УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ
И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ

КАТЕДРА „КОМПЮТЪРНИ НАУКИ”
СПЕЦИАЛНОСТ „КОМПЮТЪРНИ НАУКИ ”

ДИПЛОМНА РАБОТА

на тема:

ПРОЕКТИРАНЕ И РАЗРАБОТВАНЕ НА ПИЛОТНА СИСТЕМА ЗА АВТОМАТИЗИРАНО ОЦЕНЯВАНЕ НА ПРОГРАМЕН КОД В УНИВЕРСИТЕТСКА ОБРАЗОВАТЕЛНА СРЕДА

Дипломант:
Светослав Михайлов
редовно обучение
Ф.№ 46592р

Научен ръководител:.....
(гл. ас. д-р Л. Гоцев)

София
2025

РЕЗЮМЕ

Михайлов, С. Проектиране и разработване на пилотна система за автоматизирано оценяване на програмен код в университетска образователна среда. Научен ръководител гл. ас. д-р Л. Гоцев. С. 2025. Катедра „Компютърни науки“. Бакалавърска програма „Компютърни науки“. УНИБИТ. 66 с. Брой източници – 29, приложения – няма.

Целта на дипломната работа е проектиране, разработване и оценяване на пилотна система за автоматизирана проверка и оценка на програмен код, която да подобри учебния процес и ефективността при оценяване в университетска среда.

Предмет на изследването са онлайн платформите и технологиите за автоматизирана проверка и оценяване на програмен код.

Обект на изследването е разработването на пилотна уеб-базирана система за автоматизирано оценяване на студентски задачи по програмиране, с интегриране на подходящи технологии и софтуерни инструменти.

За постигане на поставената цел са дефинирани следните **задачи:**

- Да се направи преглед и сравнителен анализ на съществуващи онлайн системи за автоматизирана оценка на програмен код.
- Да се анализира и проектира пилотната система.
- Да се разработи, тества и оцени пилотната система с използване на технологии като ASP.NET Core, React, PostgreSQL.

В изследването са приложени **методите** на систематичен преглед, сравнителен анализ, функционален и архитектурен дизайн, приложно разработване и тестване.

Резултатите показват ефективна работа на пилотната система, която автоматизира и оптимизира процеса на оценяване на студентския програмен код.

Ключови думи: автоматизирана оценка, програмен код, онлайн платформи, ASP.NET Core, React.

СЪДЪРЖАНИЕ

РЕЗЮМЕ 1

УВОД 4

ПЪРВА ГЛАВА. ПРЕГЛЕД НА ПРЕДМЕТНАТА ОБЛАСТ 6

1.1 Системи за автоматизирано оценяване на програмен код 6

1.1.1 Функционален модел 7

1.1.2 Архитектурен модел 8

1.2 Преглед на актуални решения за автоматизирано оценяване 13

1.2.1 Leetcode 13

1.2.2 Hackerrank 16

1.2.3 SoftUni Judge 17

1.3 Сравнителен анализ на актуалните решения 17

ИЗВОДИ ПО ПЪРВА ГЛАВА 21

ВТОРА ГЛАВА. АНАЛИЗ И ПРОЕКТИРАНЕ НА ПИЛОТНА СИСТЕМА 22

2.1 Анализ 22

2.1.1 Цел и обхват 22

2.1.2 Заинтересовани страни 23

2.1.3 Функционални изисквания 24

2.1.4 Нефункционални изисквания 28

2.1.5 База от данни 30

2.1.6 Интерфейс и потребителско изживяване 31

2.1.7 Ограничения 32

2.2 Архитектура на системата 34

2.2.1 REST 34
2.2.2 Компоненти на системата 37
2.2.3 Поток на данните 40
ИЗВОДИ ПО ВТОРА ГЛАВА 42
ТРЕТА ГЛАВА. РАЗРАБОТВАНЕ И ОЦЕНКА НА ПИЛОТНАТА СИСТЕМА 43
3.1 Технологично осигуряване 43
3.1.1 Сървърна част (Back-end) 44
3.1.2 Клиентска част (Front-end) 49
3.1.3 База данни 51
3.2 Оценка на системата 53
3.2.1 Тестови сценарии 53
3.2.2 Оценка на ефективността 54
3.3 Анализ на резултатите и развитие 55
3.3.1 Идентифицирани области за подобрение 55
3.3.2 Развитие 57
ИЗВОДИ ПО ТРЕТА ГЛАВА 59
ЗАКЛЮЧЕНИЕ 60
ИЗПОЛЗВАНА ЛИТЕРАТУРА 61
СПИСЪК НА ФИГУРИТЕ 65
СПИСЪК НА СЪКРАЩЕНИЯТА 66

УВОД

Изборът на тема е мотивиран от нарастващите потребности в съвременното образование в областта на компютърните науки. С нарастващия обем от задания, свързани с програмиране, традиционните методи за проверка и оценяване стават неефективни, трудоемки и времеемки. В този контекст системите за автоматизирани оценка на програмния код предлагат значителен потенциал за подобряване на учебния процес чрез осигуряване на бърза, обективна и прозрачна обратна връзка както за студентите, така и за преподавателите.

Актуалността на темата се определя от необходимостта за повишаване качеството и ефективността на преподаването и оценяването на програмиране, което е от съществено значение за подготовката на бъдещи специалисти в сферата на софтуерните технологии. Развитието на такива автоматизирани системи е от особена важност както за университетската среда, така и за професионалните практики, включващи технически интервюта и оценка на квалификации.

Целта на дипломната работа е проектиране, разработване и оценяване на пилотна система за автоматизирана проверка и оценка на програмен код, която да подобри учебния процес и ефективността при оценяване в университетска среда.

Предмет на изследването са онлайн платформите и технологиите за автоматизирана проверка и оценяване на програмен код.

Обект на изследването е разработването на пилотна уеб-базирана система за автоматизирано оценяване на студентски задачи по програмиране, с интегриране на подходящи технологии и софтуерни инструменти.

За постигане на поставената цел са дефинирани следните **задачи:**

- Да се направи преглед и сравнителен анализ на съществуващи онлайн системи за автоматизирана оценка на програмен код.
- Да се анализира и проектира пилотната система.
- Да се разработи, тества и оцени пилотната система с използване на технологии като ASP.NET Core, React, PostgreSQL.

В рамките на изследването са приложени **методи** като систематичен преглед на съществуващи решения, сравнителен анализ на архитектури и технологии, както и практическа разработка и тестване на пилотна система.

Използваните **източници** за изследването включват научни публикации, статии, онлайн ресурси и техническа документация, публикувани предимно в последните пет години. Основният акцент е поставен върху актуални технологии и решения, които са били предмет на систематичен и сравнителен анализ.

Структурата на дипломната работа е организирана в три основни глави. Първа глава представя теоретичните основи и функционален и архитектурен преглед на актуални системи за автоматизирана оценка на програмен код. Във Втора глава е извършен детайлен анализ и проектиране на пилотната система, дефинирайки нейните изисквания и архитектура. Трета глава описва процеса на разработка, технологичното осигуряване и оценката на пилотната система, както и анализ на получените резултати. Работата завършва с изводи по всяка глава, заключение и използвана литература.

Резултатите показват, че създаденото уеб приложение отговаря на предварително дефинираните изисквания, демонстрира висока степен на функционална завършеност и предлага реална приложимост в университетска среда.

Дипломната работа е **предназначена** за студенти, преподаватели и администратори в университетската образователна среда, като предоставя практическо решение, което може да бъде интегрирано и използвано за повишаване на ефективността на учебния процес, намаляване на субективността при оценяването и подобряване на качеството на преподаване и усвояване на знанията по програмиране.

ПЪРВА ГЛАВА. ПРЕГЛЕД НА ПРЕДМЕТНАТА ОБЛАСТ

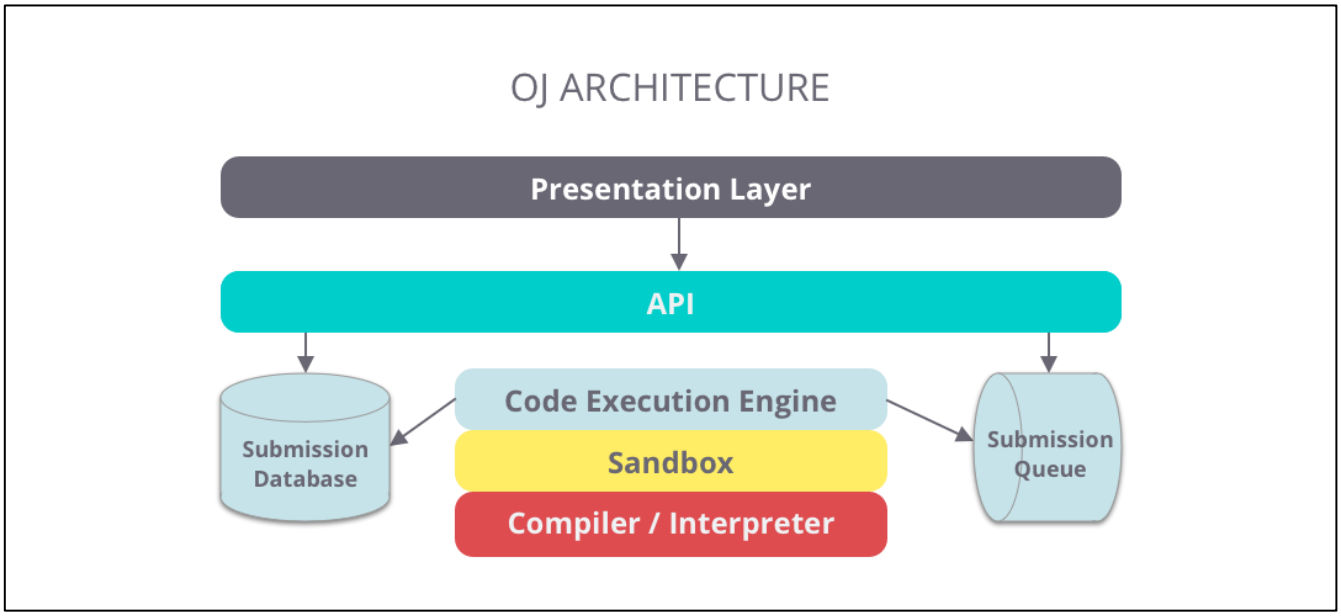
Настоящата глава представя преглед на утвърдени платформи за автоматизирано оценяване на програмен код, които намират приложение в образователни и професионални среди. Анализирани са архитектурните особености, предимства и ограничения на решения като LeetCode, HackerRank и SoftUni Judge. В края на главата е представен сравнителен анализ между разгледаните системи, който очертава ключовите фактори, повлияли върху избора на архитектурни и технологични решения за разработката на пилотната система в дипломната работа.

1.1 Системи за автоматизирано оценяване на програмен код

С разширяването на обучението по програмиране и необходимостта от ефективно оценяване на знанията и уменията на обучаемите, все по-широко разпространение намират онлайн системите за автоматизирано оценяване (**OJS** – Online Judge Systems).

Тези специализирани софтуерни решения са уеб-базирани инструменти, които автоматично приемат, компилират, изпълняват и оценяват програмния код, написан от потребителя, по зададени критерии. Оценяването се извършва предимно чрез "black-box testing", при което потребителските решения се сравняват с предварително зададени входове и очаквани изходи, като коректността се определя автоматично. OJS осигуряват бърза, надеждна и безпристрастна обратна връзка, намалявайки натоварването на преподавателите [1], [2].

OJS позволяват обективна и мащабируема проверка на знанията без нужда от ръчна намеса, като се използват както в образователна среда за обучение и изпитване на студенти, така и в индустрията – например при подбор на ИТ кадри.

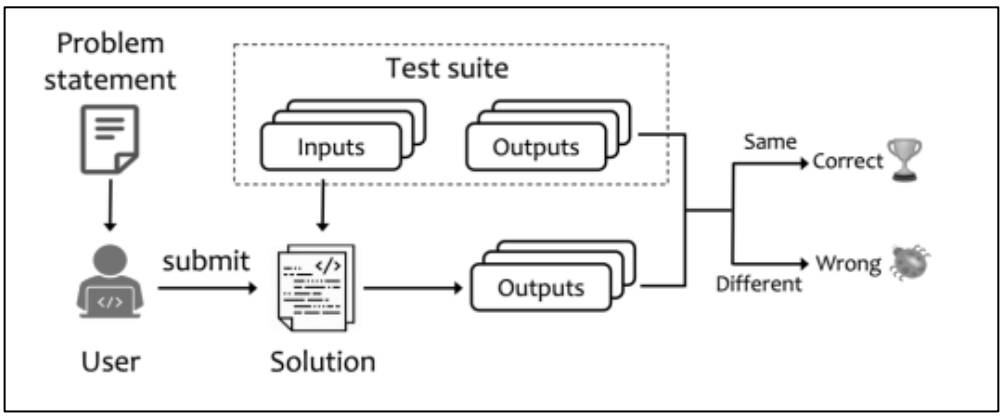


Фигура 1.1 Опростена архитектура на OJS [2]

В следващите раздели са разгледани основните принципи на функциониране на този тип системи, както и тяхната архитектура, с акцент върху ключовите компоненти, участващи в процеса на оценяване.

1.1.1 Функционален модел

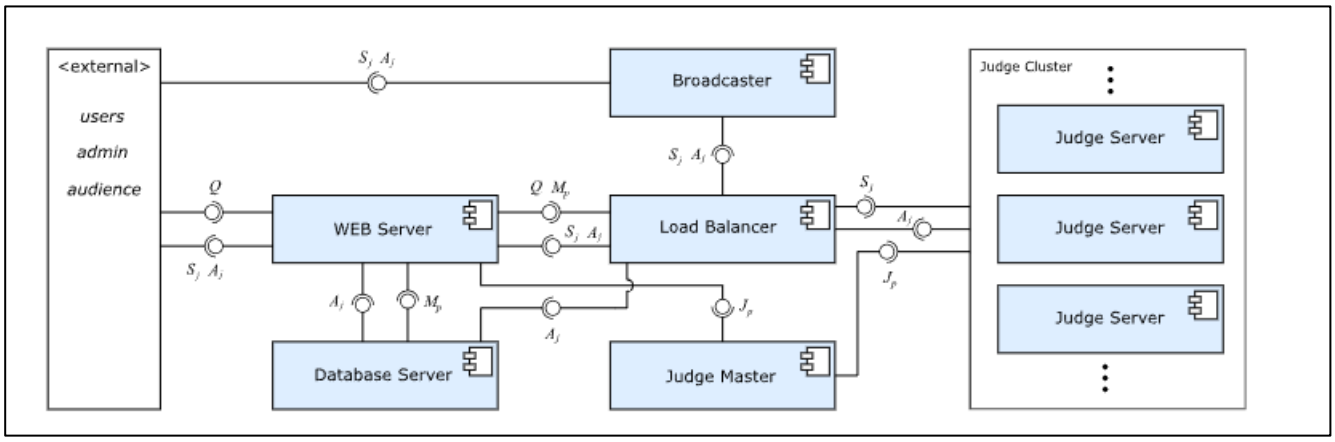
Задачата на онлайн системата за автоматизирано оценяване се състои основно от формулировка на проблема и предварително подготвен набор от тестове. Изложението на проблема въвежда задачите за програмиране и уточнява форматите на входните и изходните данни (например поредица от цели числа). Освен това той предоставя входните ограничения (напр, входът трябва да бъде цяло положително число, ако представлява число на обекти). Задачата често предоставя примерни входове и изходни данни, за да се ориентират потребителите. Потребителите могат да определят програмния език, който искат да използват, да напишат кода за решаване на задачата, и след това да изпратят своите решения през платформата. След като решението е подадено, платформата компилира решенията (ако е нужно) в съответствие с посочения език и преценява правилността на решението, като използва предварително подготвен набор от тестове. Предвиденият набор от тестове не е видим за потребителите [3].



Фигура 1.2 Работен процес на типична OJS [2]

1.1.2 Архитектурен модел

OJS се реализира и внедрява въз основа на архитектура, като се вземат предвид както функционалните, така и нефункционални изисквания. Услугата от OJS, ориентирана към няколко нефункционални изисквания, не може да бъде реализирана чрез монолитна архитектура. Вместо това множество специализирани компоненти работят заедно, за да осигурят качествено поддържана функционалност.



Фигура 1.3 Основни компоненти на OJS [4]

На Фиг. 1.3 е показана диаграма на компонентите, която дава представа за архитектурата, в която основните компоненти на OJS комуникират с данните, свързани с оценката. По принцип тя се основава на концепцията за микросървисна

архитектура (Micro-service architecture) за по-лесно интегриране на леки, свободно свързани компоненти.

Уеб сървър: Архитектурата на уеб сървъра се състои от множество сървъри, всеки от които е предназначен за предоставяне на специфични API-та. Той служи като интерфейс между външни системи (като потребителски терминали и администратори) и Online Judge системата (OJS). Тези сървъри се справят с различни задачи на бизнес логиката, като например регистрация на потребители, автентификация, сърфиране и подаване на данни [4].

Сървър за бази данни: Сървърът за бази данни също се състои от множество сървъри за бази данни, които са свързани с уеб сървърите. Сървърите за бази данни взаимодействат с уеб сървърите и с load balancer-ите на чрез частни API. В него се съхранява цялата информация, свързана с услугата, като например набор от проблеми, информация за резултатите от решенията и регистрирани потребители. Всички кодове на решения и статистически данни също се натрупват и управляват [4].

Judge Master: Специализиран сървър в архитектурата на системата за онлайн оценяване (Online Judge система, OJS), предназначен за управление на данните за оценяване в клъстера на оценяващите сървъри (Judge Cluster). Той включва механизъм за непрекъсната интеграция, което го прави ключов компонент за управление и разпределение на данните за оценяване. Вместо да изпраща големи обеми данни за оценяване (Jp) между сървъра за оценяване и load balancer-a, Judge Master съхранява всички данни за задачите и разпределя техни копия до всички оценяващи сървъри преди активирането на съответната задача.

Когато администратор добавя или актуализира данни за оценяване (Jp), процесът на непрекъсната интеграция осигурява консистентност между всички оценяващи сървъри.

Този процес включва:

1. Генериране на подмножества: Подмножества (I) се създават от елементи (G).
2. Валидиране: Всяко подмножество (I) се валидира спрямо зададените ограничения чрез валидатори (V).
3. Референтни изходи: Генерират се референтни решения (ri) и изходи (outi) за всички тестови случаи.
4. Проверка: Изходите на всички решения се сравняват, за да се гарантира коректност.

След като данните за оценяване преминат всички проверки, те се предоставят на всички оценяващи сървъри. Този подход гарантира консистентно и точно управление на данните в клъстера на оценяващите сървъри [4].

Load Balancer: Load balancer-ът е основен компонент на системата за онлайн оценяване, който служи като посредник между уеб сървърите и сървърите за оценяване. Освен това, той комуникира със сървърите за бази данни и системата за излъчване (broadcaster). Основната му роля е да управлява разпределението на натоварването за подадените заявки, като гарантира, че те се обработват чрез сигурни опашки без загуба или дублиране на данни.

Основни отговорности на load balancer-а:

1. Планиране на заявките: Всяка заявка за оценяване (Q) получава уникален идентификатор на оценката (j) и свързаните с нея метрики (Mp). Заявката (Q) се преобразува в (Sj) и се насочва към съответния сървър за оценяване чрез частно API. Очакващите заявки се управляват в специфични за сървъра опашки.
2. Управление на резултатите: Обработва резултатите от оценяването (Aj), получени от сървърите за оценяване.
3. Изпращане на известия: Load balancer-ът изпраща известия за (Sj) в опашките и съответните (Aj) до потребителите чрез различни

канали. Известията за (Aj) се изпращат също така към уеб сървъра, сървъра за бази данни и системата за излъчване.

Този процес гарантира ефективно управление на заявките, обработката на резултатите и предоставянето на актуална информация в реално време в цялата система [4].

Broadcaster: Времето за изчакване на заявките за оценяване варира в зависимост от времето за изпълнение на програмните кодове в опашките. Това може да промени реда на оценяване, докато множество потребители чакат резултатите. Broadcaster-ът е проектиран да уведомява асинхронно потребителите за промени в състоянието на заявките, които се управляват от load balancer-а.

Уведомленията за промени в състоянието на заявките (Sj) и техните резултати (Aj) се изпращат чрез три канала:

1. Първи канал: Публичен API, който предоставя постоянна информация за състоянието от базата данни в отговор на конкретни параметри на заявка, изпратени от уеб сървърите.
2. Втори канал: Друг публичен API, който предоставя информация за състоянието на наскоро изпратените заявки в опашките, управлявани в паметта на load balancer-а.
3. Трети канал: Система за уведомяване в реално време, базирана на асинхронна комуникация, осъществявана от broadcaster-а. Този канал не съхранява данни, а се фокусира върху бързото изпращане на уведомления.

Broadcaster-ът намалява напрежението у потребителите по време на изчакване и същевременно намалява броя на заявките към уеб сървъра, което от своя страна намалява натоварването на load balancer-а [4].

Judge Cluster: Load balancer-ът комуникира със сървърите за оценяване в клъстера на оценяващите сървъри (Judge Cluster). Всеки сървър за оценяване изпълнява поредица от задачи за заявка. Първо получава заявка за оценяване (Sj),

след това я изпълнява въз основа на данните за оценка (Jp), и накрая изпраща резултата (Aj). Клъстерът на оценяващите сървъри трябва да бъде физически (или поне виртуално) изолиран, така че само упълномощени процеси да могат да се свързват със сървъра за оценяване чрез частен API. Важно е да се предотврати изтичане на данни поради злонамерени или неволни операции.

За да се подобри общото качество на услугата за оценяване, клъстерът на сървърите за оценяване се формира с няколко сървъра за оценяване, като се вземат предвид следните точки:

1. За консистентност: Всеки сървър за оценяване използва специализирани хардуерни ресурси, които са изключително предназначени за изпълнение на даден код за решение.
2. За незабавност: Няколко различни кода за решение могат да се изпълняват едновременно на различни сървъри за оценяване.
3. За надеждност: Сървърите за оценяване си помагат един на друг като паралелни машини [4].

Описаният архитектурен модел на OJS, базиран на микросървисна архитектура, демонстрира висока степен на мащабируемост, надеждност и гъвкавост. Чрез ясното разграничаване на отговорностите между компоненти като Уеб сървър, Сървър за бази данни, Judge Master, Load Balancer, Broadcaster и Judge Cluster, системата осигурява ефективно управление на целия процес от подаване на код до получаване на резултати. Тази декомпозиция позволява независимо развитие и внедряване на отделните модули, улеснява поддръжката и осигурява възможност за бърза адаптация към нарастващото натоварване и променящите се изисквания. Разбирането на тези фундаментални архитектурни принципи е ключово за оценката на съществуващи решения и за успешното проектиране на нова пилотна система за автоматизирано оценяване на програмен код.

1.2 Преглед на актуални решения за автоматизирано оценяване

След като разгледахме теоретичните основи и общите архитектурни принципи на онлайн системите за автоматизирано оценяване (OJS), настоящата подточка ще представи детайлен функционален и архитектурен преглед на няколко актуални и широко разпространени платформи. Анализът на тези утвърдени решения като LeetCode, HackerRank и SoftUni Judge е от съществено значение за идентифициране на техните силни страни, ограничения и специфични подходи към автоматизираната оценка на програмен код. Разбирането на вече реализирани концепции ще послужи като важна основа за проектирането на пилотната система в контекста на настоящата дипломна работа.

1.2.1 Leetcode

LeetCode е водеща онлайн платформа, създадена през 2015 г. в Сан Диего, Калифорния, която служи като инструмент за подготовка и оценка в областта на програмирането. Тя е широко използвана от студенти, професионални програмисти и работодатели за усвояване, практикуване и оценяване на програмни умения.

Платформата представлява хибриден ресурс, съчетаващ тестове за кодиране, общностен форум и специализиран инструмент за подготовка за технически интервюта, подпомагайки потребителите в развитието на техните умения и участието в състезания по програмиране [5].

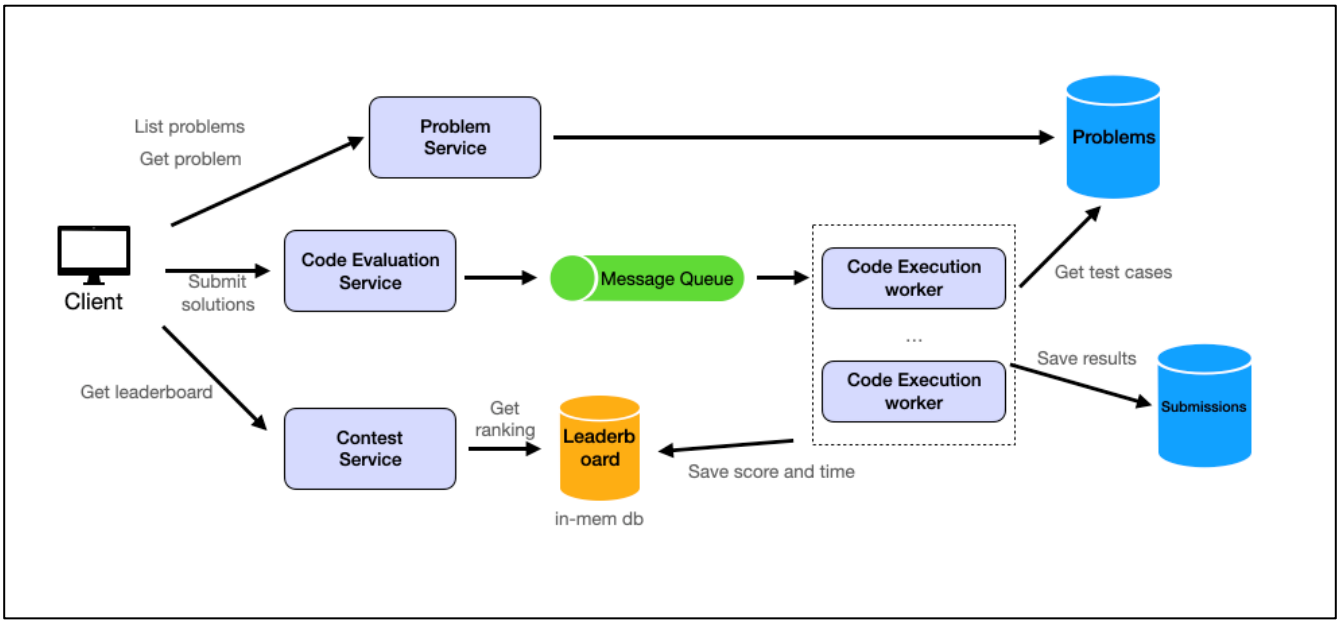
LeetCode класифицира задачите в различни категории, включително алгоритми и структури от данни, системен дизайн, SQL, машинно обучение и изкуствен интелект.

Платформата е силно насочена към подготовка за интервюта, предлагайки специализирани секции за развитие на техническите умения на кандидатите. Пример за това е "LeetCode 75" – внимателно подбран набор от въпроси, целящ да осигури широка подготовка за различни типове задачи, срещани по време на интервюта. Подобна по концепция е и секцията "Top Interview 150", която съдържа

въпроси с различна степен на трудност, подбрани за покриване на широк спектър от теми и подготовка за разнообразни предизвикателства.

Освен това LeetCode предоставя секции като "SQL 50" за позиции, изискващи познания по SQL (напр. Data Engineer, Data Scientist). За начинаещи е разработена специална програма "30 Days of JavaScript", която помага на потребители с по-малък опит да развият логическото си мислене, докато усвояват основите на езика чрез ежедневни задачи.

На Фиг. 1.4 е представено примерно архитектурно решение.



Фигура 1.4 Примерна системна архитектура аналогична на Leetcode [6]

Главните функционалности в Leetcode са следните:

- Разглеждане на задача,
- Решение на задача и
- Участие в състезание.

Разглеждане на задача: Потребителите имат възможност да преглеждат описания на проблеми, примери, ограничения и списък с всички налични задачи. Дизайнът на тази функционалност е относително прост, като достъпът до конкретна

задача се осъществява чрез директна GET заявка с подаден идентификатор на проблема.

Подаване на решение за задача: За да се осигури ефективно и сигурно оценяване на кода при висока паралелност (напр. 10 000 едновременно участващи потребители, подаващи средно по 20 решения), системата използва "Code Evaluation Service" като входна точка. Тази услуга отговаря за получаването на кода от потребителите и иницииране на процеса на оценяване. За справяне с големия обем данни и пиковете в трафика, между процеса на подаване и действителното изпълнение на кода се въвежда опашка за съобщения (message queue) като буфер. Когато потребител подаде решение, "Code Evaluation Service" изпраща съобщение с подробностите за подаването (идентификатор на потребителя, идентификатор на проблема и код) към опашката. От другата страна на опашката са разположени няколко работни групи за изпълнение на код (workers). Тези работници непрекъснато проверяват опашката за нови подавания, извличат необходимите тестови случаи от базата данни "Problems", изпълняват кода в защитен контейнер (sandboxed environment) и записват резултатите [6].

Участие в състезание: Платформата позволява на потребителите да участват в състезания по програмиране. Типично състезание е с фиксирана продължителност от 2 часа и се състои от четири въпроса. Резултатът се изчислява въз основа на броя на решените въпроси и времето, необходимо за тяхното решаване, като резултатите се показват в реално време. Самото състезание по същество представлява набор от четири задачи с ограничен времеви прозорец. За управление на състезанията може да се използва централизирана "Contest Service".

След като потребителят изпрати решение, то се оценява от "Code Evaluation Service", а резултатите се изпращат към "Contest Service", която записва резултата на потребителя в базата данни. За осигуряване на актуализации в реално време за класацията, предвид че данните се променят често, се използва база данни

в паметта (in-memory database) като Redis, която е подходяща за висока производителност и обработка на данни в реално време [6].

1.2.2 Hackerrank

Като утвърдена платформа, HackerRank е създадена през 2012 г. и е сходна по функционалност с LeetCode, предоставяйки разнообразни инструменти за обучение, практикуване на програмиране и подготовка за технически интервюта. Тя включва обширен набор от задачи, категоризирани по нива на трудност, обхващащи теми като алгоритми, структури от данни, бази данни, изкуствен интелект и други. Това я прави подходяща както за начинаещи, така и за опитни специалисти, стремящи се да усъвършенстват своите умения.

HackerRank предлага и специализирани обучителни пътеки, които помагат на потребителите да се фокусират върху конкретни технологии или професионални роли, като например анализ на данни или разработка на уеб приложения.

Една от ключовите характеристики на платформата е програмата "Get Certified". Тя дава възможност на потребителите да получат безплатни сертификати за своите умения в конкретни области. Сертификациите са налични на различни нива – базово, междинно и напреднало – и представляват ценен начин за демонстрация на компетенциите пред работодатели.

Популярността на HackerRank се дължи на универсалността ѝ – тя е предпочитан избор както сред индивиди, които желаят да развият кариерата си в ИТ сектора, така и сред компании, които използват платформата за оценка и подбор на таланти чрез практически тестове и задачи. Това я превръща в цялостен инструмент за обучение, сертификация и професионално развитие.

1.2.3 SoftUni Judge

SoftUni Judge е онлайн платформа, разработена от Софтуерния университет (SoftUni), която подпомага програмистите в усъвършенстването на техните умения чрез решаване на задачи и автоматично оценяване на програмния код. Отличителна черта на платформата е нейният отворен код, достъпен в GitHub, което позволява прозрачност и възможност за общностни приноси.

Въпреки че е основно предназначена за студентите и курсистите на SoftUni и играе важна роля в учебния процес на университета, тя е достъпна и за всеки, който желае да практикува програмиране.

Платформата предоставя богата колекция от задачи с различна сложност, покриващи теми като основи на програмирането, алгоритми, структури от данни и други. Всяка задача е съпроводена с предварително зададени тестови случаи, които SoftUni Judge използва за автоматична оценка на качеството на написания код. След подаване на решение, системата извършва проверка дали кодът преминава успешно всички тестове и предоставя подробна обратна връзка – относно коректността, наличието на грешки или проблеми с ефективността.

Тази платформа е интегрална част от учебните курсове на SoftUni, като се използва за проверка на домашни работи, упражнения и изпитни задачи, значително улеснявайки процеса на обучение. SoftUni Judge осигурява на потребителите незабавна оценка на кода им, което е голямо предимство за тези, които се стремят към бързо и ефективно учене и развитие.

1.3 Сравнителен анализ на актуалните решения

След като беше направен детайлен обзор на функционалните и архитектурни особености на водещи онлайн Judge системи като LeetCode, HackerRank и SoftUni Judge, настоящият раздел цели да представи сравнителен анализ на тези платформи. Този подход ще даде възможност за идентифициране на общи черти, ключови разлики, както и на силните страни и ограниченията на всяко решение.

Фокусът ще бъде поставен върху аспекти като поддържани програмни езици, типове задачи, методи за оценка, възможности за обратна връзка, потребителски интерфейс и архитектурни принципи.

Изведените заключения от този анализ ще послужат като критична основа за избора на архитектурни и технологични решения при проектирането и разработката на пилотната система, предмет на настоящата дипломна работа.

За целите на сравнителния анализ, разгледаните платформи са оценени по набор от ключови критерии, които са от съществено значение за функционалността и приложимостта на една система за автоматизирано оценяване на програмен код. Тези критерии включват: целева аудитория, основни функционалности, поддържани програмни езици, качество на потребителското преживяване (UX/UI), вид на достъпа (платен/безплатен), както и специфични предимства и основни недостатъци.

Въпреки че всички платформи предлагат автоматизирана оценка на код, съществуват ясни диференциации в тяхната основна насоченост и реализация. LeetCode е силно ориентиран към подготовка за технически интервюта, предлагайки широк спектър от алгоритмични задачи и симулации.

HackerRank, от своя страна, е по-универсален, обслужвайки както индивидуални разработчици, така и компании за оценка на таланти, с по-широк обхват от теми и сертификационни програми. SoftUni Judge е тясно интегриран с образователния процес на Софтуерния университет, фокусирайки се върху структурирано обучение и осигурявайки практическо приложение на курсовия материал, макар и с по-ограничен езиков обхват и по-базов интерфейс [7].

Детайлното съпоставяне по горепосочените критерии е представено в приложената таблица.

Таблица 1.1 Сравнителен анализ на водещи платформи за автоматизирано оценяване на програмен код

Критерий	**LeetCode**	**HackerRank**	**SoftUni Judge**
Целева аудитория	Средно напреднали и напреднали програмисти, търсещи работа	Широка аудитория – от начинаещи до професионалисти и работодатели	Начинаещи и учащи се в структуриран образователен контекст
Основни функционалности	Задачи по алгоритми и структури от данни, интервю симулации	Предизвикателства по различни теми, включително AI и бази данни	Оценка на задачи от курсове, теоретични и практически упражнения
Поддържани езици	18 програмни езика (вкл. C++, Python, Java, др.)	Над 20 езика, включително редки като Perl и Swift	Около 5 езика, включително C#, Java, Python, част от техните пътеки
UX/UI качество	Модерен и интуитивен интерфейс	Добър интерфейс, но понякога объркващ за нови потребители	Базов интерфейс, по-подходящ за образователна среда
Вид достъп	Безплатен с платен премиум план	Безплатен с платен премиум план	Безплатен (при записване в курсове на SoftUni)
Други предимства	Симулиране на интервюта, статистики за напредъка	Използва се от работодатели за подбор, предлага сертификати	Ясни образователни пътеки, комбинира теория и практика
Основни недостатъци и ограничения	Ограничени обяснения за начинаещи, платен достъп до част от съдържанието	Нерядко неясни формулировки на задачи, трудност в дебъгването	Ограничен езиков избор, не толкова предизвикателна като конкурентите

Синтез на резултатите от сравнителния анализ: Анализът на LeetCode, HackerRank и SoftUni Judge подчертава, че докато всички платформи споделят основната цел за автоматизирана оценка на програмен код, те се различават значително в своята целева ориентация, обхват на функционалности и потребителско изживяване. LeetCode и HackerRank са по-универсални, предлагайки по-широк спектър от езици и по-богати възможности за напреднали потребители и професионална подготовка, често включващи премиум функции. За разлика от тях, SoftUni Judge е тясно специализирана за нуждите на академичното обучение, предоставяйки по-фокусирана среда с пряка връзка към учебни програми, но с по-ограничен обхват на езици и по-базов интерфейс. Тези различия са от ключово значение за формулирането на изискванията и архитектурния дизайн на пилотната система, тъй като те помагат да се идентифицират успешни подходи и да се избегнат потенциални ограничения при създаването на ново решение, адаптирано към университетска образователна среда.

ИЗВОДИ ПО ПЪРВА ГЛАВА

В първа глава беше извършен детайлен теоретичен преглед на съвременни онлайн системи за автоматизирано оценяване на програмен код (Online Judge Systems – OJS). Бяха задълбочено разгледани както принципите на функциониране на тези системи, така и архитектурната организация на основните им компоненти – от уеб сървъра и базата данни до специализираните модули за разпределение на задачи и сигурно изпълнение на код.

Извършеният функционален и архитектурен анализ на три утвърдени платформи – LeetCode, HackerRank и SoftUni Judge – разкри съществени разлики в техните целеви аудитории, обхвата на поддържаните програмни езици, интерфейсните решения и нивото на персонализирана обратна връзка. Представената сравнителна таблица позволи систематизиране на предимствата и ограниченията на всяка платформа, като очерта критични фактори при избора и проектирането на подобна система в университетски образователен контекст.

На базата на изведените заключения от този анализ, ще бъдат формулирани функционалните и нефункционалните изисквания към разработваната пилотна система, която е основен обект на изследване в следващата глава.

ВТОРА ГЛАВА. АНАЛИЗ И ПРОЕКТИРАНЕ НА ПИЛОТНА СИСТЕМА

В настоящата глава са представени анализът и концептуалното проектиране на пилотна система за автоматизирано оценяване на програмен код в университетска образователна среда. Разгледани са целта и обхватът на системата, идентифицирани са основните заинтересовани страни и са формулирани функционалните и нефункционалните изисквания към разработката. Особено внимание е отделено на структурата на базата от данни, изискванията към потребителския интерфейс и ограниченията, свързани с внедряването. Във втората част на главата се описват архитектурният модел, ключовите компоненти на системата и логиката на обмена на данни между тях, което ще послужи като основа за предстоящата техническа реализация.

2.1 Анализ

Създаването на ефективна онлайн система за автоматизирано оценяване изисква задълбочен предварителен анализ на нуждите, ограниченията и очакванията на бъдещите потребители. В следващите подточки ще бъдат детайлно разгледани основните цели на системата, обхватът на нейната функционалност, идентифицирането на заинтересованите страни, както и специфичните изисквания и други ключови аспекти, които ще залегнат в основата на нейното проектиране.

2.1.1 Цел и обхват

Основната цел при изграждането на този тип система е да оптимизира както учебния процес, така и процеса на оценяване от страна на преподавателите. Понастоящем в много курсове по програмиране студентите изпълняват задачи, които по същество са аналогични на тези в платформите от тип онлайн система за автоматизирано оценяване (Online Judge Systems – OJS). Въпреки това, при традиционния подход липсва незабавна и видима обратна връзка за студентите, а преподавателите са силно натоварени с ръчната проверка и оценка на стотици подадени работи. Чрез изграждането на такава система целият процес по оценяване

на знанията може да стане значително по-ефективен и ползотворен както за студентите, така и за преподавателите.

Важно е да се отбележи, че разработената система не цели да представлява напълно завършен комерсиален продукт, а по-скоро се класифицира като минимален жизнеспособен продукт (MVP – Minimum Viable Product). Това означава, че решението ще включва само основните функционалности, достатъчни за демонстрация на концепцията и за първоначално използване в реална университетска среда.

Първоначалният обхват на системата е насочен към университетска среда и ще включва: управление на задачи от страна на преподаватели и автоматична проверка на решени задачи чрез използване на предефинирани тестови случаи (test cases). От административна гледна точка, обхватът ще покрива управление на потребителски акаунти и цялостно администриране на платформата.

2.1.2 Заинтересовани страни

В процеса на проектиране и анализ на пилотната система бяха идентифицирани следните основни групи заинтересовани страни (stakeholders), чиито нужди и очаквания са ключови за нейното успешно функциониране:

- **Преподаватели:** Те изискват инструмент, който ефективно автоматизира процеса на проверка и оценка, намалява субективността и значително спестява време. Също така, за тях е от съществено значение да разполагат с интуитивен и лесен за използване интерфейс за създаване, управление и проследяване на задачи и студентски резултати.
- **Студенти:** От тази група се очаква да използват интуитивна, бърза и надеждна платформа, която им позволява да решават програмни задачи, да получават незабавна и обективна автоматична обратна връзка, и по този начин да подобряват непрекъснато своите решения и умения.

- **Администратори:** Тази група е отговорна за цялостната поддръжка на системата, ефективното управление на потребителските акаунти и прецизното конфигуриране на системните параметри и настройки.

Разбирането на тези разнообразни потребности и очаквания е ключово за ефективното проектиране на системата. Този анализ служи като основа за дефинирането на функционалните и нефункционалните изисквания, които ще гарантират, че пилотната система ще отговори на нуждите на своите бъдещи потребители

2.1.3 Функционални изисквания

При проектирането на пилотната система за автоматизирано оценяване бяха дефинирани следните ключови функционални изисквания, които отразяват типичния работен процес в Online Judge Systems (OJS) и специфичните нужди на университетската среда [4]:

Подавания (submissions): Системата трябва да поддържа два основни вида подавания на решения: "CUSTOM" и "ACTUAL".

При подаване от тип "CUSTOM" потребителят има възможност да предостави свои собствени тестови данни, включващи примерни вход и очакван изход.

При подаване от тип "ACTUAL" се използват предварително дефинирани за задачата данни, които не са видими за потребителя. За този тип подавания, данните за оценяване обикновено включват:

- Еталонни/Референтни решения (R): Програми, които служат като коректен ориентир.
- Тестови входни данни (I), очаквани изходни данни (O), генератори на данни (G) и валидатори (V): Набори от данни и инструменти за автоматично тестване.
- Набор от тестови случаи (ncases): Със съответните двойки вход/изход.

Всеки проблем (задача) разполага с показатели за оценка, включително флагове, времеви ограничения и ограничения на паметта, които се използват за оценка на производителността на потребителското решение. След подаване, решението се изпраща към оценителния модул заедно с необходимата идентификационна информация [4].

Оценяване на submission: Има четири различни начина за оценяване на коректността на дадена програма в системата за онлайн съдии (OJS) въз основа на флага, свързан с подадената информация:

- DIFF: OJS сравняват стандартния изход на програмата на потребителя с изхода, подготвен от съдията. Коректността се определя от текстовите разлики между двата изхода.
- ERROR: В този случай и стандартният изход на програмата на потребителя, и изходът на съдията се състоят от реални числа. OJS оценяват коректността, като допуска определено ниво на грешка, определено от стойността „eps“, зададена в задачата.
- СПЕЦИАЛНО: OJS използват специална програма за проверка („checker“), специфична за задачата. Проверяващият сравнява стандартния изход на програмата на потребителя с изхода на съдията, като използва предварително определени входове и изходи, за да вземе решение за коректност.
- РЕАКТИВНА: OJS използват реактивна програма, която взаимодейства с програмата на потребителя. Тази реактивна програма обработва конкретно данните на проблема и оценява правилността въз основа на това взаимодействие [4].

Връщане на резултат от submission: OJS връща резултата от submission-а на програма, като използва набор от резултати от оценяването, представени с VR (evaluation value - стойност на оценяване) от някои от следните възможности:

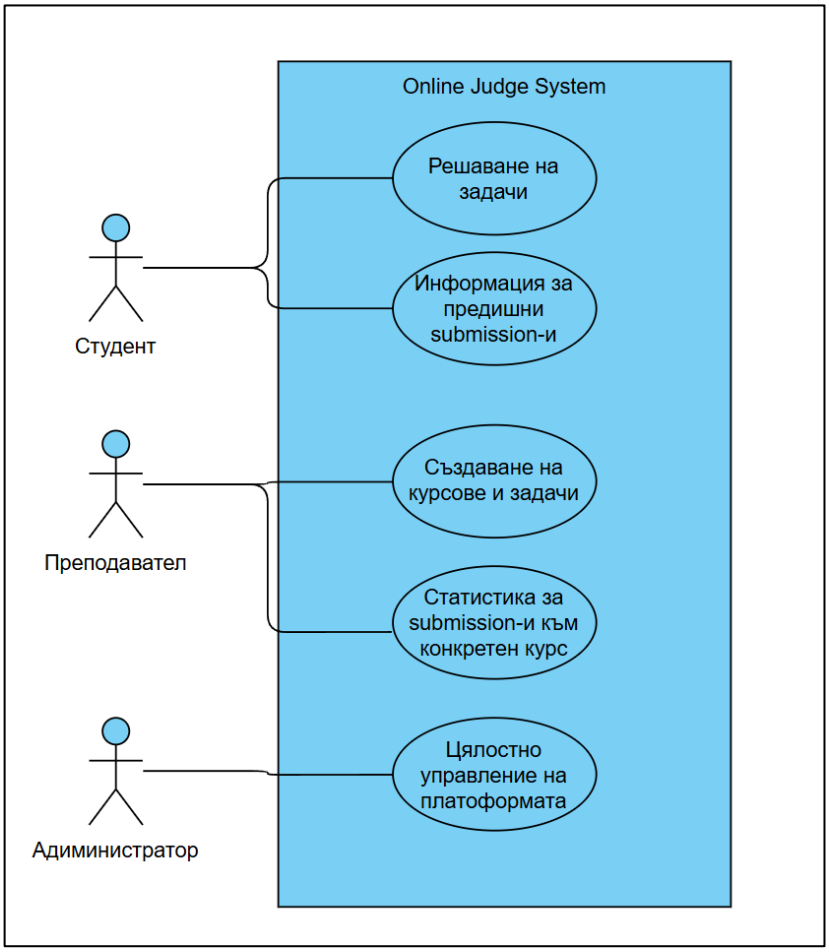
- AC (Accepted - Приет): Програмата дава правилния резултат без проблеми.
- WA (Wrong Answer - Грешен отговор): Програмата произвежда неправилен резултат (само за подаване на флаг DIFF, когато изходът не съответства на очаквания резултат).
- CE (Compile Error - Грешка при компилиране): Програмата не успява да се компилира правилно на посочения език. Изпраща се съобщение, обясняващо грешката.
- RE (Runtime Error - грешка по време на изпълнение): Програмата се сблъсква с грешка по време на изпълнение, например препълване на стека, достъп до масив извън обхвата, деление на нула или други неочаквани завършвания. Изпраща се съобщение, описващо грешката.
- TLE (Time Limit Exceeded - Превишаване на времевия лимит): Програмата превишава времевия лимит за изпълнение.
- MLE (Memory Limit Exceeded - Превишен лимит на паметта): Програмата превишава лимита на паметта по време на изпълнение.

За TLE и MLE OJS оценяват производителността на програмата въз основа на времето на процесора и използването на паметта за всеки тестови случай. Ако някое от тях надвишава границите, определени за проблема, резултатът се счита за незадоволителен. Ако флагът е DIFF, OJS връща WA, когато стандартният изход (STDOUT) не съответства на очаквания изход за входния тестови случай [4].

Създаване на задачи: Преподавателят въвежда цялата информация свързана с конкретен проблем и информация свързана с конкретен проблем – като определени входове/изходи, име на проблем, описание, шаблон за тялото на решение както и шаблон за главния клас, който ще бъде конструиран заедно с подаденото решение за да има завършена програма, която да може да се компилира и изпълни.

Панел за преподаватели: Преподавателите имат достъп до специален панел, който показва статистика за конкретната задача (напр. последните 10 submission-и, и техния резултат).

На Фиг. 2.1 е изобразена Use Case диаграма на системата, описвайки главните функционални изисквания.



Фигура 2.1 Use Case диаграма на пилотната система

Представената диаграма визуално обобщава основните функционални изисквания, като ясно разграничава взаимодействията между различните потребителски роли и системата. Тази диаграма служи като визуална основа за

детайлизирането на системните изисквания и за последващото проектиране на потребителския интерфейс и архитектурата.

След като дефинирахме основните функционалности, е необходимо да бъдат разгледани и нефункционалните изисквания, които са от съществено значение за качеството и ефективността на пилотната система.

2.1.4 Нефункционални изисквания

Нефункционалните изисквания се отнасят до качествени характеристики и ограничения при проектирането на софтуер, които не са пряко свързани с конкретни функционалности, но са критични за цялостната производителност, надеждност и използваемост на системата. Те обхващат широк спектър от аспекти, като тук ще бъдат обсъдени ключови нефункционални изисквания, ориентирани към Online Judge Systems (OJS).

Незабавност/бързодействие (Immediacy): Тъй като от OJS винаги се изисква да генерира обратна връзка в реално време, е от съществено значение да се минимизира времето за реакция между подаването на решение и получаването на съответната оценка [4]. Това изискване е един от основните показатели, които влияят върху качеството и удобството за потребителя (user-friendliness). Важно е обаче да се отбележи, че всеки отделен процес и общите комуникационни разходи могат да повлияят на закъснението на други заявки, особено когато ресурсите са споделени в облачна среда.

Сигурност (Security): Предвид факта, че OJS изпълнява произволен код, предоставен като решение, сигурността представлява един от основните проблеми. Изпратеният код не следва да компрометира основните операции на системата, като например неоторизиран достъп до вътрешни файлове или процеси. Най-важното е, че една сигурна OJS трябва ефективно да предотвратява влиянието на злонамерени кодове върху външни системи [4]. Следва да се отчете, че високите нива на сигурност често оказват влияние върху производителността, като защитената архитектура и допълнителният контрол може да са за сметка на незабавността.

Често срещаните решения за повишаване на сигурността в OJS включват създаването на „sandbox“ (изолиран контейнер) за изпълнение на изпратения код, както и ограничаване на ресурсите, чрез задаване на лимити за времето за изпълнение и използваната памет. Това предотвратява злоупотреби, като безкрайни цикли или операции, изискващи значителни ресурси. Традиционно подобни системи използват виртуални машини за изграждане на „sandbox“, но по-съвременен и ефективен подход е използването на Docker контейнери, които са значително по-леки и бързи в сравнение с виртуалните машини.

Последователност (Consistency): Системата трябва да осигурява, че оценяването на даден код за решение на определена задача е легитимно и възпроизводимо по отношение на производителността, независимо от времето или други подавания. Това означава, че сравнението между различни решения (дори от един и същ потребител) трябва да бъде последователно [4]. По този начин, системата може да бъде разгърната в централизиран специализиран сървърен клъстер (или контейнерна среда), където средата се споделя от потребителите при едни и същи изчислителни ресурси.

Устойчивост (Robustness): Системата трябва да бъде винаги на разположение и оперативно способна, дори при отсъствие на администраторски надзор. OJS трябва да е способна да се справя ефективно с кодове за решения, които консумират прекомерно много процесорни цикли, памет и ресурси на стека, без да прекъсва работа. Дори при възникване на грешки или аномалии, системата трябва да продължи да реагира и да предоставя услугата за оценка [4].

Коректност (Correctness): От решаващо значение е системата да осигурява обективни и справедливи оценки [4]. Тя не трябва да оценява грешно решение като правилно, нито да оценява правилно решение като грешно.

Скалируемост (Scalability): OJS трябва да бъде проектирана така, че да може ефективно да се скалира (увеличава или намалява капацитета си) в зависимост от броя на потребителите и динамичното натоварване от подадени заявки [4].

Наличност (Availability): От OJS се изисква да работи непрекъснато и да бъде готова за подаване на решения във всеки един момент. Наличността обикновено се изразява като процент от времето, през което системата е оперативно способна за определен период [4]. С други думи, OJS трябва да се стреми да сведе до минимум времето на престой (downtime), доколкото е възможно.

След дефинирането на функционалните и нефункционалните изисквания, следващата ключова стъпка в проектирането на пилотната система е структурирането на базата от данни, което ще осигури ефективното съхранение и управление на цялата необходима информация.

2.1.5 База от данни

За изграждането на пилотната система ще бъде използвана релационна база данни PostgreSQL. Този избор е обусловен от факта, че PostgreSQL е система с отворен код, известна със своята изключителна стабилност, висока производителност и възможност за поддръжка на сложни типове данни.

В базата данни ще се съхранява цялата необходима информация за функционирането на системата, включително:

- **Потребителски данни:** Информация за регистрираните студенти, преподаватели и администратори.
- **Информация за задачите:** Подробности за всяка задача, включително заглавие, описание, условия, както и специфични шаблони (напр. шаблон за тялото на решението или за главния клас).
- **Тестови случаи:** Предварително дефинирани входни данни и очаквани изходни резултати, обвързани с всяка задача.
- **Подадени потребителски решения** (Submissions): Изходен код, подаден от потребителите, заедно с информация за използвания език за програмиране.

- **Резултати от оценяването:** Подробности за всеки submission, включително статус (AC, WA, TLE, RE и т.н.), време за изпълнение, използване на памет и други показатели.

Тази добре структурирана база от данни ще осигури надеждна основа за всички операции на пилотната система, позволявайки ефективно управление на информацията за потребители, задачи, тестове и резултати от оценяването.

2.1.6 Интерфейс и потребителско изживяване

Потребителският интерфейс (UI) е ключов елемент при първоначалното взаимодействие на потребителя със софтуерно приложение, формиращ първото впечатление и улесняващ достъпа до функционалностите. Един добре проектиран интерфейс улеснява навигацията и води до по-добро потребителско изживяване (UX), което от своя страна е решаващ фактор за ангажираността и удовлетвореността на потребителите. Прилагането на културно съобразен дизайн също е съществено, тъй като съобразяването с културните предпочитания на потребителските групи води до значително повишена удовлетвореност.

UX/UI дизайнът играе особено важна роля при изграждането на уеб и мобилни платформи. Например, предпочитанието на потребителите към „hamburger менюто“ при мобилни устройства е показателно за нуждата от адаптивен и интуитивен интерфейс. Значението на добрия UX се потвърждава и от статистически данни – около 68% от потребителите напускат уебсайт заради лошо UX, докато 62% биха се върнали при добро потребителско изживяване. Това ясно подчертава пряката връзка между дизайна на интерфейса и успеха на дигиталните продукти.

В контекста на онлайн обучението, UX/UI дизайнът също има съществено значение. Системи за управление на обучението (LMS), като Moodle, служат като основна платформа за достъп до учебни материали, комуникация и сътрудничество между студенти. Изграждането на LMS платформи с фокус върху потребителското изживяване показва значително по-добро възприемане от страна на студентите в

сравнение с традиционни решения, които често се описват като неинтуитивни и затрудняващи учебния процес.

Допълнителна възможност за подобряване на потребителското изживяване в обучителен контекст е използването на геймификация – внедряване на игрови елементи и интерактивни интерфейси. Изследвания показват, че студенти предпочитат формати като "drag and drop" при решаване на тестове, тъй като са по-удобни и минимизират възможността за грешки. Това подчертава значението на интерфейса като първа точка на контакт и двигател за ангажираност.

Въпреки тези наблюдения, UX дизайнът често остава на заден план при разработката на образователни платформи, като основният фокус обикновено е върху съдържанието, а не върху удовлетвореността на студентите. Това може да доведе до объркване и ниска мотивация за участие в учебния процес. В същото време, качественото потребителско изживяване може да има съществено влияние върху възприеманото учене и удовлетвореността на студентите [8].

За изграждането на потребителския интерфейс на системата ще бъде използвана компонентна библиотека, която позволява създаването на съвременна, интуитивна и визуално последователна среда, съобразена с принципите на доброто потребителско изживяване. Интерфейсът е проектиран така, че да се адаптира спрямо различните потребителски роли и да осигурява лесен и бърз достъп до съответните функционалности, с което се гарантира ефективна и удобна работа със системата.

2.1.7 Ограничения

При проектирането и разработката на пилотната система за автоматизирана проверка и оценяване на програмен код се налагат редица ограничения, свързани както с технически, така и с организационни и времеви фактори. Тези ограничения определят рамките, в които системата може да бъде реализирана и да функционира ефективно.

Технически ограничения:

- Ограничение в броя на поддържаните програмни езици: Поради използването на външна система за изпълнение на кода – Judge0, която към момента поддържа 39 програмни езика, в рамките на настоящата дипломна работа ще бъдат интегрирани само част от тях. В първоначалната версия, изградена като MVP (Minimum Viable Product), ще бъдат поддържани само избрани езици, за да се гарантира стабилност и да се запази фокусът върху най-важните функционалности.
- Ограничение в сложността на задачите и методите за оценяване: Пилотната система ще се фокусира основно върху DIFF модела за оценяване на подадените решения (submissions). Това означава, че основният метод за проверка ще бъде сравнение на генерирания от потребителския код изход с предварително дефинирания очакван изход за всяка задача. Останалите, по-сложни методи за оценяване (като ERROR, СПЕЦИАЛНО, РЕАКТИВНА), дефинирани в 2.1.3, няма да бъдат включени в първоначалния обхват.

Организационни и времеви ограничения:

- Времева рамка за изпълнение на проекта: Целият проект, включително разработката на системата и изготвянето на необходимата документация, трябва да бъде завършен в рамките на една академична година. Това налага строго ограничение върху броя и сложността на функционалностите, които могат да бъдат внедрени.
- Ограничение на човешките ресурси: Съществува и организационно ограничение, свързано с факта, че разработващият екип се състои само от един човек, който поема цялостната отговорност за разработката и управлението на жизнения цикъл на софтуерния

проект (SDLC). Това изисква ефективно приоритизиране и управление на времето.

Идентифицираните ограничения са ключови за дефинирането на **реалистичен и постижим обхват** на пилотната система, като същевременно гарантират фокус върху най-важните функционалности. Взимайки предвид тези рамки, следващият раздел ще се фокусира върху проектирането на архитектурата на системата, която ще осигури ефективното изпълнение на дефинираните изисквания.

2.2 Архитектура на системата

Архитектурата на системата е съществена част от нейното планиране и реализация, тъй като определя начина, по който отделните компоненти взаимодействат помежду си и с потребителите. За пилотната система е избран модерен и утвърден подход за изграждане на уеб базирани приложения, който осигурява добра разделеност на отговорностите, гъвкавост и мащабируемост. *Архитектурният модел е изграден върху принципите на REST***.**

2.2.1 REST

REST (REpresentational State Transfer) е архитектурен стил, предназначен да осигури стандартизирана комуникация между компютърни системи в интернет. Системите, съвместими с REST, често наричани RESTful системи, се характеризират със своята безсъстоятелност (statelessness) и разделянето на отговорностите между клиента и сървъра [9].

Ключов фактор за ефективността на една REST услуга е спазването на добри практики при проектирането на самото API (Application Programming Interface).

Такива практики включват:

- Логично и последователно именуване на ресурсите: Използване на съществителни имена (напр. /users) вместо действия (напр. /getUsers).

- Прилагане на йерархична структура: За описване на зависими или вложени ресурси.
- Съответствие на HTTP методите с CRUD операциите: Използване на GET за четене, POST за създаване, PUT/PATCH за редактиране и DELETE за изтриване на ресурси.

Друг съществен принцип е несъстоятелността (statelessness), при която всяка клиентска заявка трябва да съдържа цялата необходима информация за нейното изпълнение, без сървърът да съхранява информация за предишни заявки. Това опростява логиката на сървъра и значително улеснява скалирането на системата [10], [11].

При по-големи и развиващи се проекти е важно да се прилага версия на API-то, за да се избегнат конфликти при промени във функционалността. Това може да се реализира чрез обозначения в URL адреса (напр. /api/v1) или чрез специални HTTP заглавия.

В допълнение към структурата и логиката на API-то, от съществено значение е и сигурността. Добра практика е използването на HTTPS за криптиране на данни и прилагане на механизми като OAuth за удостоверяване и контрол на достъпа. И накрая, за да се гарантира добро представяне при работа с големи обеми данни, често се използват кеширане и пагинация. Те позволяват по-бърз достъп до често заявявани ресурси и оптимизират зареждането на дълги списъци или резултати от търсене.

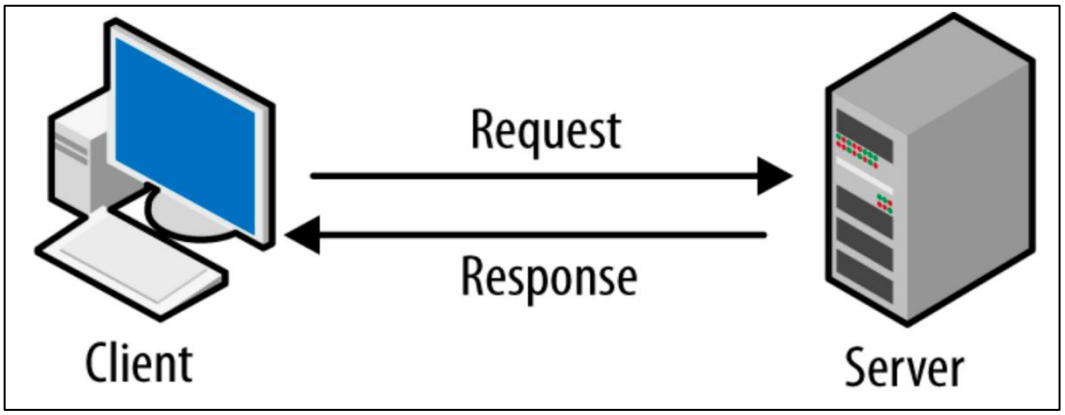
Всички тези добри практики, когато се прилагат последователно, допринасят за създаването на стабилни, сигурни и мащабируеми REST услуги, които са лесни за използване и поддръжка както от страна на разработчиците, така и от страна на потребителите [11].

Основни принципи и ограничения на REST:

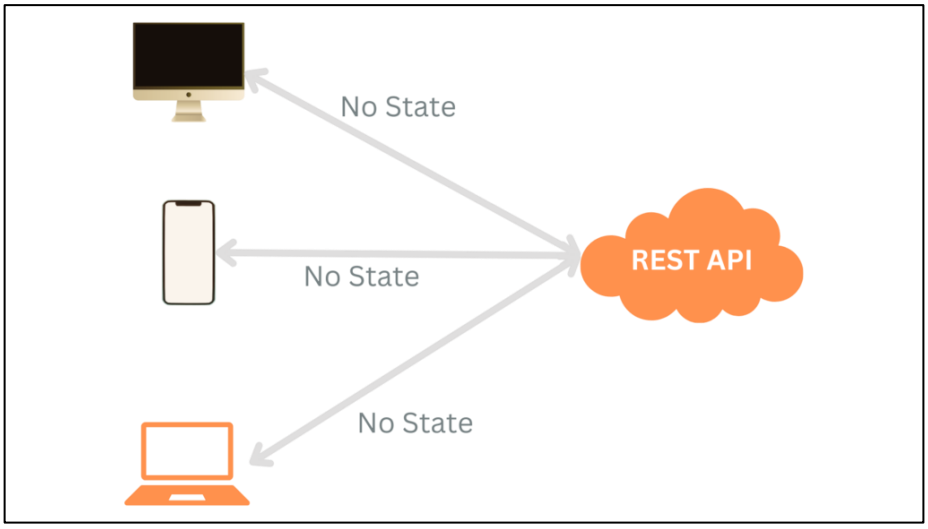
Клиент-Сървър модел: Този архитектурен принцип на REST е базиран на разделянето на отговорностите (separation of concerns). Клиентът инициира

заявки към сървъра, който ги приема, обработва и връща подходящ отговор. Чрез опростяване на функциите на сървъра се постига по-добра мащабируемост. Основното предимство на този модел е, че той подобрява преносимостта, развитието и скалируемостта на приложението [12] [13].

Слаба свързаност (Loose Coupling) и **несъстоятелност** (Statelessness): Ключова характеристика на REST е слабата свързаност. Тя намалява сложността и елиминира нежелани зависимости в рамките на системата. За постигането на този принцип са предложени няколко правила, включително факта, че REST е несъстоятелен (stateless) и се състои от капсулирани, прозрачни слоеве [12], [14].



Фигура 2.2 Модел клиент-сървър [13]



Фигура 2.3 Примерна диаграма на loose coupled и stateless при REST [14]

Поддръжка на кеширане (Cacheable): Механизмът за кеширане е фундаментален принцип в REST, който се добавя към клиент-сървър комуникацията, за да подобри мрежовата ефективност. Принципът на кеширане гласи, че данните в отговор на заявка трябва да бъдат ясно обозначени като кешируеми или некешируеми. Ако отговорът е кешируем, кешът на клиента или междинната система получава правото да използва повторно тези данни за по-късни, съответстващи заявки, което значително увеличава мащабируемостта, ефективността и производителността [12].

2.2.2 Компоненти на системата

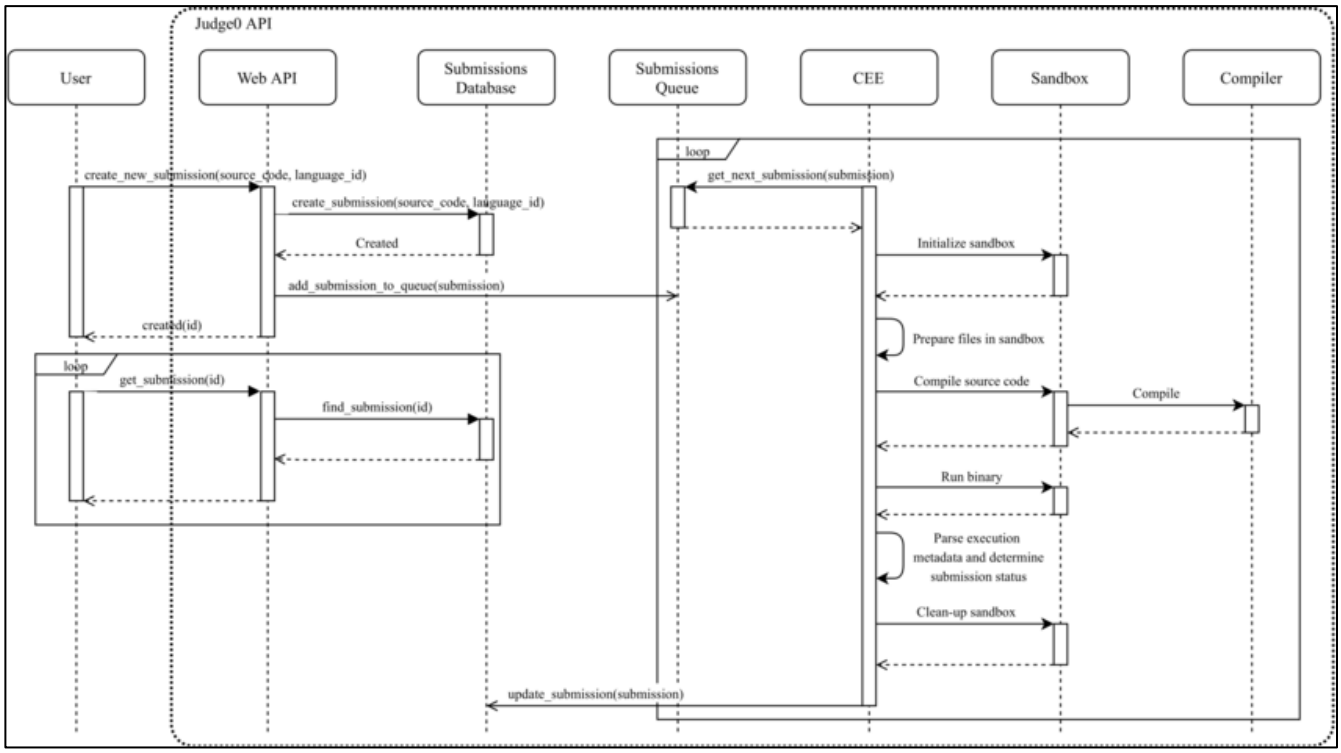
Сървърна част (Back-end): Реализирана е с помощта на C# и .NET платформата. Тя отговаря за бизнес логиката на приложението, обработката на заявки от клиента и връзката с базата данни. Основен компонент е уеб RESTful API, чрез което се осъществява комуникацията между клиентската част и сървъра.

API-то предоставя нужните функционалности за един MVP, като създаване на управление на програмни задачи, комуникацията с външния judge сървис, към който се изпращат подадените решения, съхраняване и извличане на резултати както и управление но потребителски роли и достъп.

За улесняване на разработката, тестването и бъдещата поддръжка, API-то е детайлно документирано и достъпно чрез Swagger. Чрез него могат да се преглеждат всички ендпойнти (endpoints), статус кодовете им, приетите параметри и върнатите отговори, което значително подпомага валидирането и отстраняването на грешки в хода на разработката.

Модул за оценка (Judge0): В архитектурата на системата модулът, който изпълнява подадения от потребителите код, е Judge0 — онлайн система с отворен код за автоматизирано изпълнение и оценка на програмен код, известна със своята надеждност и мащабируемост [15]. В конкретната реализация Judge0 се използва като външен сървис под формата на web API. Нашият back-end комуникира с него,

като подава студентския код за изпълнение, проверява статуса на вече изпратени submission-и и извлича резултати.



Фигура 2.4 Sequence диаграма на работа на Judge0 [15]

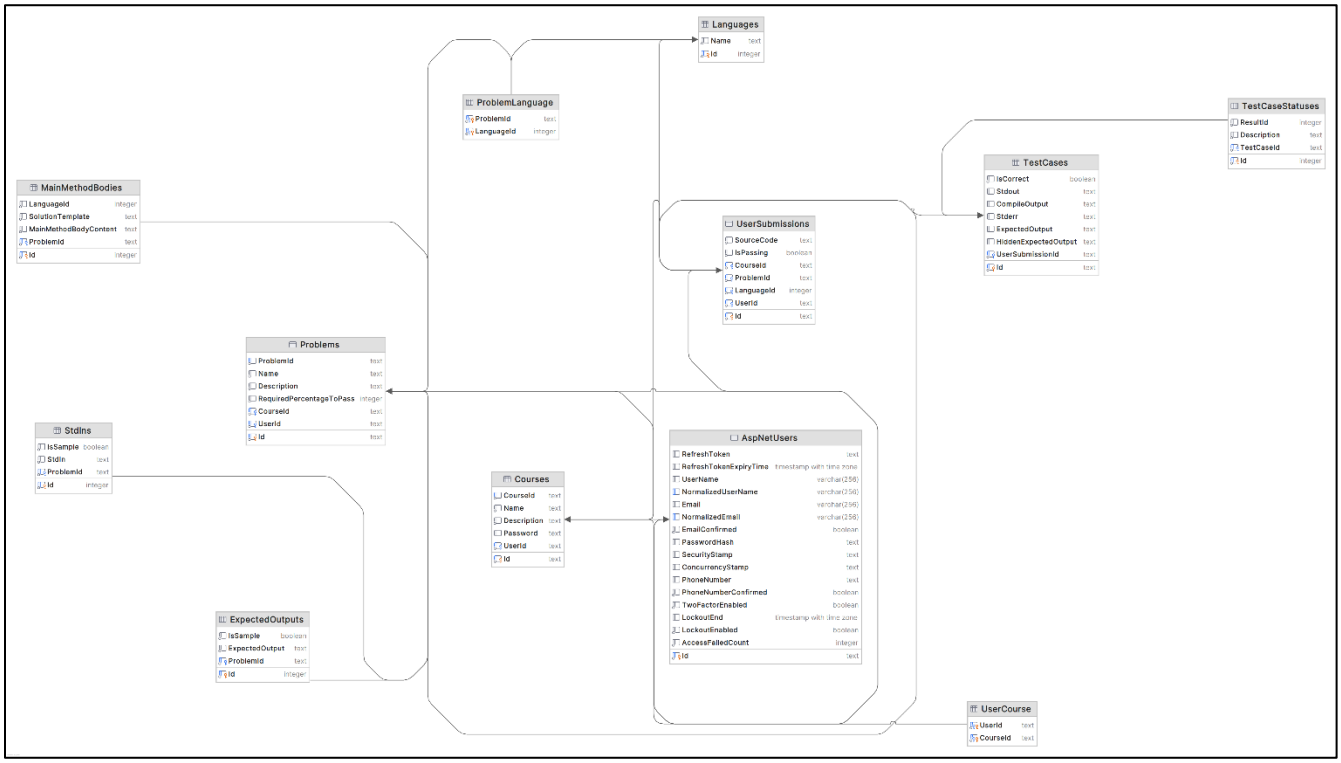
Клиентска част (Front-end): Потребителският интерфейс е разработен с библиотеката React. Той осигурява интуитивна, динамична и удобна за използване среда както за студенти, така и за преподаватели. Интерфейсът следва да се адаптира и да визуализира подходящи менюта спрямо ролята на вписания потребител. Освен това трябва да включва някои основни елементи като страница за създаване и редактиране на задачи, страница в която се визуализира редактор на код (code editor), в който студентите ще записват и изпращат своите решения, панел на страницата с редактора на код, в който да виждат резултатите от последните си submission-и както и панел за администраторите, в който да могат да управляват системата.

База данни: Както вече беше споменато, за нашата база данни ще използваме PostgreSQL.

Базата данни е интегрирана с бекенда чрез ORM (Object-Relational Mapping), което значително улеснява достъпа до данни и тяхното управление в системата.

Важно е да се отбележи, че при изграждането на базата е използван подходът „Code-First“, при който структурата на таблиците се създава въз основа на дефинираните entity класове.

Освен това използваме и вградената система ASP.NET Identity, чрез която се осигурява регистрация, автентикация и ролеви достъп. Нейните таблици не са изобразени в ER-диаграмата, тъй като не са модифицирани спрямо стандартната реализация.



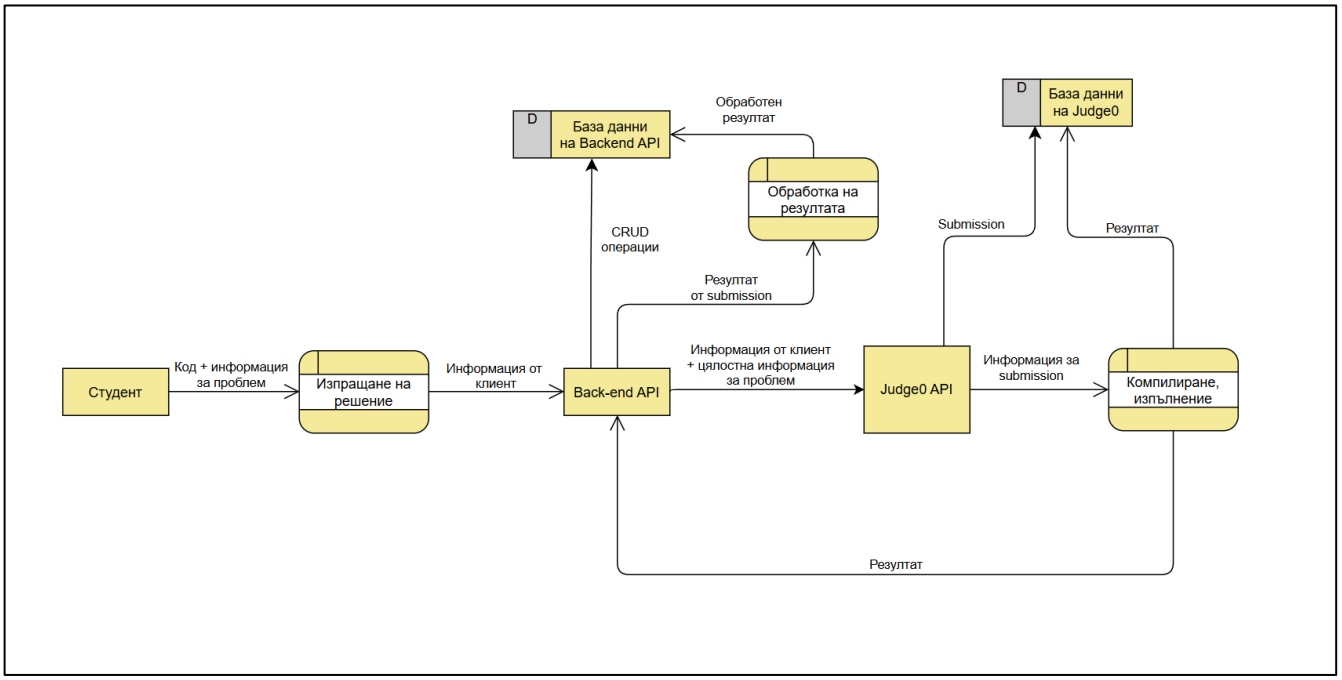
Фигура 2.5 Схема на базата от данни на разработваната система

2.2.3 Поток на данните

Потокът на данните в системата следва ясна и логична последователност, която обхваща взаимодействието между потребителя, клиентската част, сървърна част и Judge0 модула за оценка.

Процесът може да бъде описан в следните стъпки:

1. Потребителско взаимодействие:
    a. Първоначално студентът извършва вход в своя акаунт.
    b. След успешна автентикация, той навигира до избран от него курс и избира конкретна задача за решаване.
    c. Чрез потребителския интерфейс (изграден с React) студентът въвежда и подава своето решение, като посочва изходния код, избрания език за програмиране и идентификатора (ID) на задачата.
2. Обработка на заявка от бекенд:
    a. Тези данни се изпращат във формат JSON чрез HTTP заявка към бекенд сървиса.
    b. Бекендът извършва валидация и предварителна обработка на получените данни.
3. Взаимодействие с Judge0:
    a. След успешна валидация, бекендът подава заявка към Judge0 модула за оценка.
    b. Тъй като Judge0 е конфигуриран да работи асинхронно, при подаване на заявката веднага се връща submission ID.
    c. Използвайки този ID, нашата система периодично проверява (polling) статуса на изпълнение на подаденото решение.



Фигура 2.6 Data flow диаграма на системата

4. Съхранение и визуализация на резултати:
    a. След като резултатите от оценяването са готови и получени от Judge0, те първоначално се записват в неговата собствена база данни.
    b. Впоследствие тези резултати се обработват и съхраняват и в нашата основна база данни.
    c. Накрая, отговор с крайния резултат от оценяването (напр. дали решението е успешно и покрива изискуемия праг за преминаване) се изпраща обратно към клиента (фронтенда) за визуализация пред студента.

Тази детайлна последователност на действията осигурява ефективно и надеждно обработване на потребителските решения, както и своевременно предоставяне на обратна връзка. Цялостният поток демонстрира ясната комуникация между всички компоненти на системата, гарантирайки нейната функционалност и стабилност.

ИЗВОДИ ПО ВТОРА ГЛАВА

В резултат на анализа и проектирането, извършени във втора глава, бяха дефинирани основните изисквания и архитектурни насоки за изграждането на пилотна система за автоматизирано оценяване на програмен код. Чрез ясно формулирани функционални и нефункционални изисквания, определяне на заинтересованите страни и уточняване на обхвата, беше постигната пълна спецификация на очакваната функционалност. Базата от данни бе структурирана съобразно логическите зависимости между потребители, задачи и изпълнения, което гарантира надеждно съхранение и обработка на резултати.

Представената архитектура, базирана на REST принципи, демонстрира добра модулност и разширяемост, като очертава взаимодействието между отделните компоненти — уеб интерфейс, сървърен слой и външен Judge модул. Дефинираният поток на комуникация между модулите потвърждава, че проектираната система ще отговаря на изискванията за сигурност, ефективност и удобство за крайния потребител. Така поставените теоретични основи служат като стабилна база за практическата реализация, която ще бъде разгледана в следващата глава.

ТРЕТА ГЛАВА. РАЗРАБОТВАНЕ И ОЦЕНКА НА ПИЛОТНАТА СИСТЕМА

За реализирането на функционална, надеждна и мащабируема система за автоматизирана проверка и оценка на програмен код беше избран модерен стек от технологии и утвърдени софтуерни практики. В тази глава се представят основните технически решения, използвани при разработването на системата – както по отношение на back-end и front-end имплементацията, така и по отношение на базата данни, интеграцията с външни услуги и процесите по разработка и контрол на версиите.

Проектът е достъпен в GitHub: https://github.com/slavi22/Uni-Judge

Избраните технологии са съобразени със съвременните изисквания за сигурност, производителност и поддръжка, като приоритет е даден на ясната архитектура, повторната използваемост на компонентите и възможността за бъдещо разширяване и интеграция с други системи.

3.1 Технологично осигуряване

При разработката на проекта е използвана система за контрол на версиите Git, а за отдалечено хранилище – GitHub, където може да се проследят всички етапи на развитие на системата. Работният процес е организиран чрез използването на отделен клон (branch) dev, в който се извършва активната разработка, преди промените да бъдат обединени (merge) с тестовия клон (test branch) за допълнително тестване и подготовка за продукция, а впоследствие – с главния клон (main branch).

За реализиране на различните компоненти на системата са използвани изцяло инструменти на JetBrains, които значително допринасят за продуктивността на разработката. Това включва Rider – интегрирана среда за разработка на C# и .NET приложения, WebStorm – специализирано IDE за уеб разработка, използвано тук за създаване на React интерфейса, както и DataGrip – мощен инструмент за работа с бази данни, използван както самостоятелно, така и като плъгин в Rider.

3.1.1 Сървърна част (Back-end)

В процеса на разработка на сървърната част (back-end) на системата са използвани C# и .NET (.NET 9), като е изградена многослойна (layered) архитектура, базирана на принципите на разделяне на отговорностите (separation of concerns). Използвани са Dependency Injection и Repository pattern като утвърдени шаблони за проектиране (design patterns) при уеб приложенията, за осигуряване на лесна поддръжка, тестируемост и разширяемост на системата. Комуникацията между слоевете е ясно структурирана, а бизнес логиката е отделена от достъпа до данни.

C#: междуплатформен език с общо предназначение, който прави разработчиците продуктивни, докато пишат високопроизводителен код. С милиони разработчици, C# е най-популярният .NET език. C# има широка поддръжка в екосистемата и всички .NET работни натоварвания. Базиран на обектно-ориентирани принципи, той включва много функции от други парадигми, не на последно място функционално програмиране [16].

ASP.NET Core: междуплатформена, високопроизводителна рамка (framework) за изграждане на съвременни уеб приложения. Този framework с отворен код позволява на разработчиците да създават уеб приложения, услуги и API-та, които могат да работят на Windows, macOS и Linux. Тя е създадена за разработване на мащабни приложения и може да се справи с всякакъв размер натоварвания, което я прави надежден избор за приложения на корпоративно ниво [17].

Entity Framework Core и ORM (Object-Relational Mapper) е препоръчителният Object-Relational Mapper (ORM) за .NET, известен преди като .NET Core. Като наследник на EF6, EF Core е изцяло преработен и е направен с отворен код в GitHub. Обектно-релационното моделиране (ORM) представлява подход за работа с бази данни чрез използване на обектно-ориентиран програмен език, без необходимост от писане на суров SQL код. Вместо директни заявки, разработчиците боравят с обекти и класове, които отразяват структурата на базата

данни. Това улеснява процеса на разработка, намалява риска от грешки и подобрява поддръжката на кода в дългосрочен план. ORM решенията често предлагат допълнителни функционалности като кеширане, отложено зареждане (lazy loading) и управление на връзките с базата данни (connection pooling), което допринася за по-добра производителност на приложението. Те осигуряват и абстракция между логиката на приложението и структурата на базата данни, което прави системата по-гъвкава при промени в схемата или при миграция към друга база данни в бъдеще.

EF Core Migrations е функционалност на Entity Framework Core, която позволява на разработчиците да развиват схемата на базата данни постепенно и контролирано, спрямо развитието на приложението. С помощта на миграциите могат лесно да се прилагат или връщат промени по структурата на базата, без да се губят съществуващи данни. EF Core автоматично генерира и изпълнява необходимите SQL скриптове за актуализиране на схемата, което елиминира нуждата от писане на ръчни заявки и намалява риска от грешки или загуба на информация [18].

JWT (JSON Web Token): За сигурността на системата е реализирана JWT-базирана аутентикация, като е въведен Role-Based Access Control (RBAC), позволяващ дефиниране на различни права и нива на достъп за потребителите (например студент, преподавател и администратор). JSON Web Token (JWT) е отворен стандарт (RFC 7519), който определя компактен и самостоятелен начин за сигурно предаване на информация между страните като JSON обект. Тази информация може да бъде проверена и надеждна, защото е цифрово подписана. JWT могат да бъдат подписани с помощта на секретен код (с алгоритъма HMAC) или двойка публичен/частен ключ, използвайки RSA или ECDSA [19]. Автентикационната схема е базирана изцяло на JWT и включва генериране както на access token, така и на refresh token. Access token-ът се използва за удостоверяване при всяка клиентска заявка, докато refresh token-ът позволява подновяване на

сесията без повторно влизане. За повишаване на сигурността refresh token-ите се съхраняват в базата данни, свързани с конкретен потребител.

Системата използва ASP.NET Core Identity - библиотека, която автоматично създава необходимите таблици в базата данни и предоставя готови механизми за управление на потребители, роли, пароли и други аспекти на сигурността.

Google Authentication (OAuth 2.0): Освен че системата генерира и предоставя собствено подписани JWT токени за удостоверяване на потребителите, тя поддържа и възможност за вписване чрез Google акаунти. При вписване с Google, потребителят бива удостоверен чрез външния доставчик (Google), след което необходимата информация (напр. имейл и име) се извлича и автоматично се създава запис в нашата база данни, ако такъв не съществува. По този начин можем да асоциираме дейността на потребителя — като например изпратените решения (submission-и) — със съответния акаунт. На тези потребители автоматично се присвоява ролята на студент, както при стандартната регистрация в системата.

OAuth 2.0 е протокол за оторизация, който позволява на потребителите да предоставят ограничен достъп до своите данни на дадено приложение, без да се налага да споделят своите потребителски имена и пароли. Това се постига чрез използването на външна услуга за удостоверяване — например Google, Facebook или GitHub. Когато даден потребител избере да се впише чрез такава услуга (например чрез бутона "Login with Google"), той бива пренасочен към самата платформа (в случая – Google), където се извършва процесът на удостоверяване. Ако потребителят потвърди, че желае да предостави достъп до определена информация (като имейл адрес), Google — в ролята на така наречения resource сървър — издава специален токен, който потвърждава самоличността на потребителя. Обикновено тези токени са във формата на JWT (JSON Web Token), който може да бъде използван от нашето приложение за удостоверяване и управление на достъпа. Този токен съдържа подписана информация, която може да

бъде проверена и валидирана от сървъра на нашето приложение, без необходимост от допълнителна комуникация с Google. Така OAuth 2.0 предоставя сигурен и стандартизиран начин за външно удостоверяване и лесна интеграция с големи платформи, като в същото време запазва сигурността на потребителските данни [20] [21].

Unit testing: За тестването на отделните компоненти в системата е използван xUnit като основен инструмент за писане на unit тестове. За изолиране на зависимости в тестовата среда се използва библиотеката Moq, което позволява лесно „mock“-ване на външни зависимости, като бази данни и външни сървиси. Това е важно, тъй като unit тестовете се фокусират върху логиката на конкретни компоненти, а не върху тяхната интеграция с други системи. Именно поради това, за цялостна проверка на взаимодействието между компонентите са необходими и integration tests. В системата са реализирани над 50 unit теста, обхващащи основно контролери и сървиси, като се цели висока покриваемост на бизнес логиката и устойчивост на приложението.

Обработка на грешки: В презентационния слой (Web API) е конфигуриран глобален механизъм за обработка на изключения (global exception handler), базиран на интерфейса IExceptionHandler.

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json
Content-Language: en

{
 "type": "https://example.com/probs/out-of-credit",
 "title": "You do not have enough credit.",
 "detail": "Your current balance is 30, but that costs 50.",
 "instance": "/account/12345/msgs/abc",
 "balance": 30,
 "accounts": ["/account/12345",
              "/account/67890"]
}
```

Фигура 3.1 Примерен JSON по RFC 9457 спецификацията [22]

При възникване на необработена грешка, вместо приложението да прекъсне изпълнението си, грешката се логва в конзолата на сървъра, а на клиента

се връща стандартизиран JSON отговор във формат Problem Details, съгласно спецификацията RFC 9457 [22].

Контейнеризация с Docker: За улесняване на разработката, тестването и бъдещото разгръщане (deploymnet) на системата, сървърната част е контейнеризирана с помощта на Docker. Той е отворена платформа за разработка, доставка и изпълнение на приложения, която позволява изолиране на приложенията от основната инфраструктура. Това улеснява и ускорява процеса на разработка и внедряване на софтуер. С помощта на Docker, приложенията се пакетират в самостоятелни, изолирани среди, наречени контейнери. Контейнерите съдържат всички необходими зависимости за стартиране на приложението – включително код, библиотеки и конфигурации – което гарантира, че приложението ще работи еднакво независимо от средата, в която се изпълнява. Те са леки, стартират бързо и позволяват едновременно изпълнение на множество приложения на един и същ хост, без конфликти между тях [23].

Проектът включва Dockerfile за бекенда, в който са описани инструкциите за изграждане на образа (docker image), както и docker-compose.yml файл, който дефинира конфигурацията на отделните услуги – включително базата данни и външния Judge0 сървис. По този начин цялата система може да бъде стартирана локално или в облачна среда с една единствена команда. Използването на Docker улеснява бъдещ deployment на приложението, тъй като позволява бързо и надеждно публикуване в различни среди.

OpenAPI и Swagger: По време на разработката е използван OpenAPI спецификацията, като документацията и тестването на API-то се осъществяват чрез Swagger, по-точно чрез Swagger UI [24]. Цялостното API е детайлно документирано и достъпно през браузър, което позволява лесно визуализиране на всички крайни точки, както и тяхното тестване. Това е полезно както за разработчици, така и за крайни потребители, които искат да проверят сървърната логика или да интегрират системата с външни решения.

3.1.2 Клиентска част (Front-end)

Клиентската част на системата е реализирана с помощта на React – една от най-популярните JavaScript библиотеки за изграждане на интерактивни потребителски интерфейси. Целта е да предостави интуитивна, бърза и достъпна среда за работа както на студентите, така и на преподавателите и администраторите.

React е компонентно-базирана JavaScript библиотека, предназначена за изграждане на динамични и интерактивни потребителски интерфейси. Тя значително улеснява създаването на едностранични приложения (Single Page Applications – SPA), като поставя акцент върху висока производителност и лесна поддръжка на кода. React е разработена и се поддържа от Meta (бившият Facebook), и е една от най-широко използваните библиотеки за уеб разработка в съвременния технологичен стек. Последната стабилна версия към момента е React 19. Езикът използва виртуален DOM, което позволява бързо и ефективно обновяване на потребителския интерфейс без нужда от презареждане на страницата. Подходът и е декларативен, което означава, че разработчиците описват как изглежда интерфейсът във всеки един момент, а самата библиотека се грижи за необходимите промени. Освен това, React прилага еднопосочно обвързване на данните (one-way data binding), което осигурява по-добър контрол върху поведението на приложението и улеснява проследяването на потока на данните в него [25].

Typescript е език за програмиране от високо ниво, разработен от Microsoft, който надгражда JavaScript чрез добавяне на статична типизация. Това го прави особено подходящ за големи и мащабируеми проекти. Основното му предимство спрямо чистия JavaScript е възможността за дефиниране на типове данни, което повишава надеждността на кода, улеснява работата в екип и подобрява четимостта и поддръжката на проекта. TypeScript позволява откриването на грешки още по време на разработка, което води до по-стабилен и предвидим код.

Redux Toolkit: Тъй като в React състоянието на компонентите се запазва само докато те са рендерирани, нашето приложение изисква глобален мениджър на

състоянието (global state manager). Това е необходимо, за да се съхранява и споделя информация, която трябва да бъде достъпна между различни части на приложението – например дали потребителят е вписан, избраната тема (тъмна/светла) и други подобни. Ако подобна информация не се съхранява централно, се стига до проблеми – например: потребителят се вписва, но при преминаване към друга страница приложението "забравя", че той е логнат. За да избегнем това, използваме централизирано хранилище за състояние. React предлага вградено решение чрез Context API, но то не е оптимално за по-големи приложения или такива, в които състоянието се променя често. В такива случаи е препоръчително използването на по-ефективно решение, каквото е Redux Toolkit, който използваме в нашия проект.

RTK Query: В предложеното решение често се налага да взимаме данни от сървъра – било то списък със задачи, резултати от решения или информация за текущия потребител. Освен че трябва да ги извличаме, понякога ги променяме, изпращаме тези промени обратно към сървъра и очакваме данните на клиента да се обновят и останат в синхрон. Тези неща не са толкова прости, колкото изглеждат на пръв поглед – трябва да следим кога нещо се зарежда (за да покажем например спинър), да не изпращаме една и съща заявка многократно, да показваме промяната в UI още преди сървърът да е върнал отговор (за да изглежда всичко по-бързо), както и да кешираме резултатите, за да не презареждаме постоянно едни и същи данни. Затова използваме RTK Query – библиотека от екосистемата на Redux Toolkit, която автоматизира всичко това. Тя ни помага да пишем по-малко код и в същото време ни дава всичко необходимо: вградени индикатори за зареждане, кеширане, автоматично обновяване на данните и лесно управление на заявките. Благодарение на RTK Query, комуникацията със сървъра става по-лесна и по-надеждна, което ни позволява да се фокусираме върху логиката на самото приложение [26].

ShadCN е библиотека с отворен код, която предоставя модерен и гъвкав начин за изграждане на потребителски интерфейси в React приложения. Вместо да бъде готов набор от компоненти като други UI библиотеки (например Material UI или Bootstrap), ShadCN предоставя добре организирана система от компоненти, базирани на Tailwind CSS и Radix UI, които могат лесно да бъдат персонализирани и адаптирани към нуждите на конкретния проект [27].

Zod е библиотека за валидация, създадена с акцент върху TypeScript. Тя позволява дефиниране на схеми, чрез които могат да се валидират различни типове данни - от прости стойности (например низове), до комплексни и вложени обекти. За разлика от статичната типизация на TypeScript, която действа само по време на компилация, Zod осигурява валидация по време на изпълнение (runtime). Това означава, че данните, идващи от външни източници - като форми, API заявки или локално съхранение - могат да бъдат проверени в реално време, преди да се използват в приложението [28].

Monaco Editor: Текстовият редактор, използван в приложението за писане и редактиране на решенията на задачите, е Monaco Editor – същият редактор, който стои в основата на Visual Studio Code. Monaco е мощен, уеб-базиран редактор, който предоставя богат набор от функционалности, включително синтактично оцветяване (syntax highlighting), автоматично довършване на код (autocomplete) и др [29].

3.1.3 База данни

В основата на системата стои релационна база данни, реализирана с PostgreSQL, която съхранява и управлява основните структурни единици на приложението – потребители, задачи, информация свързана със задачите (шаблони за решения, финални класове за решения и т.н), изпратени решения и съответните тестови случаи. Базата е моделирана така, че да поддържа ясна и последователна логика на връзките между различните единици (entities) в системата.

Основните таблици включват:

- Users (AspNetUsers) – съдържа информация за потребителите на системата. Използва се ASP.NET Core Identity за управление на автентикацията и авторизацията. Таблиците, свързани с Identity, се генерират автоматично и позволяват поддръжка на роли, refresh токени и друга сигурност-свързана информация.
- Courses – съдържа информация за конкретен курс: име, описание и парола, в случай че достъпът до курса е ограничен. Релацията между курсове и потребители (студенти) е от тип много към много (m:n), реализирана чрез свързваща таблица UserCourses.
- Problems – основна таблица, съдържаща задачите, дефинирани от преподавателите. Всяка задача включва метаданни като име, описание и изисквания за минимален процент за успешно преминаване
- Languages – съдържа списък с поддържаните езици за програмиране. Таблицата се попълва автоматично чрез seed-ване при инициализация на базата. Всеки език може да бъде свързан с множество задачи, което налага m:n релация с таблицата Problems, реализирана чрез ProblemLanguages.
- ExpectedOutputs – съдържа очакваните изходи за конкретни входни данни. Чрез булевата колона IsSample се определя дали даден изход е публичен (видим за потребителя) или е част от скритите тестове.
- MainMethodBodies – съхранява шаблони за решения към задачи. Включва съдържание на основния метод (main class), шаблонен код (SolutionTemplate) и съответния програмен език.
- TestCases – съдържа входно-изходни примери за автоматична проверка на решенията. Всеки тестов случай е свързан с конкретна задача чрез външен ключ (foreign key). Поддържат се както публични, така и скрити тестове.

- TestCaseStatuses – съдържа информация за състоянието на даден тестов случай, включително резултата от Judge0 и описание на статуса (например resultId = 3, description = Accepted).
- UserSubmissions – регистрира всяко решение, подадено от студент. Записва се изходният код, избраният език за изпълнение както и резултатът от проверката.

3.2 Оценка на системата

След разработката на пилотната версия на системата беше извършена цялостна оценка, целяща да провери както техническите характеристики (точност, бързина на проверка и стабилност). Оценката беше разделена на два основни аспекта: провеждане на тестове с реални задачи, както и събиране на количествени и качествени данни за ефективността и.

3.2.1 Тестови сценарии

За да се провери стабилността и функционалността на системата в реални условия, беше проведена серия от тестови сценарии, симулиращи действителна употреба от крайни потребители – студенти и преподаватели. Тези тестове имаха за цел да обхванат основните функционалности на системата и да валидират поведението и в различни ситуации.

Първоначално бяха създадени **тестови акаунти с различни роли** – както чрез стандартна регистрация, така и чрез вписване с Google акаунт. Проведени бяха тестове, при които „студентите“ решаваха два типа задачи. Първият тип включваше изкуствено създадени, опростени задачи, като например: „подай списък и върни първия му елемент“. Целта на този тип тестове беше да се провери коректното функциониране на основната бизнес логика на системата в различни гранични и стандартни случаи.

След валидирането на основните функционалности, бяха използвани и **реални задачи**, взети от популярни платформи за програмиране. Един от примерите, с който беше проведено тестване, е задачата FizzBuzz – добре познат

проблем с ниво на трудност „лесно“, често използван както в обучителни среди, така и в интервюта за оценка на базови програмни умения.

Също така беше проверена системата за **автоматично оценяване** чрез подадени решения с грешки – както логически, така и синтактични – с цел да се проследи дали се връща адекватна обратна връзка към потребителя.

Освен това се тестваха и **административните функционалности** – създаване на нови задачи, добавяне на входно-изходни примери, организиране на курсове и самозаписване на студентите към даден курс. По този начин се симулираше реалната работа на преподавател в рамките на учебен курс.

Резултатите от проведените сценарии показаха стабилна работа на системата, както и правилно прилагане на бизнес логиката при различни ролеви действия и взаимодействия. Сценариите покриха както положителни, така и негативни случаи (напр. изпращане на синтактически неправилен код, опити за достъп до неразрешени ресурси), което помогна да се гарантира по-висока надеждност на системата в бъдеща реална употреба.

3.2.2 Оценка на ефективността

Ефективността на системата беше оценена предимно от техническа гледна точка, като основен фокус беше поставен върху скоростта на обработка на подадените решения и точността на автоматичната проверка.

В проведените тестове беше наблюдавано, че системата обработва подадените решения в рамките на до 5 секунди. Това време включва няколко последователни стъпки: изпращане на заявка от потребителския интерфейс към бекенда, препращане към Judge0, получаване и обработка на резултата, както и съхраняване на информацията в базата данни.

Важно е да се отбележи, че самото компилиране и изпълнение на изпратения код от страна на Judge0 отнема приблизително 0.02 секунди. Причината за общата продължителност от около 5 секунди е, че нашата система извършва допълнителна обработка на резултатите. Бизнес логиката ни не използва директно

вградените механизми на Judge0 за проверка по тестови случаи, а се възползва от него само за компилация и изпълнение на кода. Самата проверка спрямо тестовите случаи се извършва от нашата система, което позволява реализиране на допълнителни функционалности – например визуализиране на дебъг съобщения чрез console.log() или други механизми за потребителска обратна връзка. Това налага допълнителни стъпки в бекенд логиката, които обаче значително повишават гъвкавостта и възможностите на системата спрямо директното използване на Judge0.

По отношение на точността, системата последователно връщаше очакваните резултати за различни типове вход. Беше извършена проверка на механизмите за валидиране на резултати от Judge0, като се проследяваше дали за всеки submission се запазва коректният статус и описание. Тези данни се използват за адекватна обратна връзка към потребителя.

Като цяло, системата демонстрира стабилна работа, бърза обработка на заявките и надеждно поведение в различни тестови ситуации, което я прави подходяща за реална употреба в университетска среда.

3.3 Анализ на резултатите и развитие

В този раздел се разглеждат получените резултати от реализацията и тестването на системата, като се оценява нейната ефективност, стабилност и съответствие с поставените изисквания. Извършва се идентифициране на силните и слабите страни, както и възможностите за бъдещо усъвършенстване и развитие на проекта.

3.3.1 Идентифицирани области за подобрение

След провеждането на тестове и симулации с реални сценарии, може да се заключи, че системата функционира стабилно и изпълнява основните си цели – автоматизирано оценяване на подадени решения, поддръжка на различни роли и сигурна автентикация.

Въпреки това, бяха идентифицирани няколко **потенциални области за подобрение**:

Прозрачност на обратната връзка към потребителя: Системата може да бъде по-прозрачна по отношение на резултатите от оценяването. Например, макар Judge0 да връща допълнителна информация като време за изпълнение, използвана памет и други полезни показатели, в момента тази информация не се визуализира на потребителя. Нещо повече, тя не се използва и при самото оценяване на решенията – практика, която е стандартна в други подобни системи. Така например, ако решението на даден потребител използва твърде много памет, други платформи автоматично го отбелязват като грешно, докато в текущата реализация това не се случва.

Функционалност на административния панел: Втората по-значима слабост е свързана с административния панел. Към момента той е доста базов и служи основно като входна точка към системата, предлагайки минимален набор от възможности. Този аспект не е пропуск, а по-скоро резултат от фокуса върху основната функционалност – изграждането на стабилен механизъм за автоматизирано оценяване. Интерфейсът за администрация не е бил приоритет в настоящия етап на разработка, но в бъдеще определено има сериозен потенциал за разширяване и подобрение с повече възможности за мониторинг, управление на потребители и т.н.

Идентифицирането на тези области за подобрение е ключово за бъдещото усъвършенстване и разширяване на системата. *Тези слабости не подкопават основната функционалност, но представляват ясни насоки за бъдещи итерации, които ще допринесат за по-пълноценно потребителско преживяване и по-висока обща зрялост на продукта.*

3.3.2 Развитие

Системата, в текущия си вид, представлява стабилна основа за бъдещ продукт с реален потенциал за разширение и интеграция в учебната среда. Въпреки че основната функционалност е успешно реализирана, има няколко ключови направления, в които платформата би могла да се развие допълнително:

1. **Разширена поддръжка на програмни езици:** На първо място, поддръжката на повече програмни езици би увеличила гъвкавостта на системата. Макар че Judge0 поддържа над 40 езика за програмиране, в нашето приложение в момента са достъпни само 2 – C# и JavaScript. Това се дължи на факта, че Judge0 се използва основно за компилиране и изпълнение на кода, но останалата логика – включително разпознаване на езика, проверка на тестови случаи (test case-ове) и обработка на резултатите – се реализира изцяло в нашата собствена система. Следователно, за да бъде добавен нов език, е необходимо и нашата система да го поддържа на логическо и техническо ниво.
2. **Интеграция с платформи за управление на обучението (LMS):** Друг възможен път за развитие е интеграцията с LMS платформи, като например Moodle. Като начална стъпка вече е реализирана поддръжка на вписване чрез Google акаунти – метод, който се използва и в системата за онлайн обучение на УниБИТ. Това отваря възможност за по-дълбока интеграция, например чрез изграждане на плъгин за Moodle, който да комуникира с предоставеното от нас REST API и да визуализира данни от нашата система директно в средата за обучение.
3. **Добавяне на модул за състезателно програмиране:** Накрая, би могла да се разгледа възможността за добавяне на модул за състезателно програмиране. Тази функционалност би включвала създаване на състезания с времево ограничение, автоматично оценяване в реално време и класиране на участниците. Подобен модул би разширил обхвата на системата и би добавил

значителна стойност за преподавателите и студентите, които искат да се подготвят за реални събития или да практикуват в по-интензивна среда.

Всички тези възможности за развитие не само биха обогатили функционалността на системата, но и биха увеличили нейната стратегическа стойност като платформа. Инвестирането в тяхното реализиране ще осигури по-широко приложение, по-висока ангажираност на потребителите и дългосрочна устойчивост на проекта в образователната сфера.

ИЗВОДИ ПО ТРЕТА ГЛАВА

В резултат на практическата реализация, описана в трета глава, беше създадена функционираща пилотна система за автоматизирано оценяване на програмен код, отговаряща на предварително дефинираните изисквания. Реализацията обхвана съвременно технологично осигуряване – сървърна част с ASP.NET Core, клиентска част с React и логически свързана база от данни. Използваните технологии демонстрираха добра съвместимост и осигуриха стабилна, разширяема архитектура.

Оценката на системата чрез реални тестови сценарии показа висока степен на ефективност при изпращане, обработка и оценяване на задачи. Анализът на резултатите разкри потенциални области за подобрение, включително разширяване на поддържаните езици и надграждане на административния и потребителския интерфейс за по-добро изживяване. Предложените идеи за развитие очертават ясни възможности за по-широко приложение на системата в академична среда и нейното усъвършенстване в пълноценна платформа за обучение и оценка.

ЗАКЛЮЧЕНИЕ

Настоящият проект постигна основната си цел – изграждане на платформа за автоматизирано оценяване на решения по програмиране, съобразена с нуждите на университетската образователна среда. Успешно бяха реализирани ключови функционалности като регистрация и автентикация на потребители (включително чрез Google акаунти), подаване и проверка на решения чрез външен API (Judge0), поддръжка на роли и курсове, както и основен административен панел.

Системата предоставя стабилна и гъвкава основа за създаване на реална учебна среда, в която преподавателите могат лесно да дефинират задачи, а студентите – да подават решения и получават автоматична обратна връзка. Подобен тип платформи значително облекчават процеса на преподаване и оценяване, особено при голям брой участници, и допринасят за по-обективна и последователна проверка на знанията.

Проектът демонстрира, че автоматизацията в програмирането и проверката на решения е напълно приложима в университетски контекст. Предстоят редица възможности за надграждане – поддръжка на повече програмни езици, по-дълбока интеграция с платформи като Moodle, както и разработване на модули за състезателно програмиране. Всички тези направления могат да разширят обхвата на системата и да я превърнат в още по-полезен инструмент за преподаване и учене.

С оглед на получените резултати и стабилността на реализираната архитектура, може да се заключи, че проектът има потенциал за дългосрочно развитие и реална практическа стойност.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

[1] D. Hou, „A Framework of Online Judge Systems for Assessing Programming Skills,“ Октомври 2019. [Онлайн]. Available: https://www.researchgate.net/publication/351120306_A_Framework_of_Online_Judge_Systems_for_Assessing_Programming_Skills. [Отваряно на 6 Юни 2025].

[2] TianPan.co, „Designing Online Judge or Leetcode,“ TianPan.co, [Онлайн]. Available: https://tianpan.co/notes/243-designing-online-judge-or-leetcode. [Отваряно на 12 Януари 2025].

[3] K. Liu, Y. Han, J. M. Zhang, Z. Chen, F. Sarro, M. Harman, G. Huang и Y. Ma, „Who Judges the Judge: An Empirical Study on Online Judge Tests,“ 13 Юли 2023. [Онлайн]. Available: https://dl.acm.org/doi/10.1145/3597926.3598060. [Отваряно на 6 Юни 2025].

[4] Y. Watanobe, M. M. Rahman, T. Matsumoto, U. K. Rage и P. Ravikumar, „Online Judge System: Requirements, Architecture, and Experiences,“ Юни 2022. [Онлайн]. Available: https://doi.org/10.1142/S0218194022500346. [Отваряно на 6 Юни 2025].

[5] M. Girardin, „What Is LeetCode?,“ Forage, 15 Декември 2023. [Онлайн]. Available: https://www.theforage.com/blog/skills/what-is-leetcode. [Отваряно на 6 Юни 2025].

[6] System Design School, „Design LeetCode,“ System Design School, [Онлайн]. Available: https://systemdesignschool.io/problems/leetcode/solution. [Отваряно на 6 Юни 2025].

[7] R. Forrester, „LeetCode vs HackerRank: Which is Best?,“ Five.co, 3 Август 2023. [Онлайн]. Available: https://five.co/blog/leetcode-vs-hackerrank/. [Отваряно на 6 Юни 2025].

[8] T. K. Miya и I. Govender, „UX/UI design of online learning platforms and their impact on learning: A review,“ 19 Декември 2022. [Онлайн]. Available: https://www.ssbfnet.com/ojs/index.php/ijrbs/article/view/2236/1623. [Отваряно на 19 Юни 2025].

[9] Team, Codeacademy, „What is REST?,“ Codeacademy, [Онлайн]. Available: https://www.codecademy.com/article/what-is-rest. [Отваряно на 24 Май 2025].

[10] A. A. Prayogi, M. Niswar, Indrabayu и M. Rijal, „Design and Implementation of REST API for Academic,“ 2020. [Онлайн]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/875/1/012047/pdf. [Отваряно на 19 Юни 2025].

[11] P. Gowda и A. N. Gowda, „Best Practices in REST API Design for Enhanced Scalability and Security,“ 20 Февруари 2024. [Онлайн]. Available: https://urfjournals.org/open-access/best-practices-in-rest-api-design-for-enhanced-scalability-and-security.pdf. [Отваряно на 19 Юни 2025].

[12] S. Sampangi, „REST AS A WEB ARCHITECTURAL DESIGN 1,“ Септември 2024. [Онлайн]. Available: https://www.researchgate.net/publication/384411040_REST_AS_A_WEB_ARCHITECTURAL_DESIGN_1. [Отваряно на 24 Май 2025].

[13] A. Madooei, „Client-Server Application,“ [Онлайн]. Available: https://madooei.github.io/cs421_sp20_homepage/client-server-app/. [Отваряно на 24 Май 2025].

[14] L. Gupta, „Statelessness in REST API,“ 6 Ноември 2023. [Онлайн]. Available: https://restfulapi.net/statelessness/. [Отваряно на 24 Май 2025].

[15] H. Z. Došilović, „Robust and Scalable Online Code Execution System,“ Септември 2020. [Онлайн]. Available: https://www.researchgate.net/publication/346751837_Robust_and_Scalable_Online_Code_Execution_System. [Отваряно на 25 Май 2025].

[16] Microsoft, „A tour of the C# language,“ Microsoft, 21 Март 2025. [Онлайн]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview. [Отваряно на 26 Май 2025].

[17] Microsoft, „Overview of ASP.NET Core,“ Microsoft, 21 Април 2025. [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-9.0. [Отваряно на 26 Май 2025].

[18] ZZZ Projects, „What is Entity Framework Core?,“ ZZZ Projects, [Онлайн]. Available: https://www.learnentityframeworkcore.com/. [Отваряно на 26 Май 2025].

[19] jwt.io, „Introduction to JSON Web Tokens,“ jwt.io, [Онлайн]. Available: https://jwt.io/introduction. [Отваряно на 26 Май 2025].

[20] D. Hardt, „The OAuth 2.0 Authorization Framework,“ Internet Engineering Task Force (IETF) , Октомври 2012. [Онлайн]. Available: https://datatracker.ietf.org/doc/html/rfc6749#page-4. [Отваряно на 29 Май 2025].

[21] M. Anicas, „An Introduction to OAuth 2,“ DigitalOcean, 28 Юли 2021. [Онлайн]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. [Отваряно на 29 Май 2025].

[22] M. Nottingham, E. Wilde и S. Dalal, „Problem Details for HTTP APIs,“ Internet Engineering Task Force, Юли 2023. [Онлайн]. Available: https://www.rfc-editor.org/rfc/rfc9457. [Отваряно на 26 Май 2025].

[23] Docker Inc, „What is Docker?,“ Docker Inc, [Онлайн]. Available: https://docs.docker.com/get-started/docker-overview/. [Отваряно на 26 Май 2025].

[24] SmartBear, „What Is OpenAPI?,“ SmartBear, [Онлайн]. Available: https://swagger.io/docs/specification/v3_0/about/. [Отваряно на 29 Май 2025].

[25] GeeksForGeeks, „React Introduction,“ GeeksForGeeks, 8 Април 2025. [Онлайн]. Available: https://www.geeksforgeeks.org/reactjs-introduction/. [Отваряно на 30 Май 2025].

[26] D. Abramov, „RTK Query Overview,“ 23 Февруари 2025. [Онлайн]. Available: https://redux-toolkit.js.org/rtk-query/overview. [Отваряно на 30 Май 2025].

[27] shadcn, „ShadCN Introduction,“ [Онлайн]. Available: https://ui.shadcn.com/docs. [Отваряно на 30 Май 2025].

[28] C. McDonnell, „Zod Introduction,“ [Онлайн]. Available: https://zod.dev/. [Отваряно на 30 Май 2025].

[29] Microsoft, „Monaco - The Editor of the Web,“ Microsoft, [Онлайн]. Available: https://microsoft.github.io/monaco-editor/. [Отваряно на 30 Май 2025].

СПИСЪК НА ФИГУРИТЕ

Фигура 1.1 Опростена архитектура на OJS [2] 7

Фигура 1.2 Работен процес на типична OJS [2] 8

Фигура 1.3 Основни компоненти на OJS [4] 8

Фигура 1.4 Примерна системна архитектура аналогична на Leetcode [6] 14

Фигура 2.1 Use Case диаграма на пилотната система 27

Фигура 2.2 Модел клиент-сървър [13] 36

Фигура 2.3 Примерна диаграма на loose coupled и stateless при REST [14] 36

Фигура 2.4 Sequence диаграма на работа на Judge0 [15] 38

Фигура 2.5 Схема на базата от данни на разработваната система 39

Фигура 2.6 Data flow диаграма на системата 41

Фигура 3.1 Примерен JSON по RFC 9457 спецификацията [22] 47

СПИСЪК НА СЪКРАЩЕНИЯТА

API	Application Programming Interface
CRUD	Create, Read, Update, and Delete
CSS	Cascading Style Sheets
DOM	Document Object Model
EF	Entity Framework
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
JWT	JSON Web Token
LMS	Learning management system
MVP	Minimal viable product
OJS	Online Judge System
ORM	Object-Relational Mapper
RBAC	Role-Based Access Control
REST	Representational State Transfer
SDLC	Software Development Life Cycle
SPA	Single Page Application
SQL	Structured Query Language
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience