тъй като не е адаптивен, съдържа досадни грешки и е труден за използване. Навигацията е неконсистентна и при преминаване между различните изгледи, системата в определени ситуации е нестабилна.

С цел сигурност и контрол върху използваните изчислителни ресурси, в системата има и сериозни ограничения относно елементите на езика за програмиране, които могат да се използват в студентските решения - основно свързани с многонишково програмиране, достъп до файловата система и т.н.. Изходният код на системата е отворен, но липсва документация, поддръжка и общност около проекта. От архитектурна гледна точка, при всяко качване на студентско решение, се пуска нов процес на операционна система за обработката му (компиляция, изпълнение на тестовете и т.н.). Ресурсите на операционната система - файлова система, памет и т.н. обаче се споделят между всичко процеси. В следствие, липсва изолация по време на изпълнение на кода на студентите, който се изпълнява на машината.

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 Бостънския университет, Принстънския университет, Университета на Айова, Университета Корнел и други.

Системата има добро бързодействие и е лесна за използване.

Основен фокус на системата codePost е да помогне на преподавателите по компютърни науки да предоставят качествена обратна връзка на студентите относно архитектурата, дизайна и качеството на кода на техните решения. Системата е разработена в Принстънския университет, в началната си версия

от екип от студенти, чиято основна мотивация е била дигитализиране на целия процес по предаване, проверка и оценяване на задания по програмиране.

Основни предимства на codePost

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 стандартизиран, предварително дефиниран от преподавателския екип, каталог от забележки и коментари. Такъв каталог от една страна улеснява проверката, а от друга, намалява субективността при оценяването. 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, с цел ефективно откриване на грешки в студентските решения, дори в курсове с много голям брой студенти. Преподавателският състав може да се възползва от обикновени входно-изходни тестове (в стил Online Judge), които не изискват писане на код. Могат също да се автоматизират и по-гъвкави и сложни 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 могат да се визуализират или като обекти, или като прости текстови файлове. При извеждането на резултатите, автоматично се добавят или изваждат точки за всеки тест според резултата от изпълнението му, може да се добавят обяснения, а ако тестът не се е изпълнил успешно, да се визуализира и подсказка на студента за тествания сценарий. Системата позволява да се посочват лимити на броя тестове, които могат да се

изпълняват. Могат да се задават и изисквания за имената и размерите на файловете, които трябва да се качат като студентско решение. Тестовите се изпълняват в изолирана среда.

Отворена и оперативно съвместима

Освен функционалностите, предоставяни от системата, възможностите ѝ могат да се разширяват или интегрират с външни системи чрез скриптове и публично API, 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 покачва 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, за да предостави максимална сигурност, бързодействие и удобство. В сравнение с Web-CAT, характеристиките, които изпъкват най-много, са стабилността на системата и интуитивността при работа с уеб интерфейса. CodePost има и недостатъци. Основните слабости на Web-CAT са преодолени 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 преподавателски 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 създават курсове и задания.

След сдобиването с преподавателски 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 естетичен, което улеснява работата. Има малко на брой изгледи, което намалява шанса за объркване или “изгубване” в навигацията. 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 преподавателят да предостави 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.

Премайнавайки към оценяването на решения и писането на обратна връзка за студентите, виждаме може би най-добрата функционалност на codePost - 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 т.нар. *рубрики* от съобщения за обратна връзка. Системата дава възможност да се създадат рубрики, представляващи различни видове грешки, например 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 евентуален срив на системата в критичен

момент за курса като край на срок за предаване на някакъв проект или домашно е съществено неудобство. Друго затруднение, с което ще се сблъскат лекторите, е извличането на резултатите от тестовете, 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 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 ценен опит на студентите и ще ги научи на добри практики за работния процес, заради нейната същност.

Основни силни страни

Удобно оценяване

GitHub спестява време на преподавателския състав като използва автоматично тестване за оценяване на задания. Тестовете 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 тестовете 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 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.

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 2022 г.), няма никакви проблеми с мащабирането и поддържането на стотици студенти в един курс.

Преподавателите могат да разчитат на автоматичното разпределяне на задания и автоматичното тестване на решенията.

GitHub Classroom също прави лесно създаването на задания със стартов код и разпределянето им към студентите.

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, преподавателят вече е наясно с над 50% от функционалностите, менютата и бутончетата, с които 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 характеристика ще е това, че принадлежат на един и същи курс или използвайки GitHub термините - класна стая. Преди да премине към създаването на класната стая, преподавателят има възможност да покани хора в организацията и евентуално да ги направи администратори, като така те ще могат да му помагат с управляването на курсовете. Най-накрая преподавателят може да продължи със създаването на класната стая. Тази операция започва с избирането на асистенти, което не е финално и винаги може да се добавят или премахват в по-късен етап. За да се завърши процесът 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 може да се запишат факултетните номера на студентите (или друг идентификатор като имена или имейл), по един 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, което в последствие ще изисква ръчна модификация.

Последната функционалност, която завършва целия процес на провеждане на курс по програмиране с автоматично оценяване, е създаването на заданията. Това е процес от три 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 тестовете, които ще се изпълняват срещу решенията на студентите. Те могат да са написани на какъвто и да е език. Тук се натъкваме на сериозен 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 евентуални подобни действия.

Накратко, GitHub Classroom, точно както предните две системи, покрива голяма част от изискваните функционалности, като се справя с доста от проблемите по по-иновативен и удобен начин, но и тази система има сериозни недостатъци, споменати по-горе, 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 решения	Не	Да	Да
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 (изисква интеграция, чрез пългин)	Не (според документацията има, но към момента на писане не работи)	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 3 - Сравнение на подобни системи)

Обобщение

Както виждаме в таблица 3, 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, липсата на скрити тестове от студентите е значителен недостатък, който отново се среща само в GitHub Classroom. Нито един от тези инструменти обаче не се интегрира лесно с външни системи против плагиатство. Това принуждава учителите да инсталират плъгини, да извършват тази дейност ръчно или да си пишат собствени програми, които да го правят вместо тях. Откъм цена за ползване се срещат два 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 удовлетворява напълно изискванията ни. Web-CAT е без съмнение най-далеч от изградената ни визия за оптимална 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 неинтуитивна 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 трите, около 50 пъти по-непопулярна от GitHub (според брой резултати в Google) и също е платена за големи екипи, каквито ще са тези за курсовете. Така BitBucket отпада от възможностите за система за контрол на версиите. GitHub е около 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. Към момента, това са непреодолими проблеми.

След преразглеждане на алтернативите, GitLab се очертава като най-подходящото решение. 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 софтуер, който всеки може да 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 изградили собствен интерфейс, който е добре проектиран и работи стабилно. 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 възможността за скриване на тестовете 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 администратор в поверените им хранилища, а в същото време те ще могат да виждат само своя код. Преподавателят 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 преподавателския 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.

MOSS е ефективна система, разработена в Станфордския университет, за намиране на сходен софтуер, която се използва ръчно от много професори за откриване на плагиатство.

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 следващите подглави.

Предоставяне на функционалностите, съществуващи в конкурентните системи

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 тип UUID. Освен това, интегрирайки GitLab с нашата система, ще има 1 към 1 съответствие от курс в системата към група в GitLab. Поради тази причина, ще трябва да се пази идентификаторът и името на GitLab групата.

Сървърът на нашата система ще има връзка с базата и ще извършва Create, Read, Update, Delete (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 нишки. 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 изведат резултатите.

Този скрипт ще се изпълнява в специален Docker контейнер, който ще е Kubernetes Pod. На контейнерите може да се налагат 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 например 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 тестовете. Това позволява разширяването функционалността на системата със само няколко реда 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 от авторите си.

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 достъпи или използва 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 улесняват тази дейност. Те извършват и още много други операции, 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 през юни 2014 г. и в момента 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, blue-green deployment и canary deployment. 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 са приключили с ненулев статус код, заменя и насрочва наново контейнери, когато възлите, на които са вървели, се сринат. 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 нужно 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, са анализирани характеристиките на двата подхода.

- 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	Използва се най-актуалната информация
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

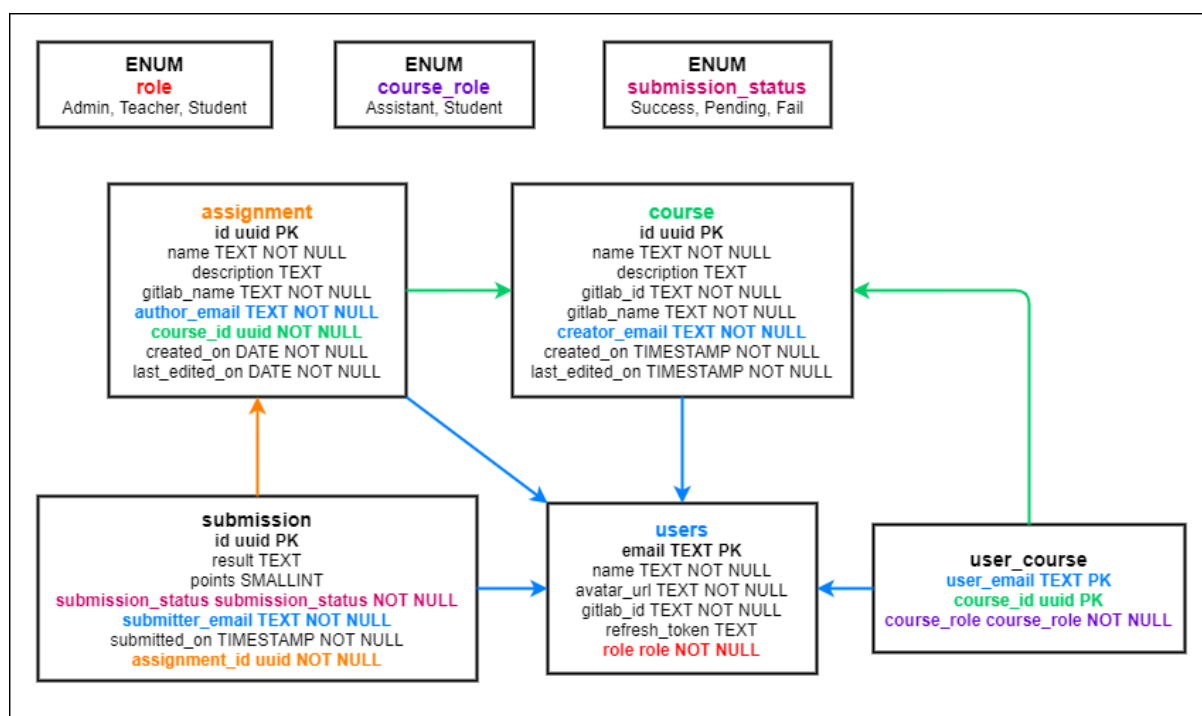
Таблица 4 - Сравнение на ACID и BASE

~ (Birgen 5)

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 (преподавател) 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 (завършили с неуспех). 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 релация many-to-many (много към много), 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, са събрани положителните страни на конкурентните системи и са анализирани бизнес нуждите на приложението.

Съвременните приложения основно се делят на такива с много HTML страници и такива с една HTML страница. По-старият 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 Single Page Application (SPA).

Технологията, избрана за създаване на приложението, е React.js, поради голямата си популярност. Също така, тя 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 всички потребители, които са влизали в системата, позволяващ задаването на преподавателска роля в системата (достъпен само за администратори);
- Административен изглед на даден курс с всички негови членове, позволяващ добавянето на студенти и асистенти в него (достъпен само за преподаватели).

Проектиране на API сървъра

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, сървъра 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 т.н. Много важна характеристика на такъв тип уеб услуги е, че те не пазят състояние за заявките. Един потребител изпраща отделните заявки и те нямат 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 предвиди 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, 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 авторизация 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, сървъра 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, отнемащ 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 сървъра в този момент. След отваряне на страницата отново и отново на sluки, 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 потребителят 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 с краен размер, 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 според ситуацията.

За самостоятелния учител, водещ малък курс, най-лесно би била формата на софтуер като услуга, притежаващ определени 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 ученици ще могат бързо и лесно да започнат да я употребяват, заемайки много малко ресурси от цялостната система.

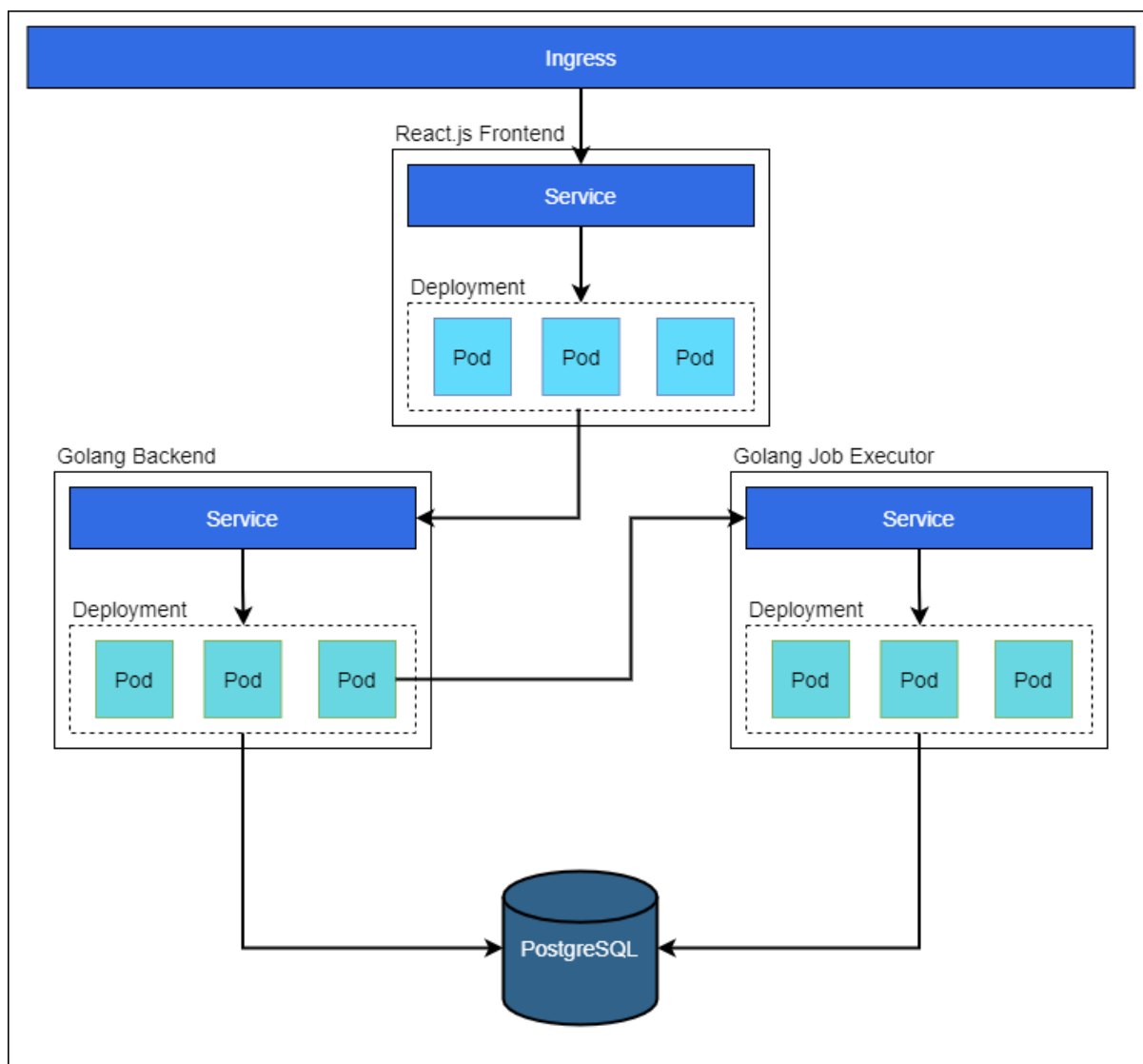
За лектори в университети, по-подходящо може да е програмата да работи на собствен хардуер. В този случай ще е нужен системен администратор, грижещ се за висока наличност на системата. От друга страна, по този начин приложението ще частно и изолирано специално за членовете на курса или университета, 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 безплатен софтуер, който всеки може да инсталира на собствен хардуер, включваща 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 характеристики, включително колко копия (*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 достъпят от 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, използващи се за чувствителни данни, и ConfigMap - за всички останали конфигурации. Има и други ресурси, настройващи мрежовата комуникация с всеки компонент. Всеки ресурс се създава с 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 тези качества са нужни за системата и нейното лесно доставяне едновременно като уеб услуга и като лесна за инсталиране програма на хардуера на клиентите.



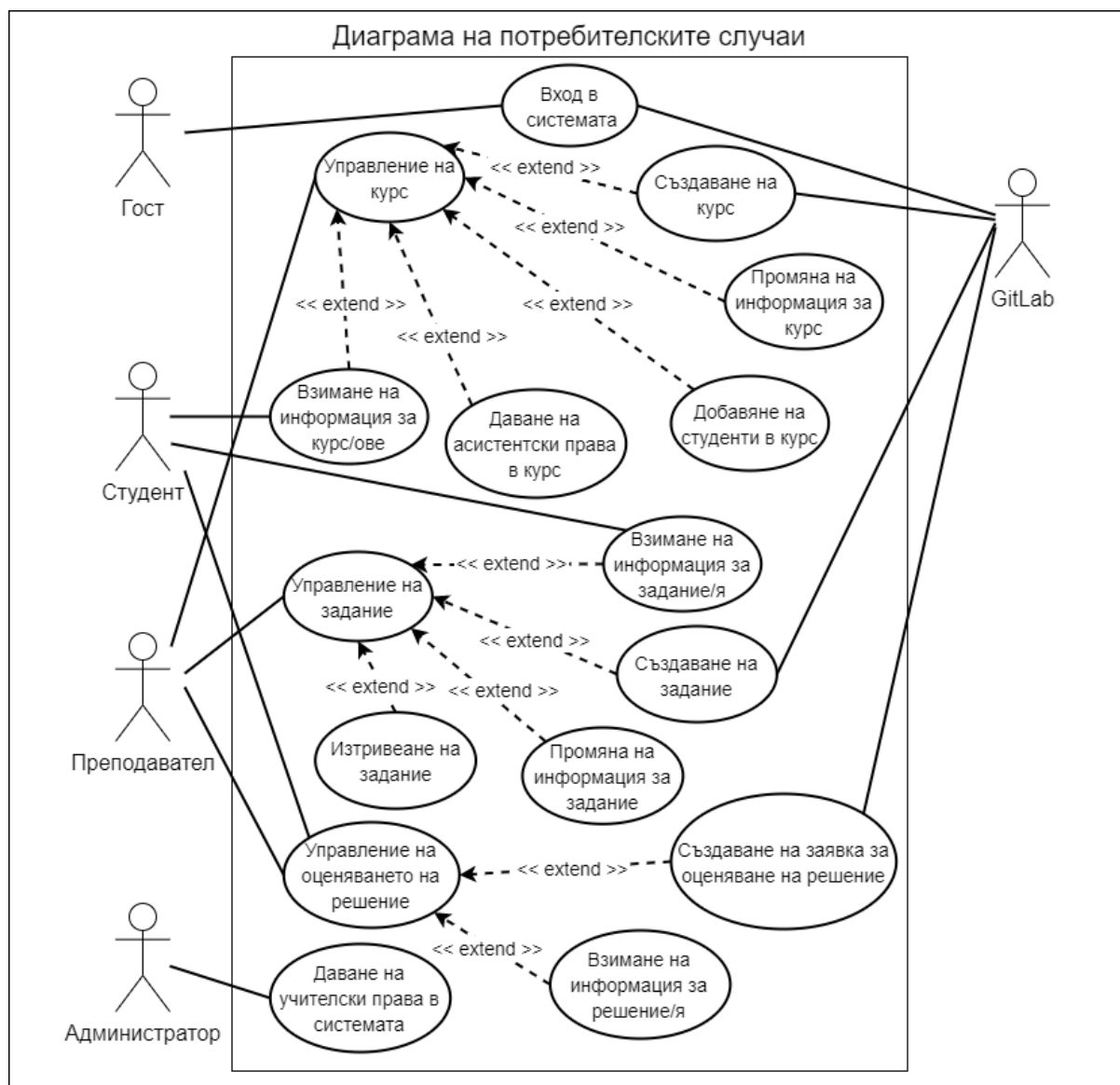
Фигура 2 - Диаграма на Kubernetes архитектурата

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 интерфейса 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

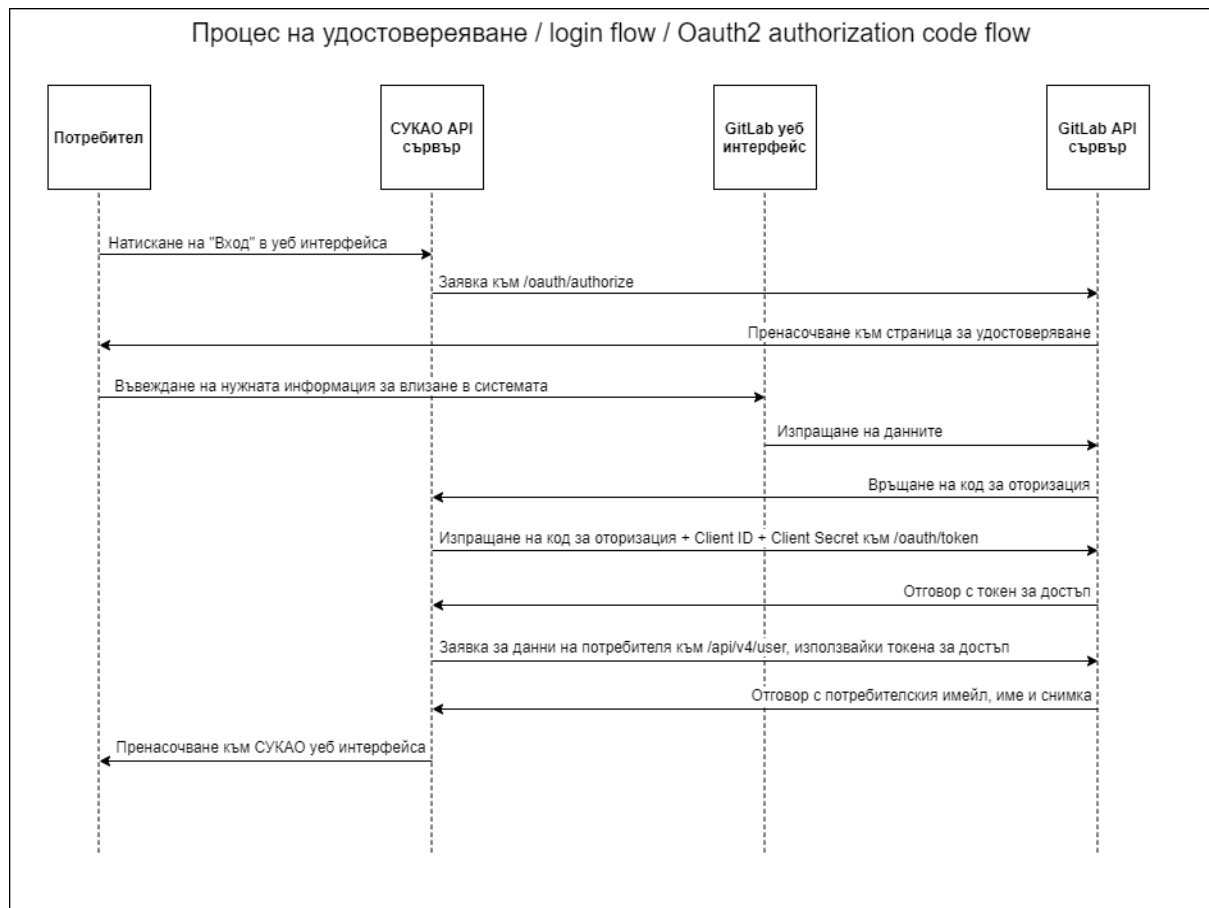
Реализирането на системата следва същите стъпки и се базира на проектирането. В допълнение, ще бъдат разгледани набор от диаграми, както на потребителските случаи, позволяващи по-широк поглед над цялостната система, така и на последователността, показващи в детайли най-сложните процеси, които системните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

Основни работни потоци



Фигура 3 - Диаграма на потребителските случаи

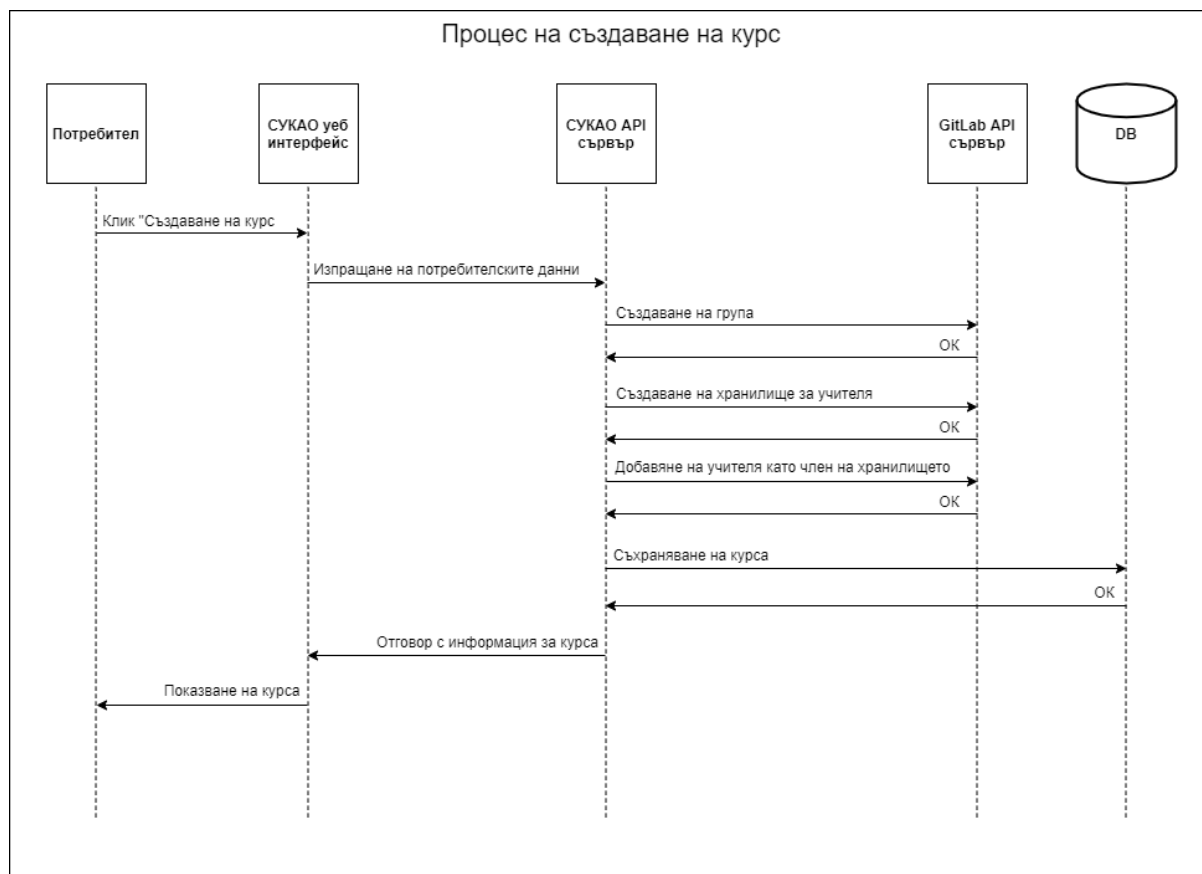
Във фигура 3 се вижда едно обобщение на системата, нейните потребители, наречени актьори, и техните взаимодействия. Реализирани са четири роли - гост, студент, преподавател и администратор. Последният актьор, изобразен на диаграмата, е външната система GitLab.



Фигура 4 - Процес на удостоверяване - диаграма на последователност

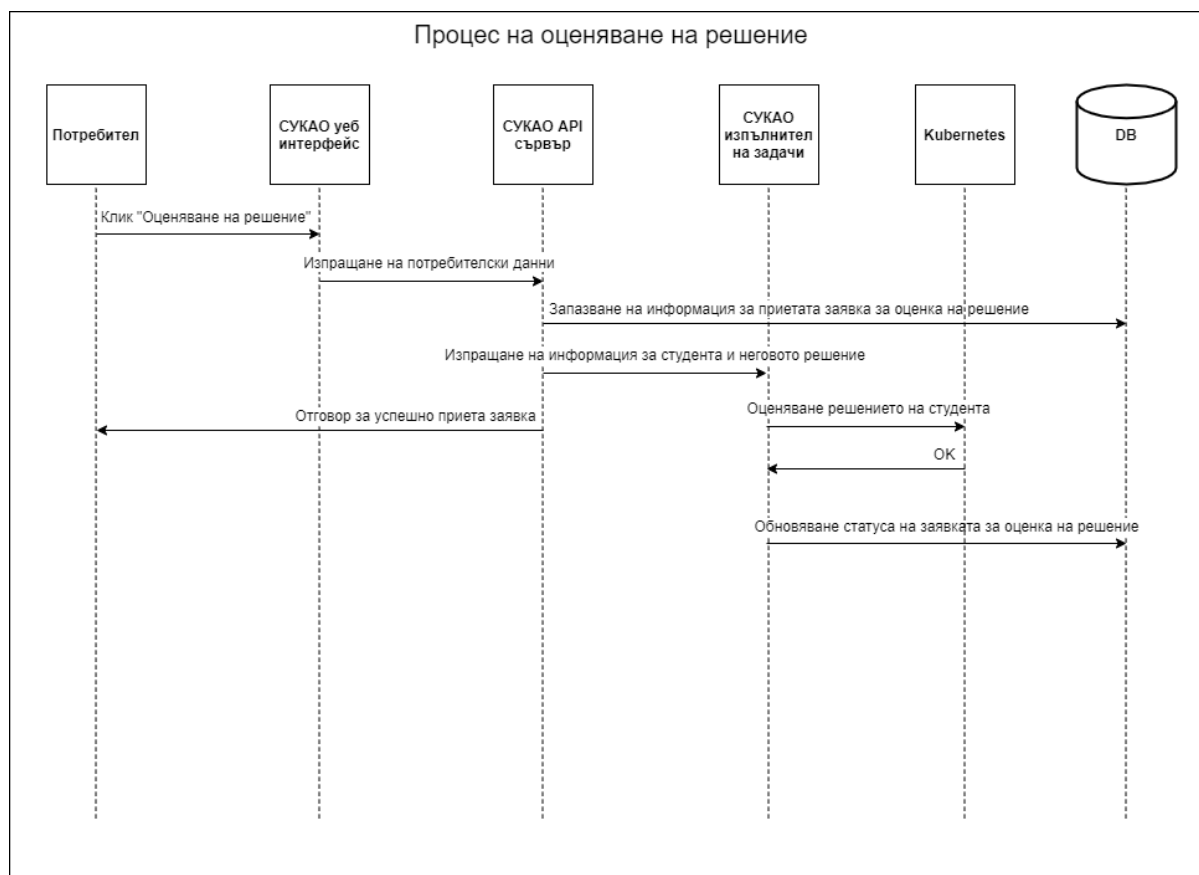
Във фигура 4 се вижда процесът на удостоверяване в детайли. Той следва потока OAuth2 с код за оторизация. Обектите, които си взаимодействат тук са уеб интерфейсът на Системата за Управление на Курсове и Автоматично Оценяване (СУКАО), СУКАО API сървърът, GitLab уеб интерфейсът и GitLab API сървърът. Започвайки с натискане на бутона "Login", уеб интерфейсът на системата прави заявка към сървъра. Той вика сървъра на GitLab с път /oauth/authorize и така потребителят е пренасочен към страницата за удостоверяване на GitLab. След въвеждане на нужната информация, СУКАО API сървърът 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, сървърът 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, потребителят се счита за удостоверен и се пренасочва към потребителския интерфейс на СУКАО.



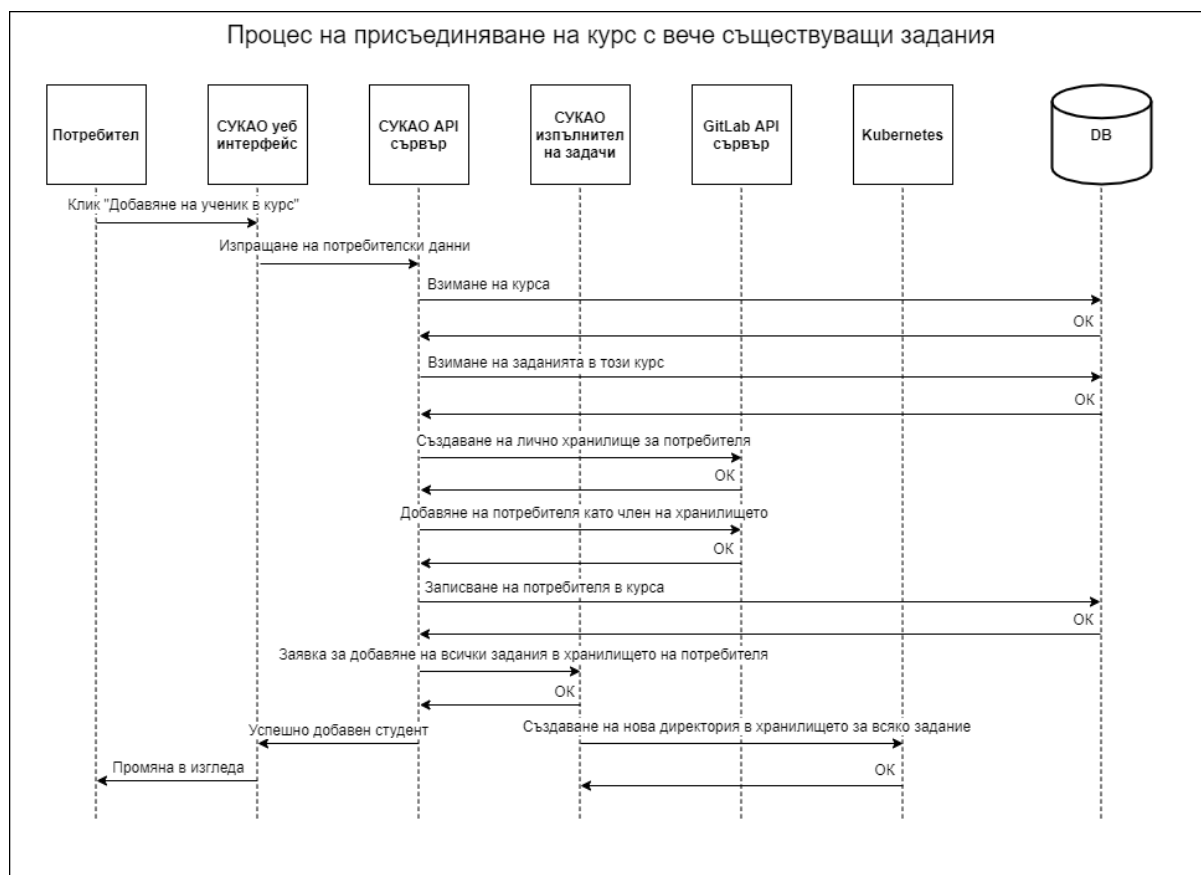
Фигура 5 - Процес на създаване на курс - диаграма на последователност

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 всички компоненти на системата, включително Kubernetes API сървърът и GitLab API сървърът. Той започва с натискането на един бутон в веб 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, като междувременно добавя задачата в опашката. Задачата включва създаването на Kubernetes Pod чрез Kubernetes API сървъра. В него се пуска контейнер, който тегли решението на студента, тестовите на лектора, компилира ги, форматира резултатите и ги извежда на стандартния изход. Изпълнителят на задачи ги извлича и ги запазва в базата от данни и променя статуса на завършен.



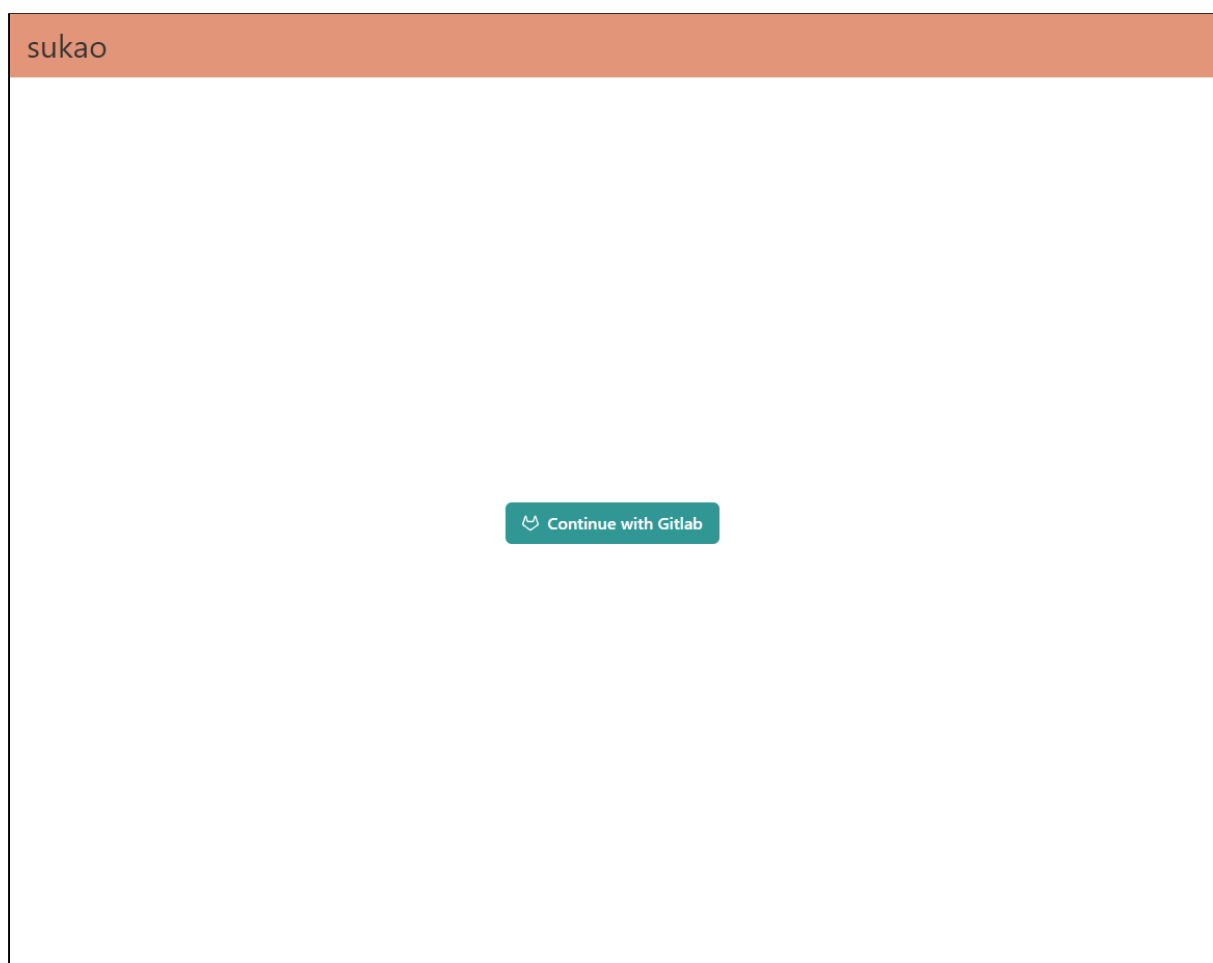
Фигура 7 - Процес на присъединяване на курс с вече съществуващи задания - диаграма на последователност

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, лекторът 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 студентът 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, контейнерът може да добави директориите, съдържащи условията на заданията.

Реализиране на уеб интерфейса

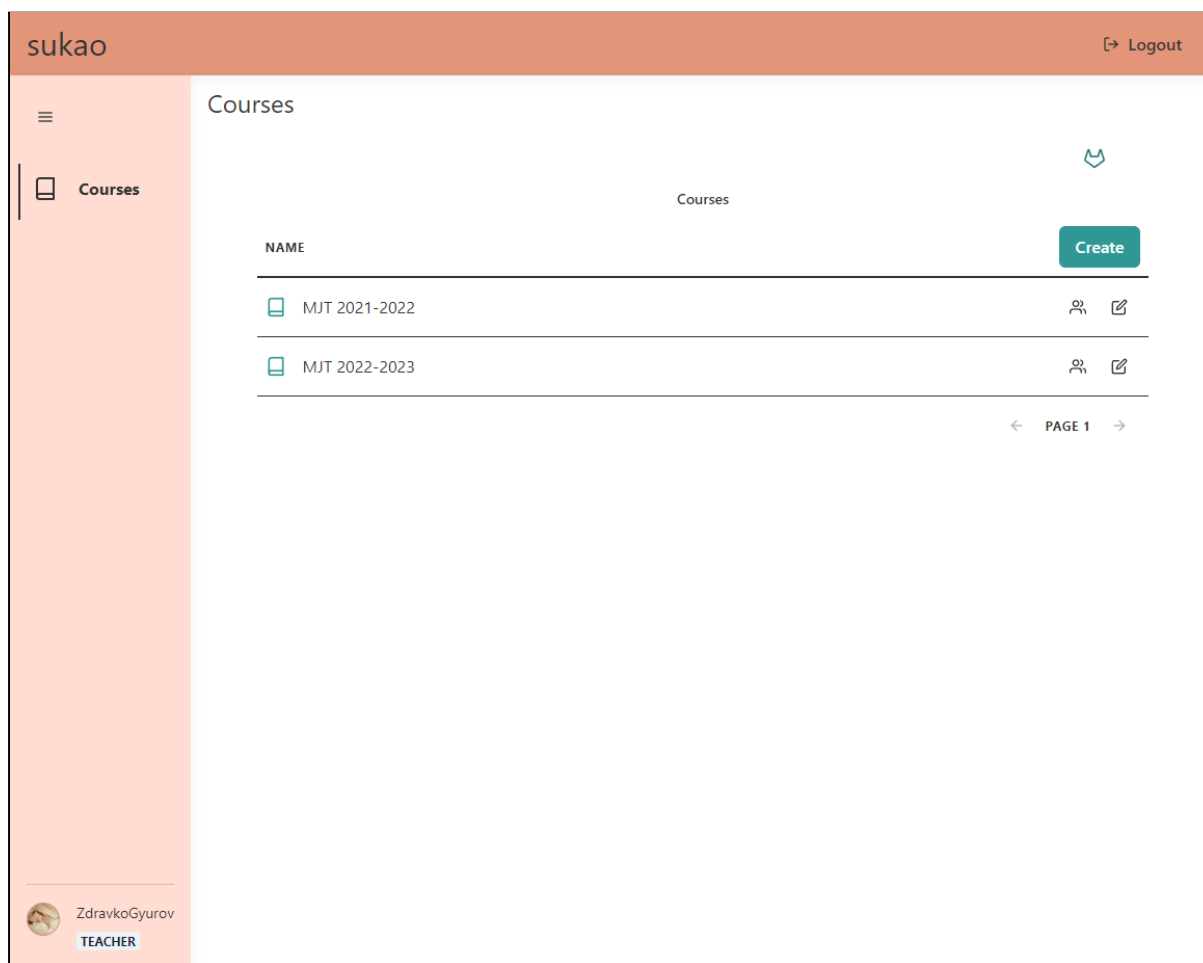
За разработването на уеб интерфейса е използвана JavaScript библиотеката React, докато за изграждането на красив и ефектен дизайн е използвана библиотеката Chakra UI, която предоставя градивните елементи.



Фигура 8 - Начална страница на СУКАО уеб интерфейса

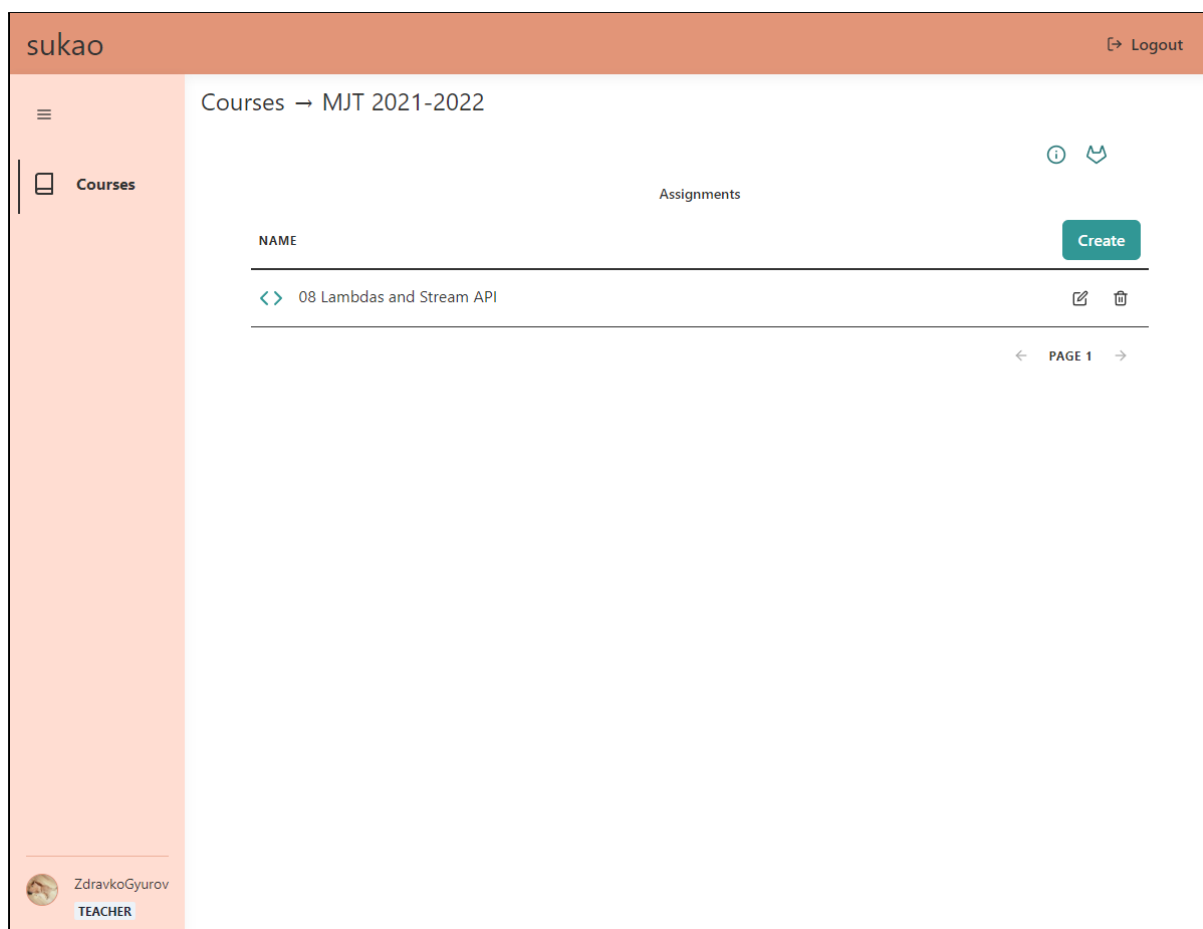
На фигура 8 се вижда страницата, с която се посреща всеки гост потребител. Навигирането към други изгледи е забранено и опитите за тази дейност завършат в пренасочване обратно към тази страница, тъй като единственото нещо, което е позволено на тази роля, е да се удостовери. Страницата съдържа

само името на системата и един бутон “Продължете с Gitlab”, който започва процеса от Фигура 4.



Фигура 9 - Страница с всички курсове в уеб интерфейса

След успешен вход в системата, потребителите с роля “Учител” виждат изгледа от фигура 9. Той включва странична навигация и бутон за изход, които се срещат на всяка страница освен тази за удостоверяване. Навигацията съдържа хипервръзки, водещи към различните функционалности на уеб интерфейса и информация за потребителя - неговото име, роля и профилната му снимка. В центъра на всеки изглед се намира най-важната информация, в случая списък с всички курсове, в които членува потребителя. Заради наличието на учителски права, са показани допълнителни бутони, позволяващи създаване нов курс, редактирането на вече съществуващ - промяна на име и описание и добавяне на студенти. Кликане на някой курс води до страница, съдържаща всички задания в курса - изобразено на фигура 10.



Фигура 10 - Страница с всички задания в веб интерфейса

Фигура 10 изобразява страница, съдържаща всички задания в определен курс. По подобие на изгледа от фигура 9, тази също е минимална, съдържаща малък брой бутони и е лесна за използване. В средата има лист с имената на заданията, които при клик водят до страница с всички предадени решения от текущия потребител за избраното задание - изобразено на фигура 11. Тук отново се виждат бутони за редакция на задание, служещи за смяна на името и описанието, което е в Markdown формат, позволяващ писането на богат, подреден и стилизиран текст. Има бутон за създаване на ново задание и също бутон, водещ към GitLab хранилището на потребителя, за да може той бързо да намери, къде трябва да си публикува кода.

sukao

Logout

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

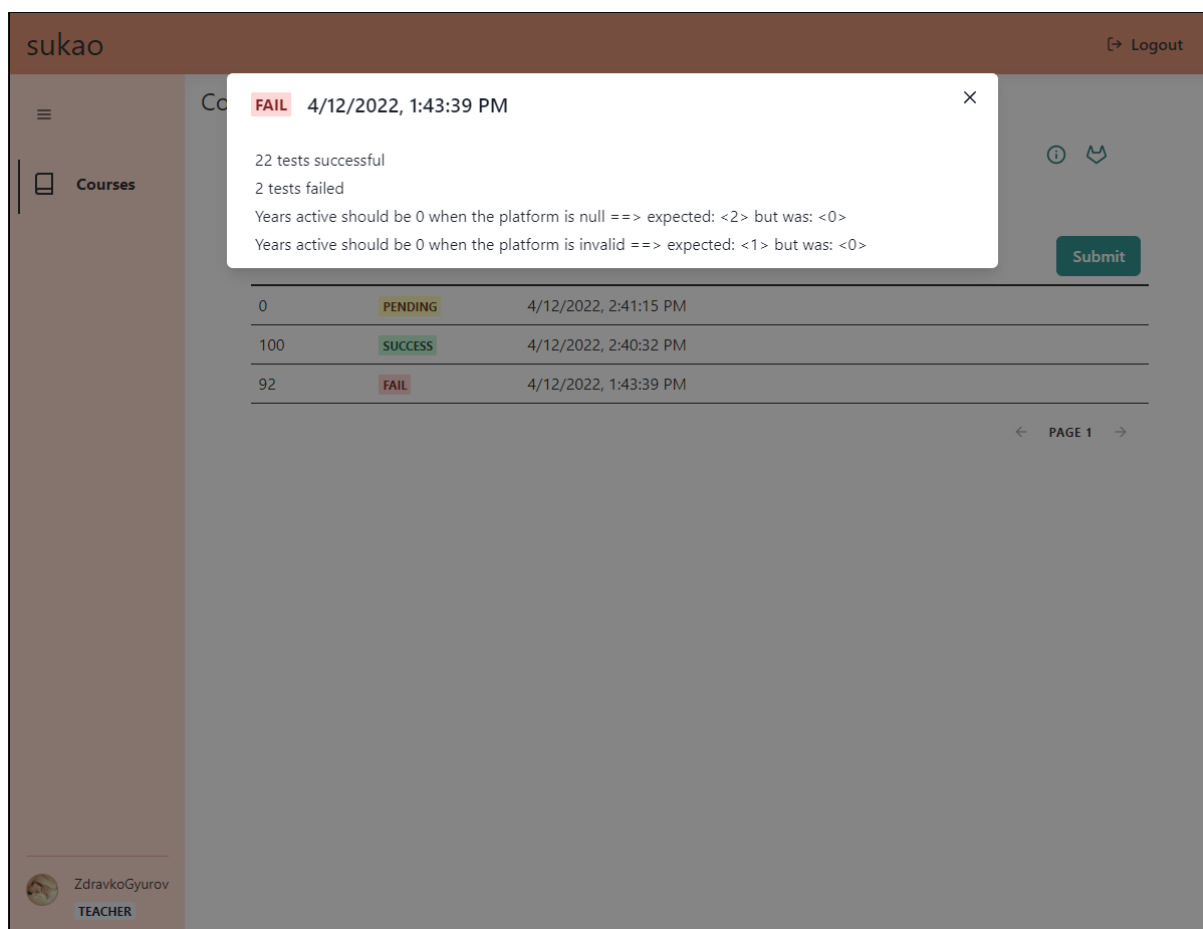
→

Zdravko Gyurov

TEACHER

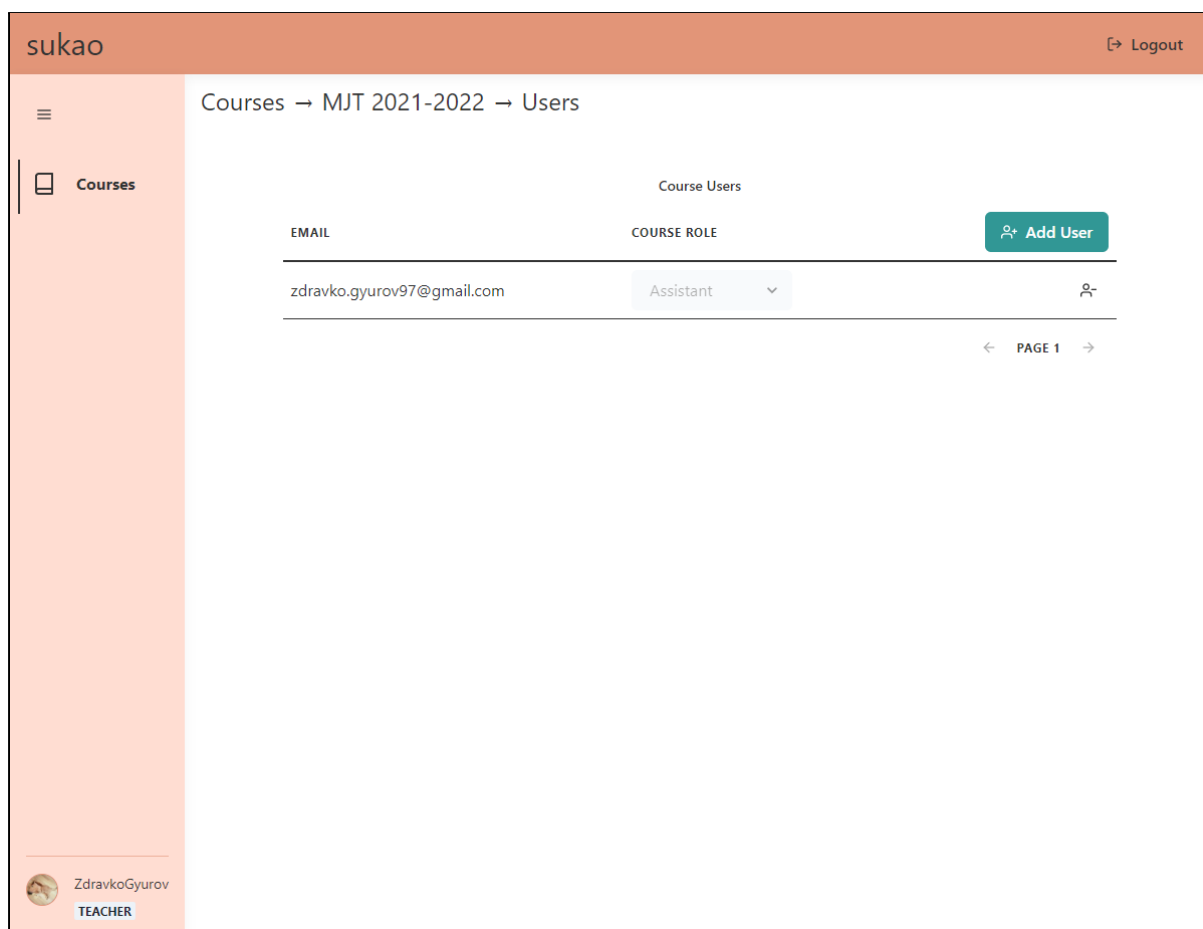
Фигура 11 - Страница с предадените решения в веб интерфейса

На фигура 11 е показана страница с предадените решения на студент за заданието от фигура 10. Този изглед е малко по-различен от предните два, но все пак има подобен вид и усещане. Отново най-отгоре има бутони за информация за заданието и хипервръзка към GitLab. В центъра се намират данни за резултатите в табличен вид, подредени по дата - най-скорошните са най-отгоре. Първата колона показва изкараните точки, които също могат да се кликат, водейки до изгледа от фигура 12. Втората колона съдържа значките за състоянието на предадените решения. При насрочване на оценяване, заявката е с жълта значка - "Обработка се". След като приключи с обработката, двата възможни резултати са червена значка - "Неуспех" и зелена значка - "Успех". Третата и последна колона дава информация за точната дата и час, в която е предадено решението и служи за сортиране.



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

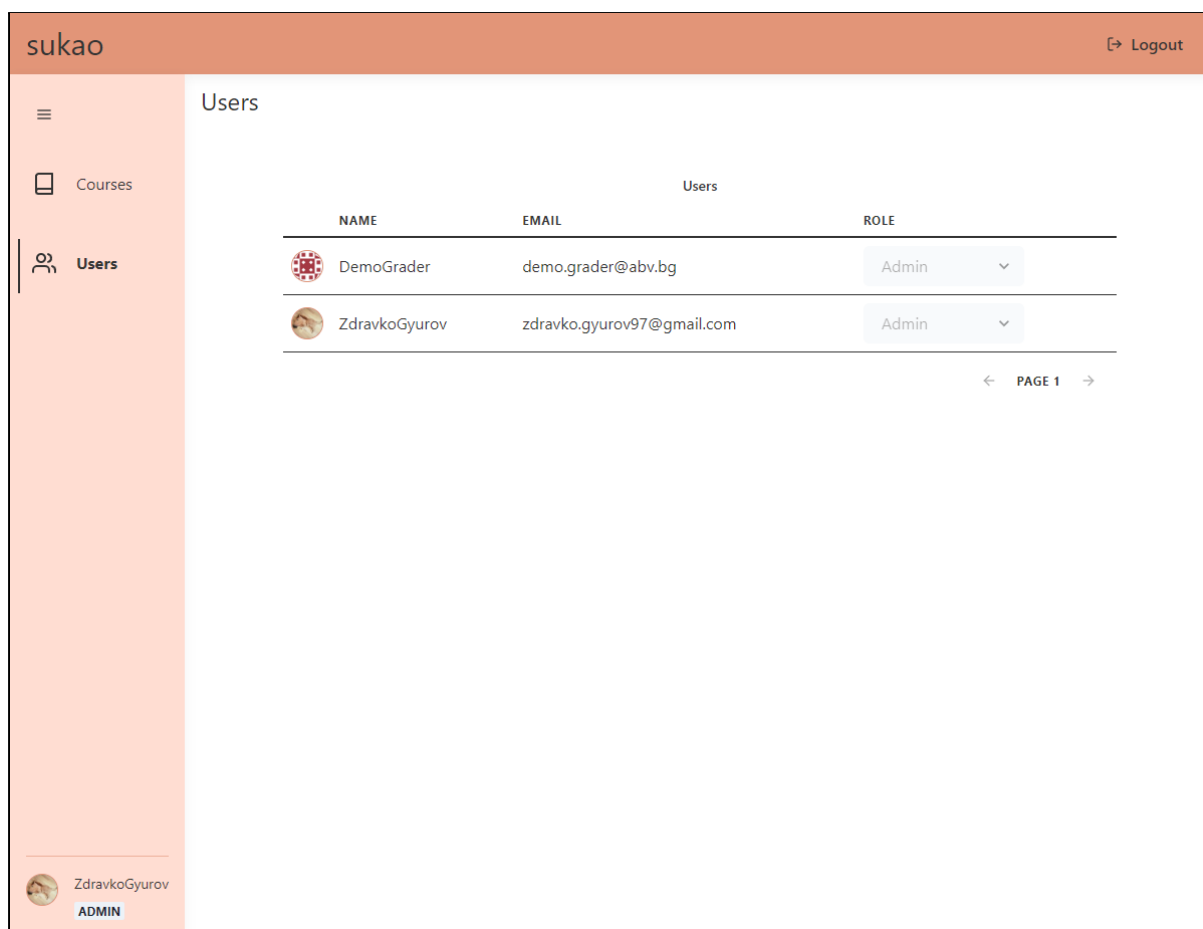
Фигура 12 цели да покаже обратната връзка от системата за някое предадено решение. Тя се извежда в малко прозорче, което изскача над страницата от фигура 11 и включва статуса на избраната заявка и датата в която е направена. В допълнение, излизат данни за броя на всички успешни и неуспешни тестове, като за всеки неуспешен тест, има съобщение, написано от лектора, даващо кратка информация на студента, къде може да му е грешката.



Фигура 13 - Страница с всички потребители в даден курс в уеб интерфейса, достъпна само за учители

Страницата, която е изобразена на фигура 13, показва всички потребители в даден курс. До нея може да се стигне от изгледа с всички курсове чрез натискане на специален бутон, достъпен само за преподаватели.

Функционалностите, които се предоставят тук са добавяне на студент в курса, премахване на студент от курса и промяна на ролята на член на курса. След създаване на нов курс, тук лекторът може да добави всички студенти и да направи част от тях асистенти, за да му помагат в процеса на обучение.



Фигура 14 - Страница с всички потребители в уеб интерфейса, достъпна само за администратори

Последният основен изглед на уеб интерфейса, е страницата с всички потребители, която е достъпна само за администратори. Дейностите за управление на курс са поверени на самите лектори, които са създали съответния курс, а регистрирането в системата става напълно автоматично, без нуждата от ръчна намеса. Поради тази причина, администраторите имат една единствена цел - да дават учителски права на ниво система. На фигура 14 се вижда списък, съдържащ имената на всички потребители на системата, техния имейл и падащо меню, позволяващо промяна на роля на избран потребител.

Реализиране на API сървъра

Структура

API сървърът на SUKAO е написан на Go. Той е сравнително нов език, разработен от Google, чийто силни страни са лекота на четене и писане, богатата стандартна библиотека, бързодействие, ефикасност откъм използване на процесорно време и памет и бързо разработване на конкурентен код.

Исходният код е разделен на 3 слоя:

1. Слой за работа с базата от данни;
2. Слой, съдържащ основната бизнес логика, валидации и други;
3. Слой, грижещ се за HTTP спецификите - обработване на заявки, създаване на бисквитки и други.

Първо, слой 1 има функция за връзка с базата, което създава специален клиент с конфигурирани максимално изчакване при връзка, максимално изчакване при заявка и максимален брой паралелни връзки. След това има методи, които използват този клиент и са абстракция над SQL заявките. Те се използват от втория слой за по-лесна работа.

Второ, слой 2 представлява основна част от кодовата база. Той използва информацията от слой 3 като входни данни и изпълнява заявките. Има два вида логика в този слой - вътрешна за приложението и такава, която изисква връзка с друг сървър (GitLab API сървър или API сървър на компонента за изпълняване на задачи).

Трето, слой 3 се грижи за всичко свързано с HTTP - преглеждат се бисквитките за информация за потребителя, прочита се JSON тялото на заявката и се превръща в Go структури. За разлика от REST тялото, вторият слой разбира от тези структури и ги обработва. Накрая се връща резултат и третия слой го преобразува в HTTP отговор с тяло и статус код.

Приложно-програмни интерфейси

Вход в системата

HTTP метод: GET

Път: /login/oauth/gitlab

Примерен вход:

Начинът за консумация на този път е с просто пренасочване.

Примерен изход:

Следвайки всички HTTP пренасочвания, потребителя се озовава на GitLab страницата за вход.

Продължение на OAuth2 потока за вход в системата, извиква се само от GitLab

HTTP метод: GET

Път: /login/oauth/gitlab/callback?code=<...>

Примерен вход:

Подава се само параметър на заявка

Примерен изход:

Създаване на бисквитки с токени за оторизация към системата и пренасочване към потребителския интерфейс.

Вземане на информация за текущия потребител

HTTP метод: GET

Път: /userInfo

Примерен вход:

Подава се единствено бисквитка с токен.

Примерен изход:

Статус 200 OK

```
{  
  "email": "dummyuser@gmail.com",  
  "name": "dummyuser",  
  "avatarUrl": "dummyavatar",  
  "gitlabId": "12345678",  
  "roleName": "Student"  
}
```

Вземане на информация за всички потребители в системата

HTTP метод: GET

Път: /user

Примерен изход:

Статус 200 OK

```
[
  {
    "email": "dummyuser@gmail.com",
    "name": "dummyuser",
    "avatarUrl": "dummyavatar",
    "gitlabId": "12345678",
    "roleName": "Student"
  },
  ...
]
```

Промяна на ролята на потребител в системата

HTTP метод: PATCH

Път: /userInfo

Примерен вход:

```
{
  "email": "dummyuser@gmail.com",
  "roleName": "Teacher"
}
```

Примерен изход:

Статус 200 OK

```
{
  "email": "dummyuser@gmail.com",
  "name": "dummyuser",
  "avatarUrl": "dummyavatar",
  "gitlabId": "12345678",
  "roleName": "Teacher"
}
```

Обновяване на токена за достъп до системата

HTTP метод: POST

Път: /token

Примерен вход:

Подава се единствено бисквитка с токен.

Примерен изход:

Създава се нов токен за оторизация с обновено време на изтичане.

Изход от системата

HTTP метод: DELETE

Път: /logout

Примерен вход:

Подава се единствено бисквитка с токен.

Примерен изход:

Премахва се бисквитката с токена.

Създаване на курс

HTTP метод: POST

Път: /course

Примерен вход:

```
{  
  "name": "my-course1",  
  "description": "my-course-description-1",  
  "gitlabName": "my-course1"  
}
```

Примерен изход:

Статус 201 Created

```
{  
  "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",  
  "name": "my-course1",  
  "description": "my-course-description-1",  
  "gitlabId": "12345678",  
  "gitlabName": "my-course1",  
  "creatorEmail": "dummyuser@gmail.com",  
  "createdOn": "2022-04-11T12:46:53.724837Z",  
  "lastEditedOn": "2022-04-11T12:46:53.724837Z"  
}
```

Вземане на информация за всички курсове, в които членува текущият потребител

HTTP метод: GET

Път: /course

Примерен изход:

Статус 200 OK

```
[
  {
    "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
    "name": "my-course1",
    "description": "my-course-description-1",
    "gitlabId": "12345678",
    "gitlabName": "my-course1",
    "creatorEmail": "dummyuser@gmail.com",
    "createdOn": "2022-04-11T12:46:53.724837Z",
    "lastEditedOn": "2022-04-11T12:46:53.724837Z"
  },
  ...
]
```

Вземане на информация за определен курс

HTTP метод: GET

Път: /course/{id}

Примерен изход:

Статус 200 OK

```
{
  "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
  "name": "my-course1",
  "description": "my-course-description-1",
  "gitlabId": "12345678",
  "gitlabName": "my-course1",
  "creatorEmail": "dummyuser@gmail.com",
  "createdOn": "2022-04-11T12:46:53.724837Z",
```



```
    "lastEditedOn": "2022-04-11T12:46:53.724837Z"
  }
```

Редактиране на име и описание на курс

HTTP метод: PATCH

Път: /course/{id}

Примерен вход:

```
{
  "description": "my-course-description-changed-1"
}
```

Примерен изход:

Статус 200 OK

```
{
  "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
  "name": "my-course1",
  "description": "my-course-description-changed-1",
  "gitlabId": "12345678",
  "gitlabName": "my-course1",
  "creatorEmail": "dummyuser@gmail.com",
  "createdOn": "2022-04-11T12:46:53.724837Z",
  "lastEditedOn": "2022-04-11T12:46:53.724837Z"
}
```

Изтриване на курс

HTTP метод: DELETE

Път: /course/{id}

Примерен изход:

Статус 204 No Content

Създаване на задание

HTTP метод: POST

Път: /assignment

Примерен вход:

```
{
```

```
    "name": "my-assignment1",
    "description": "my-assignment-description-1",
    "gitlabName": "my-assignment1"
}
```

Примерен изход:

Статус 201 Created

```
{
  "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e88",
  "name": "my-assignment1",
  "description": "my-assignment-description-1",
  "authorEmail": "dummyuser@gmail.com",
  "courseId": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
  "gitlabName": "my-assignment1",
  "createdOn": "2022-04-11T12:46:53.724837Z",
  "lastEditedOn": "2022-04-11T12:46:53.724837Z"
}
```

Вземане на информация за всички задания в курс

HTTP метод: GET

Път: /assignment

Примерен изход:

Статус 200 OK

```
[
  {
    "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e88",
    "name": "my-assignment1",
    "description": "my-assignment-description-1",
    "authorEmail": "dummyuser@gmail.com",
    "courseId": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
    "gitlabName": "my-assignment1",
    "createdOn": "2022-04-11T12:46:53.724837Z",
    "lastEditedOn": "2022-04-11T12:46:53.724837Z"
  },
  ...
]
```

]

Вземане на информация за конкретно задание в определен курс

HTTP метод: GET

Път: /assignment/{id}

Примерен изход:

Статус 200 OK

```
{
  "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e88",
  "name": "my-assignment1",
  "description": "my-assignment-description-1",
  "authorEmail": "dummyuser@gmail.com",
  "courseId": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
  "gitlabName": "my-assignment1",
  "createdOn": "2022-04-11T12:46:53.724837Z",
  "lastEditedOn": "2022-04-11T12:46:53.724837Z"
}
```

Редактиране на име и описание на задание

HTTP метод: PATCH

Път: /assignment/{id}

Примерен вход:

```
{
  "description": "my-assignment-description-changed-1"
}
```

Примерен изход:

Статус 200 OK

```
{
  "id": "b12cfa6f-a6b5-41da-b225-bb12a0476e88",
  "name": "my-assignment1",
  "description": "my-assignment-description-changed-1",
  "authorEmail": "dummyuser@gmail.com",
  "courseId": "b12cfa6f-a6b5-41da-b225-bb12a0476e87",
  "gitlabName": "my-assignment1",
}
```

```
    "createdOn": "2022-04-11T12:46:53.724837Z",  
    "lastEditedOn": "2022-04-11T12:46:53.724837Z"  
}
```

Изтриване на задание

HTTP метод: DELETE

Път: /assignment/{id}

Примерен изход:

Статус 204 No Content

Предаване на решение за тестване

HTTP метод: POST

Път: /submission

Примерен вход:

```
{  
  "assignmentId": "f46a6301-5518-4ba9-be61-98c8e744b977"  
}
```

Примерен изход:

Статус 202 Accepted

```
{  
  "id": "06955361-af3c-46d9-a38a-2865bfa20fd1",  
  "result": "",  
  "points": 0,  
  "submissionStatusName": "Pending",  
  "submitterEmail": "dummyuser@gmail.com",  
  "submittedOn": "2022-04-24T16:03:29.293831Z",  
  "assignmentId": "f46a6301-5518-4ba9-be61-98c8e744b977"  
}
```

Вземане на резултатите от тестовете на всички предадени решения за определено задание

HTTP метод: GET

Път: /submission?assignmentId=...

Примерен изход:

Статус 200 OK

```
[
  {
    "id": "885fa230-b515-4b93-992d-b5288439f98b",
    "result": "22 tests successful\n2 tests failed\nYears active should be 0
when the platform is null ==\u003e expected: \u003c2\u003e but was:
\u003c0\u003e\nYears active should be 0 when the platform is invalid
==\u003e expected: \u003c1\u003e but was: \u003c0\u003e\n",
    "points": 92,
    "submissionStatusName": "Fail",
    "submitterEmail": "dummyuser@gmail.com",
    "submittedOn": "2022-04-12T10:43:39.871366Z",
    "assignmentId": "f46a6301-5518-4ba9-be61-98c8e744b977"
  },
  ...
]
```

Вземане на резултатите от тестовете на предадено решение за определено задание

HTTP метод: GET

Път: /submission/{id}

Примерен изход:

Статус 200 OK

```
{
  "id": "885fa230-b515-4b93-992d-b5288439f98b",
  "result": "22 tests successful\n2 tests failed\nYears active should be 0 when
the platform is null ==\u003e expected: \u003c2\u003e but was:
\u003c0\u003e\nYears active should be 0 when the platform is invalid
==\u003e expected: \u003c1\u003e but was: \u003c0\u003e\n",
  "points": 92,
  "submissionStatusName": "Fail",
  "submitterEmail": "dummyuser@gmail.com",
  "submittedOn": "2022-04-12T10:43:39.871366Z",
  "assignmentId": "f46a6301-5518-4ba9-be61-98c8e744b977"
```

```
}
```

Добавяне на потребител в курс

HTTP метод: POST

Път: /userCourse

Примерен вход:

```
{  
  "courseId": "0a13dc11-d088-4cf7-aca7-0a61ee372696",  
  "courseRoleName": "Student",  
  "userEmail": "dummyuser@gmail.bg",  
}
```

Примерен изход:

Статус 201 Created

```
{  
  "userEmail": "dummyuser@gmail.bg",  
  "courseId": "0a13dc11-d088-4cf7-aca7-0a61ee372696",  
  "courseRoleName": "Student"  
}
```

Вземане на информация за ролите на всички потребители в даден курс

HTTP метод: GET

Път: /userCourse?courseId=0a13dc11-d088-4cf7-aca7-0a61ee372696

Примерен изход:

Статус 200 OK

```
[  
  {  
    "userEmail": "dummyuser@gmail.com",  
    "courseId": "0a13dc11-d088-4cf7-aca7-0a61ee372696",  
    "courseRoleName": "Student"  
  },  
  ...  
]
```

Промяна на ролята на потребител в курс

HTTP метод: PUT

Път: /userCourse

Примерен вход:

```
{
  "courseId": "0a13dc11-d088-4cf7-aca7-0a61ee372696",
  "courseRoleName": "Assistant",
  "userEmail": "dummyuser@gmail.bg",
}
```

Примерен изход:

Статус 201 OK

```
{
  "userEmail": "dummyuser@gmail.bg",
  "courseId": "0a13dc11-d088-4cf7-aca7-0a61ee372696",
  "courseRoleName": "Assistant"
}
```

Премахване на потребител от курс

HTTP метод: DELETE

Път:

/userCourse?userEmail=dummyuser@gmail.bg&courseId=0a13dc11-d088-4cf7-aca7-0a61ee372696

Примерен изход:

Статус 204 No Content

Реализиране на компонента за изпълняване на тестове

Структура

Компонента на SUKAO за изпълняване на задачи представлява уеб услуга с малък брой приложно-програмни интерфейси, която също е написана на Go. Тя има идентична структура на тази на API сървъра - 3 слоя:

1. Слой за работа с базата от данни;

2. Слой, съдържащ основната бизнес логика, валидации и други;
3. Слой, грижещ се за HTTP спецификите - обработване на заявки, създаване на бисквитки и други.

Основните разлики са, че тук не е имплементирано JWT удостоверяване, тъй като този компонент ще се извиква само и единствено от API сървъра и те ще са предпазени в частна мрежа. В допълнение, разработен е Golang пакет за изпълняване на задачи в Docker контейнери, намиращи се в Kubernetes Pod. Отделно има и пакет, имплементиращ шаблона за дизайн - множество (pool) от работници, което е постигнато лесно и интуитивно с примитивите за конкурентност, присъщи за Go.

Приложно-програмни интерфейси

Създаване на заявка за тестване на предадено решение от студент

HTTP метод: POST

Път: /job

Примерен вход:

```
{  
  "submissionId": "885fa230-b515-4b93-992d-b5288439f98b"  
}
```

Примерен изход:

Статус 202 Accepted

Създаване на заявка за добавяне на работни директории в хранилищата на новонавлезлите студенти в курс

HTTP метод: POST

Път: /assignment

Примерен вход:

```
{  
  "courseGroup": "12345678",  
  "assignmentPaths": "assignment1;assignment2;assignment3",  
  "gitlabUsernames": "dummyuser1;dummyuser2;dummyuser3"  
}
```


Реализиране на Kubernetes шаблоните

За внедряването на системата се използва Kubernetes и Helm, които правят менажирането на Docker контейнери лесно и автоматично. Kubernetes шаблоните са конфигурационни файлове в .yaml формат, представляващи в повечето случаи един ресурс или няколко малки, групирани смислено. При нужда да се приложат всички шаблони наведнъж, се срещат затруднения, които се решават от Helm пакетите, които просто групират няколко конфигурационни файла и ги прилагат наведнъж. Друг проблем, който се решава по този начин, е промяната на конфигурациите. Без Helm, при нужда за промяна, трябва да се променят директно шаблоните, което допълнително се усложнява, поради липсата на контрол на версиите. Използвайки го, те стават параметризирани и всички променливи конфигурации се намират на едно централно място - values.yaml файл. В допълнение, при инсталиране на Helm пакет, може директно чрез командата да се презаписват настройките.

Системата за управление на курсове и автоматично оценяване зависи на 2 Helm пакета - пакет (postgresql), внедряващ PostgreSQL база от данни и друг (ingress-nginx), който добавя поддръжка за Ingress ресурсите и активира стабилизатор на натоварването. Първият не е задължителен и може да се замени от база от данни като услуга, предоставена от някой облачен доставчик.

Шаблоните на системата са групирани по тип.

Започвайки от Configmap ресурсите - API сървър, компонента за изпълняване на задачи и уеб интерфейса си имат по един. Те са набор от конфигурации (примерно на какъв хост и порт да вървят приложенията), запазени в чист текст, тъй като не са чувствителни данни.

Второ, има Secret ресурси, които са почти същите като Configmap-овете, с основната разлика, че те са кодирани и предвидени за чувствителни данни като пароли или сертификати. В случая тук се пазят паролата за базата, клиентски идентификатор и клиентска тайна за GitLab, токени за достъп и други.

Трето, всеки компонент си има Kubernetes Deployment и Service. Ресурсите за внедряване посочват колко реплики ще има всеки компонент, който контейнер ще върви в Pod-овете и кои конфигурации ще се монтират. Знаейки за тези ресурси и портовете на контейнерите в тях, Услугите могат да създадат домейн име за тях в рамките на частната мрежа. Така компонентите могат да се извикват един друг на конкретен хост - името на съответната услуга, а не да разчитат на променливи IP адреси, които се сменят при рестартиране на всеки Pod.

До този момент приложението все още не е достъпно за външния свят. Това се променя от Ingress ресурса, който позволява трафика към уеб приложението чрез посочването на HTTP път, име на Kubernetes услуга и порт, към който да се пренасочва трафика.

Допълнително, има шаблон за Kubernetes Job, който стартира контейнер, извършващ миграциите на базата в началната фаза от инсталирането на Helm пакета.

Последно, когато компонента за изпълняване на задачи стартира контейнери, той ги пуска в Kubernetes Pod, което е възможно, заради допълнително добавената му Kubernetes роля, която включва създаване на pod-ове, наблюдаване на pod-ове (следене на статуса им) и четене на логовете на pod-ове (нужно за прочитане на резултатите от тестовете).

5. Експерименти и анализ на резултатите

За анализиране на създадената система, е направен ръчен тест на всички функции. Проверени са всички функционалности нужни на един преподавател да влезе в системата, да си направи нов курс с ново задание в него. В последствие са публикувани тестове за заданието и примерно решение на учителя, с които той да провери дали всичко се компилира и изпълнява коректно. Финално е предадено решението за получаване на обратна връзка от системата.

Експериментът е проведен на чисто нова инсталация на системата и е описан в стъпки:

1. Преподавателят влиза в началната страница на системата, където избира да продължи с GitLab акаунт.
2. Той въвежда своя GitLab акаунт или се логва чрез някой от другите предоставени доставчици на идентичност.
3. Преподавателят е пренасочен към изгледа за курсове на системата, като в този момент все още е с права на студент.
4. По същият начин, администратор влиза в системата, но тъй като потребителският идентификатор на този акаунт е предварително конфигуриран като администратор, още при първоначален вход в системата, той има административни права.
5. Администраторът отива на изгледа с потребители.
6. Администраторът дава учителски права на преподавателския акаунт и получава съобщение за успешно изпълнена операция.
7. При следващ вход в системата, преподавателят вече има обновени права, които му дават възможност да види бутона за създаване на курс в потребителския интерфейс.
8. Учителят започва процеса на създаване на курс като натиска бутона и след това въвежда име, GitLab име на група и описание на курса и потвърждава действията си.
9. Системата изкарва съобщение за успех и в списъка с курсове вече се вижда новосъздаденият курс.
10. Преподавателят натиска курса и е препратен към друг изглед, в който има информация за курса и връзка към GitLab групата.
11. Потребителят създава и задание по същия начин като процеса за създаване на курс - задава се име на заданието, GitLab име на хранилището за заданието и описание, след което се потвърждава.
12. Системата изкарва съобщение за успех и в листа със задания вече се вижда новосъздаденото задание.
13. Учителят натиска заданието и е препратен към друг изглед, в който има информация за заданието и връзка към работното GitLab хранилище.
14. Учителят отива в GitLab, чрез натискане на гореспоменатия бутон, и вижда почти празно хранилище съдържащо единствено README.md файл - условието на заданието.

15. Преподавателското хранилище е предвидено да съдържа тестовете за заданието, които ще се изпълняват с всяко публикувано студентско решение.
16. Първоначално, при създаване на ново задание, той качва тестовете и своето решение за проверка за потенциални компилационни или други грешки в програмата си или в самите тестове.
17. Учителят се връща в СУКАО изгледа на заданието и предава решението си за оценяване чрез натискане на един бутон.
18. В листа от предадени решения се появява нов запис с 0 точки, статус "В обработване" и дата на предаване.
19. След кратко време (по-малко от 5 секунди) преподавателят презарежда страницата и вижда, че решението му е проверено - има 100 точки и статус "Успех", с което той е сигурен, че задачата и тестовете, които е създал са готови за даване на студентите.

6. Заключение

Постигнати резултати

Проучени са системи за компютърно подпомагано обучение, които или предоставят възможност за управление на курсове, или възможност за автоматично оценяване на решенията на студентите. След подробни изследвания обаче, разгледаните съществуващи алтернативи са с очевидни пропуски. Основните недостатъци са незадоволително бързодействие, ограничения на безплатния план - малък брой заявки или малък брой позволени членове на курсовете, неудобен уеб интерфейс, неинтуитивен процес за предаване на решения и други.

Проектирана е система за създаване на курсове, задания и предаване на решения с възможност за тяхното автоматично оценяване. Изградена е нормализирана структура на базата от данни с идеята, че ще се пазят голямо количество данни, с които трябва да се работи бързо. Наличието на индекси

подпомага търсенето по уникални полета, а ограниченията от външните ключове осигуряват интегритет и консистентност на данните. Създадена е Kubernetes архитектура на системата, включваща три компонента - уеб интерфейс, API сървър, компонент за извършване на асинхронни задачи. Функционалността на приложението е разпределена семантично между тях.

Реализирана е разпределена система, работеща в контейнери на Kubernetes за да предостави висока достъпност, надеждност и сигурност. Тя включва модерен и интуитивен уеб интерфейс, написан на React, REST API сървър, написан на Go, който е предпазен от JWT удостоверяване и действа като посредник към базата от данни и в допълнение прави заявки към GitLab, за да създава групи и хранилища за студентите там и компонент за тестване на предадените решения от студентите, който ги изпълнява в Docker контейнери с ограничени ресурси и права, за да предостави сигурна и високопроизводителна среда.

Приноси

Научни

Проучени са научни работи, свързани с компютърно подпомагано обучение, учебни машини и автоматично оценяване на задания по програмиране. Изготвен е сравнителен анализ на съществуващи системи като са съпоставени техните предимства и недостатъци. Направено е сравнение по функционални и нефункционални изисквания.

Научно-приложни

Проектирана е разпределена архитектура на Kubernetes за системата, която предоставя издръжливост срещу претоварване от трафик, висока достъпност и надеждност и толерантност към грешки.

Приложни

Разработена е системата - уеб интерфейса, API сървър и компонента за изпълняване на задачи, Kubernetes архитектурата на системата и структурата на базата от данни.

Апробация

Системата ще се ползва за управлението и автоматичното оценяване на решенията на студентите от ФМИ от изборната дисциплина Съвременни Java технологии 2022-2023.

Насоки за бъдеща работа; Перспективи

За момента комуникацията в частната мрежа между отделните API компоненти става с директна синхронна HTTP заявка. Този метод е предразположен към грешки, ако получателят в момента не е на разположение или просто се забави с отговора. Една възможност за бъдещо развитие е добавянето на опашка за съобщения като rabbitMQ. Това е брокер на съобщения, който предоставя висока надеждност и сигурност на данните.

Друг проблем при комуникацията между компонентите написани на Go, е че те си комуникират по HTTP протокола, а не HTTPS. Трябва и двата компонента да реализират TLS протокола, като взаимно се верифицират един друг.

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.

Calza, Julio C., and Jose M. Del Alamo. "Programming assignments automatic grading: Review of tools and implementations." 2013, p. 10.

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.

Sharma, Rishu. "Computer Assisted Learning - A Study." April 2017, p. 4,
<http://ijaret.com/wp-content/themes/felicity/issues/vol4issue2/rishu.pdf>.
Accessed 11 5 2022.

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.