АРХИТЕКТУРЕН ПРОЕКТ

Тема на проекта:

"Оценител на задачи по програмиране"

Група 11

Автори:

Стоил Георгиев Стоилов
Радина Иванова Сърбакова
Манас Манасов
Константин Ангелов Чимев
Деян Фархан Азми
Деян Димитров
Тодор Цанков Христов

Дата: 7 януари 2026

1. ВЪВЕДЕНИЕ

Този документ представя архитектурното описание на системата за автоматично оценяване на задачи по програмиране. Системата е предназначена предимно за студенти от първи курс на ФПМИ към ТУ-София и осигурява автоматизирано тестване на C# програми с незабавна обратна връзка чрез интеграция с Moodle.

Архитектурният проект представя техническото решение на изискванията, описани във функционалното описание, като дефинира структурата на системата, нейните компоненти, взаимодействията между тях и начина на внедряване.

1.1. Участници в проекта

Екип за разработка:

- Група 11 - студенти от ФПМИ, ТУ-София

Заинтересовани страни:

- Преподаватели по програмиране - основни потребители, дефиниране на задачи
- Студенти от първи курс - крайни потребители, предаване на решения
- Системни администратори (ако има такива) - инсталация и поддръжка
- Ръководство на ФПМИ - одобрение и ресурси

2. ПРЕДНАЗНАЧЕНИЕ

2.1. Обхват

Настоящият архитектурен проект обхваща проектирането и техническата спецификация на системата за автоматично оценяване на задачи по програмиране. Документът включва:

- Архитектурни изгледи - детайлно описание на структурата и компонентите
- Технологичен стек и инфраструктура
- Интеграция с Moodle и Judge0
- Нефункционални изисквания и тактики за качество
- Модел на данните и база данни

2.2. Актьори

Архитектурата е проектирана да обслужва следните заинтересовани страни:

Актьор	Интерес към проекта
Преподаватели	Use-case изглед, логически изглед, изглед на данните - за разбиране на функционалност и възможности за дефиниране на задачи
Студенти	Use-case изглед, процесен изглед - за разбиране как да предават решения и какво представлява процеса на оценяване
Администратори	Изглед на внедряването, изглед на имплементацията - за инсталация, конфигурация и поддръжка на системата
Разработчици	Всички изгледи - за разбиране на цялостната архитектура и имплементация на системата

2.3. Използвани термини и символи

Термин	Описание
REST API	Representational State Transfer Application Programming Interface - архитектурен стил за комуникация между компоненти
Sandbox	Изолирана среда за изпълнение на код, ограничаваща достъпа до системни ресурси
Docker Container	Леко виртуализирана среда за изолирано изпълнение на приложения
Judge0 CE	Open-source система за компилиране и изпълнение на код в sandbox среда

Термин	Описание
Moodle Assignment	Модул в Moodle за създаване и управление на задачи за предаване
RBAC	Role-Based Access Control - контрол на достъпа базиран на роли
SSO	Single Sign-On - еднократно удостоверяване за достъп до множество системи
PostgreSQL	Релационна база данни с отворен код
Entity Framework	ORM (Object-Relational Mapping) framework за .NET

2.4. Източници

1. Judge0 CE Documentation - https://ce.judge0.com/
2. Moodle Web Services API - https://docs.moodle.org/dev/Web_services
3. Docker Documentation - https://docs.docker.com/
4. ASP.NET Core Documentation - https://docs.microsoft.com/aspnet/core
5. PostgreSQL Documentation - https://www.postgresql.org/docs/
6. Функционално описание на проекта \"Оценител на задачи по програмиране\", Група 11, 19 октомври 2025

3. АРХИТЕКТУРЕН ОБЗОР

Архитектурата на системата е представена чрез шест архитектурни изгледа, всеки от които разглежда системата от различна перспектива и предоставя специфична информация за различните заинтересовани страни. Този подход осигурява цялостно разбиране на системата от различни гледни точки и улеснява комуникацията между участниците в проекта.

По-долу е представено кратко описание на всеки от изгледите. Подробното представяне на всеки изглед, включващо диаграми и детайлно текстово описание, се намира в следващите раздели на документа.

1. Use-case изглед

Представя функционалните възможности на софтуера от гледна точка на крайните потребители - студенти и преподаватели. Този изглед описва основните сценарии на използване (use cases) и взаимодействията между актьорите и системата. Използват се UML Use-case диаграми в комбинация с текстово описание на основните функции. Изгледът е особено важен за преподавателите и студентите, за да разберат какви възможности им предоставя системата.

2. Логически изглед

Представя софтуера като композиция от модули, пакети и компоненти, организирани в многослойна архитектура. Илюстрира връзките между

архитектурно значимите компоненти на системата - Presentation Layer, Application Layer, Infrastructure Layer и External Services. Използват се UML компонентни диаграми за визуализация на структурата. Този изглед е критично важен за разработчиците и архитекти, за да разберат как е организиран системният код.

3. Процесен изглед

Описва динамичното поведение на системата чрез представяне на алгоритмите, реализиращи архитектурно значимите функции. Фокусира се върху процеса на автоматично оценяване на предадени решения - от момента на предаване до получаване на обратна връзка. Използват се UML Activity диаграми в комбинация с текстово описание на всяка стъпка. Изгледът помага на всички участници да разберат как работи системата в реално време.

4. Изглед на данните

Представя модела на обработваните данни, включващ логическите връзки между тях. Детайлно описва схемата на релационната база данни PostgreSQL с всички таблици, полета, типове данни и релации (1-1, 1-N, N-M). Използват се ER (Entity-Relationship) диаграми за графично представяне. Този изглед е от ключово значение за разработчиците и администраторите на бази данни.

5. Изглед на внедряването

Описва физическата инфраструктура и разпределението на софтуерните компоненти върху хардуерните ресурси. Представя трите основни сървъра (Web Server, Application Server, Database Server), тяхната конфигурация, мрежовата топология и организацията в три зони за сигурност (DMZ, Internal, Data). Използват се Deployment диаграми. Изгледът е критично важен за системните администратори, отговорни за инсталация и поддръжка.

6. Изглед на имплементацията

Представя детайли от гледна точка на имплементацията, включително концепцията за слоеве, технологичния стек и правилата за реализация. Описва организацията на кода в отделни проекти (API, Application, Domain, Infrastructure), naming conventions, dependency injection и други best practices. Използват се слоеви диаграми (Layer diagrams) за визуализация. Този изглед е предназначен основно за разработчиците, които ще имплементират системата.

3.1. Use-case изглед

3.1.1. Общо описание

Use-case изгледът представя функционалността на системата от гледна точка на основните актьори - студенти и преподаватели. Системата осигурява два основни сценария на използване:

- **Управление на задачи** - преподавателите създават задачи, дефинират тестове и наблюдават напредъка
- **Предаване и оценяване** - студентите предават решения и получават автоматична обратна връзка

3.1.2. Основни use cases

UC-1: Създаване на задача

- **Актьор:** Преподавател

- **Предусловие:** Преподавателят е автентикиран в системата
- **Основен сценарий:**
  - Преподавателят избира опция за създаване на нова задача
  - Въвежда име, описание, максимални точки
  - Конфигурира ограничения (време, памет, диск)
  - Добавя тестови случаи с входове/изходи
  - Опционално качва еталонно решение за генериране на изходи
  - Свързва задачата с Moodle Assignment
  - Системата запазва задачата и потвърждава създаването
- **Следусловие:** Задачата е създадена и достъпна за предаване от студенти

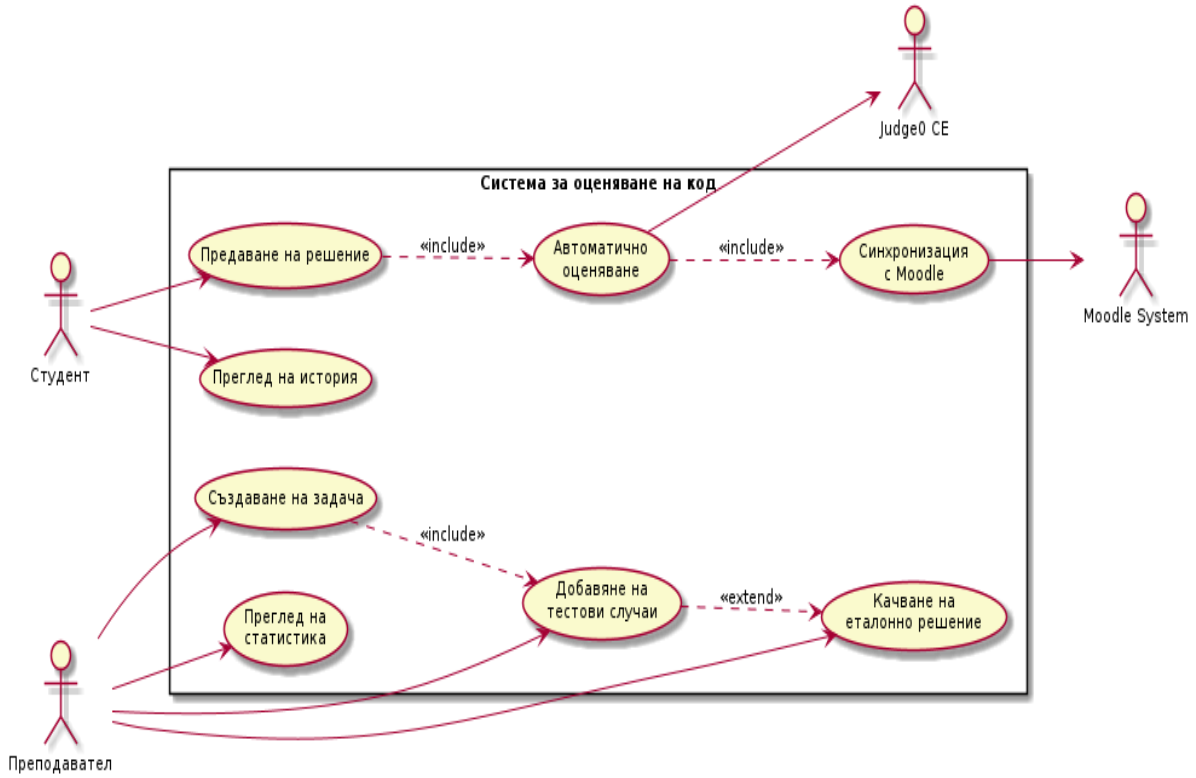
UC-2: Предаване на решение

- **Актьор:** Студент
- **Предусловие:** Студентът е автентикиран чрез Moodle
- **Основен сценарий:**
  - Студентът отваря Assignment в Moodle
  - Пише C# код в текстовото поле
  - Натиска бутон за предаване
  - Системата записва предаването и стартира автоматично оценяване
  - Кодът се компилира в sandbox среда
  - Изпълняват се всички тестови случаи
  - Изчислява се оценка и генерира обратна връзка
  - Резултатът се записва в Moodle Grade и Feedback
  - Студентът вижда оценката и обратната връзка в Moodle
- **Алтернативен сценарий - компилационна грешка:**
  - При компилационна грешка се връща пълното съобщение от компилатора
  - Оценката е 0
- **Алтернативен сценарий - timeout:**
  - При превишаване на лимита за време се прекратява изпълнението
  - Връща се съобщение за timeout с използвано време
- **Следусловие:** Предаването е записано, оценено и оценката е налична в Moodle

UC-3: Преглед на статистика

- **Актьор:** Преподавател
- **Предусловие:** Има предадени решения от студенти
- **Основен сценарий:**
  - Преподавателят избира задача или студент
  - Преподавателят може да прегледа всички опити на студент
  - Може да вижда код, резултати и обратна връзка
  - Опционално експортира данните в CSV/Excel

3.1.3. Диаграма на use cases



3.2. Логически изглед

3.2.1. Общо описание

Логическият изглед представя архитектурата на системата като композиция от модули, слоеве и компоненти. Системата следва многослойна архитектура с ясно разделение на отговорностите:

- **Presentation Layer** - потребителски интерфейс (Moodle Plugin + Web Interface)
- **Application Layer** - бизнес логика (ASP.NET Core Web API)
- **Infrastructure Layer** - достъп до данни и външни услуги
- **External Services** - Judge0 CE, Moodle API

3.2.2. Основни компоненти

Компонент	Отговорности
Moodle Plugin	• Разширява Assignment модула с функционалност за код оценяване • Осигурява интерфейс за предаване на C# код • Комуникира с Backend API за оценяване • Показва резултати и обратна връзка
Web Interface (HTML/CSS/Java Script)	• Интерфейс за преподаватели за управление на задачи • Преглед на детайлна статистика • Експорт на данни

Компонент	Отговорности
Backend API	• REST API за всички операции • Бизнес логика за оценяване • Управление на задачи и предавания • Оркестрация на Judge0 извиквания • Изчисляване на оценки • Синхронизация с Moodle
Judge0 Service	• Компилиране на C# код • Изпълнение в изолирана Docker среда • Контрол на ресурси (време, памет, диск) • Връщане на резултати
Database Access	• Entity Framework Core за ORM • Repository pattern за достъп до данни • CRUD операции за всички entity
Moodle Integration	• Moodle Web Services API клиент • Синхронизация на потребители и курсове • Записване на оценки • SSO автентикация

3.2.3. Слоева архитектура

Системата е организирана в три основни слоя с ясно дефинирани отговорности:

Presentation Layer

- **Moodle Plugin (PHP)** - Assignment подтип за оценяване на код
- **Web Application (HTML/CSS/JavaScript)** - административен интерфейс
  - ➔ Single Page Application (SPA) с Vanilla JavaScript
  - ➔ Bootstrap 5 за UI компоненти и responsive дизайн
  - ➔ Client-side routing с History API
  - ➔ Fetch API за асинхронна комуникация
- Комуникира само с Application Layer чрез REST API

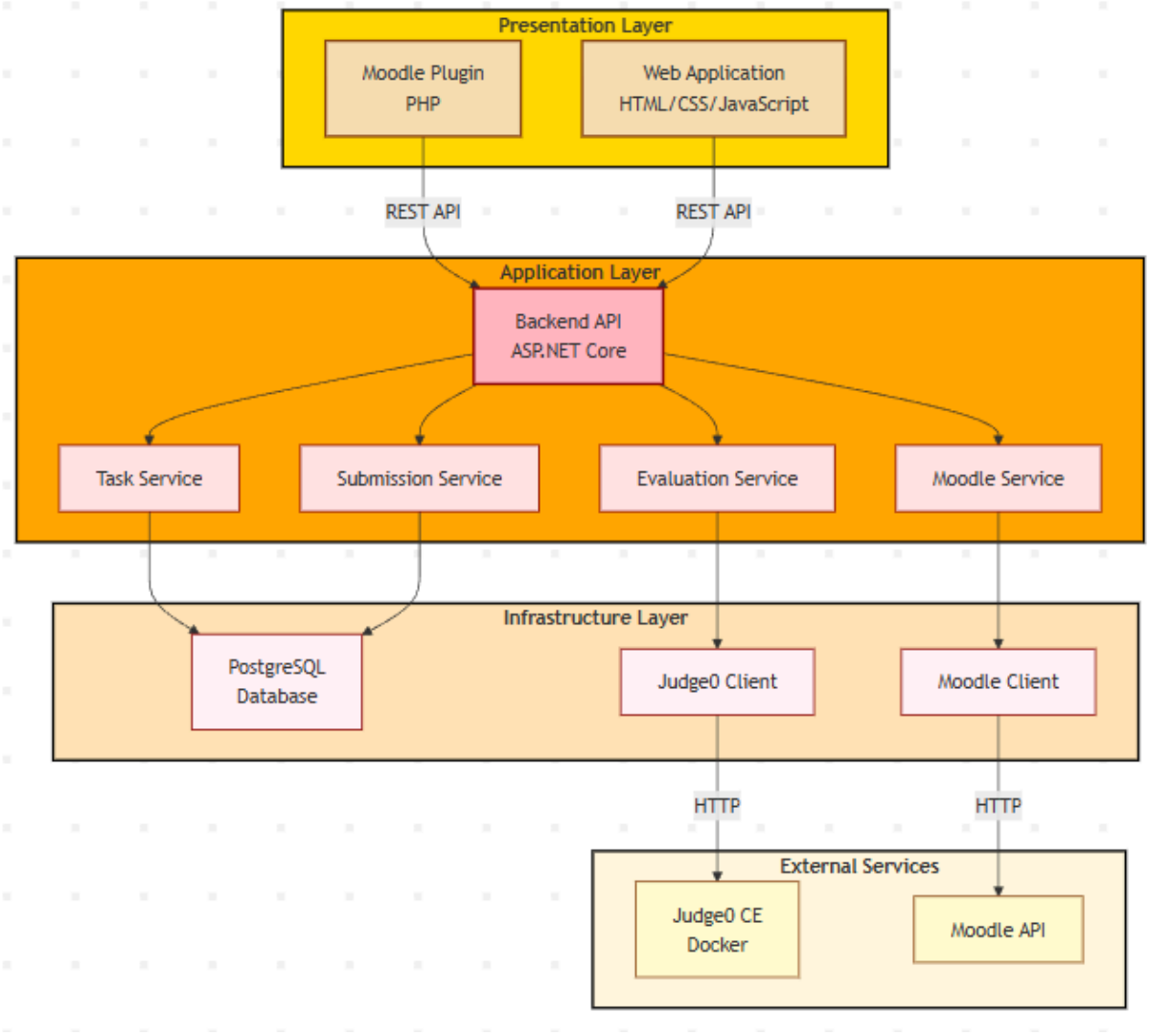
Application Layer

- **Controllers** - REST endpoints за всички операции
- **Services** - бизнес логика (TaskService, SubmissionService, EvaluationService)
- **DTOs** - обекти за трансфер на данни
- **Validators** - валидация на входни данни
- Не зависи от конкретни технологии за достъп до данни

Infrastructure Layer

- **Repositories** - абстракция над Entity Framework
- **DbContext** - Entity Framework конфигурация
- **Judge0Client** - HTTP клиент за Judge0 API
- **MoodleClient** - HTTP клиент за Moodle Web Services

3.2.4. Диаграма на класове (опростена)



3.3. Процесен изглед

3.3.1. Общо описание

Процесният изглед описва динамичното поведение на системата, включително алгоритмите за автоматично оценяване и взаимодействието между компонентите. Основният процес е автоматичното оценяване на предадено решение.

3.3.2. Процес на автоматично оценяване

Този процес е централен за функционалността на системата и включва следните стъпки:

Стъпка 1: Предаване на код

7. Студентът предава C# код чрез Moodle Assignment
8. Moodle Plugin извиква Backend API endpoint: POST /api/submissions
9. Backend създава запис Submission в базата данни със статус 'Pending'

Стъпка 2: Зареждане на задача и тестове

10. Зареждане на Task с всички TestCase записи
11. Зареждане на конфигурация (time_limit, memory_limit, disk_limit)

Стъпка 3: Компилация

12. Изпращане на код до Judge0: POST /submissions?base64_encoded=true
13. Параметри: language_id=51 (C#), source_code (base64)
14. Judge0 връща token за проследяване
15. Polling на GET /submissions/:token до статус != 'In Queue' и != 'Processing'
16. При компилационна грешка: статус 'Compilation Error', запазване на stderr, прекратяване

Стъпка 4: Изпълнение на тестове

17. За всеки TestCase:
    - Изпращане на нов submission до Judge0 с input_file (base64)
    - Параметри: cpu_time_limit, memory_limit, max_file_size
    - Polling до получаване на резултат
    - Сравняване на stdout с expected_output
    - Създаване на TestResult със статус: 'Pass', 'Fail', 'Timeout', 'Runtime Error'
    - Записване на execution_time_ms и memory_used_mb

Стъпка 5: Изчисляване на оценка

18. Преброяване на успешни тестове: passed_count
19. Формула: grade = (passed_count / total_tests) × max_points
20. Закръгляване до 2 десетични знака

Стъпка 6: Генериране на обратна връзка

21. При компилационна грешка: връща компилационната грешка
22. При неуспешен тест: "Неуспешно изпълнение на тест "<Име на тестовия случай>" - резултатът се различава от очакваното."
23. При timeout: "Програмата не преминава тестът "<Име на тестовия случай>" за достатъчно кратко време."
24. При disk limit: "Програмата не преминава тестът "<Име на тестовия случай>" поради изчерпване на позволеното дисково пространство."
25. При успех: "Поздравления! Всички тестове са преминати успешно!"

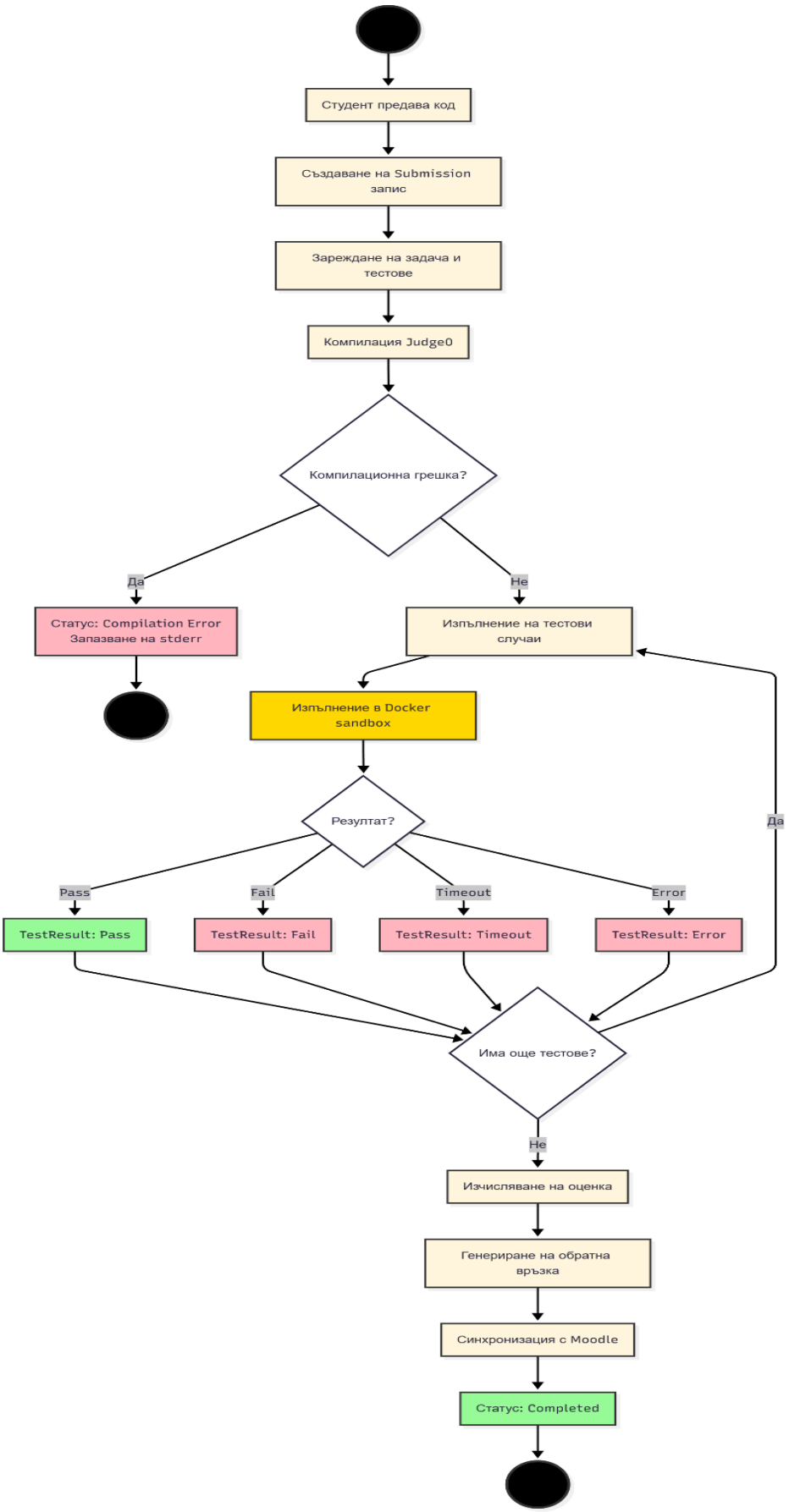
Стъпка 7: Синхронизация с Moodle

26. Извикване на Moodle Web Service: mod_assign_save_grade
27. Параметри: assignmentid, userid, grade, attemptnumber, feedback
28. Moodle записва оценката и обратната връзка
29. Студентът вижда резултата в Moodle веднага

Стъпка 8: Финализация

30. Обновяване на Submission статус: 'Completed'
31. Запазване на final_grade и feedback
32. Връщане на резултат към Moodle Plugin

3.3.3. Диаграма на процеса


Студент предава код
Създаване на Submission запис
Зареждане на задача и тестове
Компилация Judge0
Компилационна грешка?
Да
Не
Статус: Compilation Error
Запазване на stderr
Изпълнение на тестови случаи
Изпълнение в Docker sandbox
Резултат?
Pass
Fail
Timeout
Error
TestResult: Pass
TestResult: Fail
TestResult: Timeout
TestResult: Error
Има още тестове?
Да
Не
Изчисляване на оценка
Генериране на обратна връзка
Синхронизация с Moodle
Статус: Completed

3.4. Изглед на данните

3.4.1. Общо описание

Изгледът на данните представя структурата на релационната база данни PostgreSQL и връзките между таблиците. Моделът е проектиран да поддържа всички функционални изисквания с ясни релации и нормализация до BCNF (Boyce-Codd Normal Form).

3.4.2. Основни таблици

Базата данни включва следните основни таблици:

Users

Съхранява информация за потребители (студенти и преподаватели).

Поле	Тип	Описание
user_id	INT	Primary Key, Auto Increment
moodle_user_id	INT	Уникален идентификатор от Moodle, NOT NULL, UNIQUE
username	VARCHAR(100)	Потребителско име, NOT NULL
email	VARCHAR(255)	Email адрес, NOT NULL
role	VARCHAR(50)	Роля: 'Student', 'Teacher', 'Admin'

Tasks

Съхранява дефинициите на задачите.

Поле	Тип	Описание
task_id	INT	Primary Key, Auto Increment
title	VARCHAR(255)	Заглавие на задачата, NOT NULL
description	TEXT	Подробно описание
max_points	DECIMAL(5,2)	Максимални точки за задачата
time_limit_ms	INT	Лимит време в милисекунди, DEFAULT 5000
memory_limit_mb	INT	Лимит памет в MB, DEFAULT 256

Поле	Тип	Описание
moodle_assignment_id	INT	Връзка с Moodle Assignment, UNIQUE
created_by	INT	Foreign Key към Users, NOT NULL

TestCases
Съхранява тестови случаи за всяка задача.

Поле	Тип	Описание
testcase_id	INT	Primary Key, Auto Increment
task_id	INT	Foreign Key към Tasks, NOT NULL
name	VARCHAR(100)	Име на теста, NOT NULL
input_data	TEXT	Входни данни за теста
expected_output	TEXT	Очакван изход, NOT NULL
is_public	BOOLEAN	Видимост за студенти, DEFAULT false
points	DECIMAL(5,2)	Точки за теста (за weighted scoring)

Submissions
Съхранява предадени решения от студенти.

Поле	Тип	Описание
submission_id	INT	Primary Key, Auto Increment
task_id	INT	Foreign Key към Tasks, NOT NULL
student_id	INT	Foreign Key към Users, NOT NULL
source_code	TEXT	Предаден C# код, NOT NULL
submission_date	TIMESTAMP	Дата и час на предаване, DEFAULT CURRENT_TIMESTAMP
status	VARCHAR(50)	'Pending', 'Processing', 'Completed', 'Error'

Поле	Тип	Описание
final_grade	DECIMAL(5,2)	Финална оценка
feedback	TEXT	Обратна връзка за студента

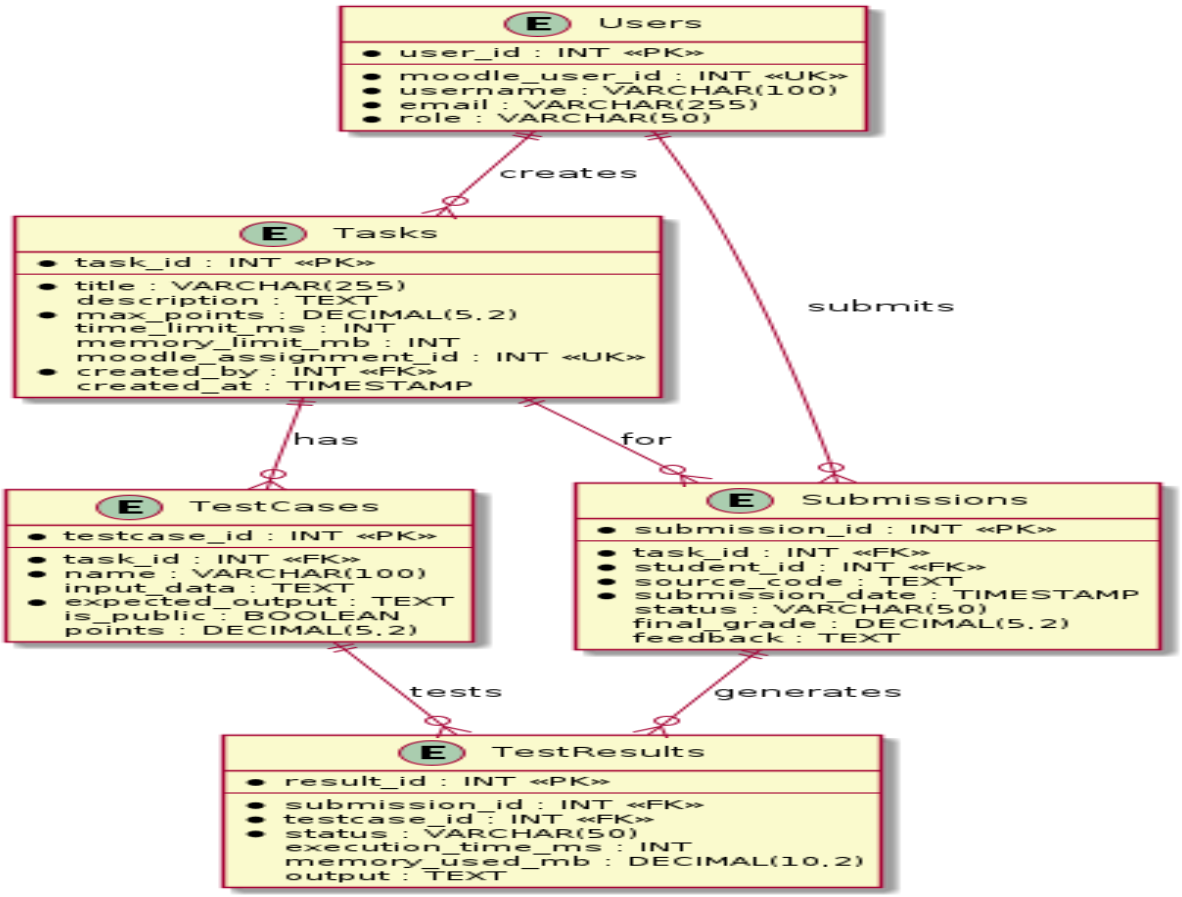
TestResults

Съхранява резултати от изпълнение на тестови случаи.

Поле	Тип	Описание
result_id	INT	Primary Key, Auto Increment
submission_id	INT	Foreign Key към Submissions, NOT NULL
testcase_id	INT	Foreign Key към TestCases, NOT NULL
status	VARCHAR(50)	'Pass', 'Fail', 'Timeout', 'Runtime Error', 'Memory Limit'
execution_time_ms	INT	Време за изпълнение в ms
memory_used_mb	DECIMAL(10,2)	Използвана памет в MB
output	TEXT	Получен изход от програмата

3.4.3. Релации

- Users (1) : Tasks (N) - един преподавател може да създаде много задачи
- Tasks (1) : TestCases (N) - една задача има много тестови случаи
- Tasks (1) : Submissions (N) - за една задача има много предавания
- Users (1) : Submissions (N) - един студент може да предаде много решения
- Submissions (1) : TestResults (N) - едно предаване има много резултати от тестове
- TestCases (1) : TestResults (N) - един тестов случай се изпълнява за много предавания



3.5. Изглед на внедряването

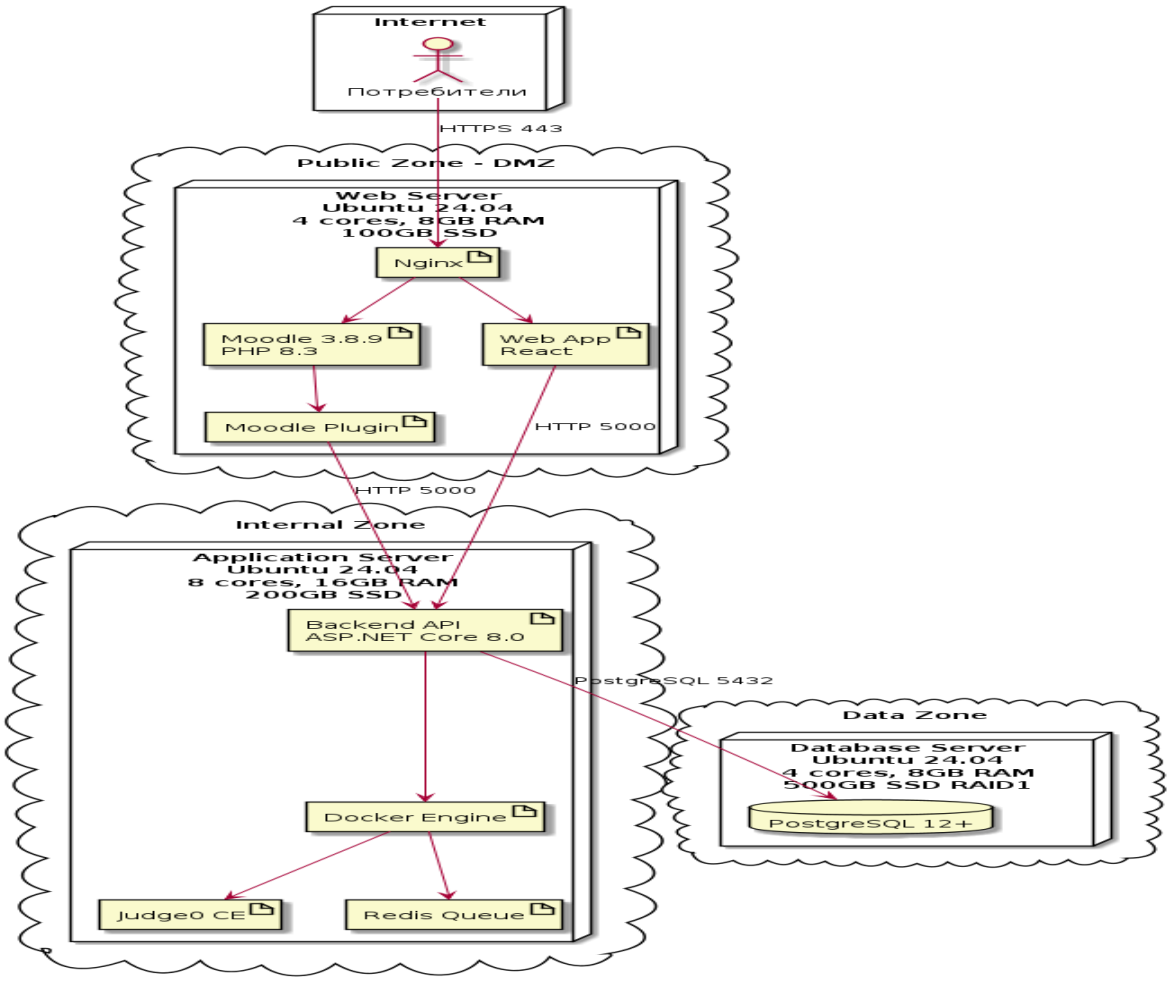
3.5.1. Общо описание

Изгледът на внедряването описва физическата инфраструктура и разпределението на софтуерните компоненти върху хардуерните ресурси. Системата е проектирана за deployment в on-premise среда с три основни сървъра.

3.5.2. Хардуерна конфигурация

Сървър	Спецификация
Web Server	• CPU: 4 cores (2.4 GHz+) • RAM: 8 GB • Storage: 100 GB SSD • OS: Ubuntu Server 24.04 LTS • Software: Nginx, PHP 8.3, Moodle 3.8.9
Application Server	• CPU: 8 cores (2.8 GHz+) • RAM: 16 GB • Storage: 200 GB SSD • OS: Ubuntu Server 24.04 LTS • Software: .NET 8.0 Runtime, Docker, Judge0 CE

Сървър	Спецификация
Database Server	• CPU: 4 cores (2.4 GHz+) • RAM: 8 GB • Storage: 500 GB SSD (с RAID 1 за redundancy) • OS: Ubuntu Server 24.04 LTS • Software: PostgreSQL 12+



3.5.3. Разпределение на компоненти

Сървър	Компоненти
Web Server	• Moodle 3.8.9 (PHP) • Moodle Plugin за Code Evaluation • Web Application (Single Page Application с Vanilla JavaScript) за преподаватели • Nginx Web Server

Сървър	Компоненти
Application Server	• Backend API (ASP.NET Core) • Judge0 CE (Docker containers) • Redis (за Judge0 queue) • Docker Engine
Database Server	• PostgreSQL 12+ Database • Automated backups (daily)

3.6. Изглед на имплементацията

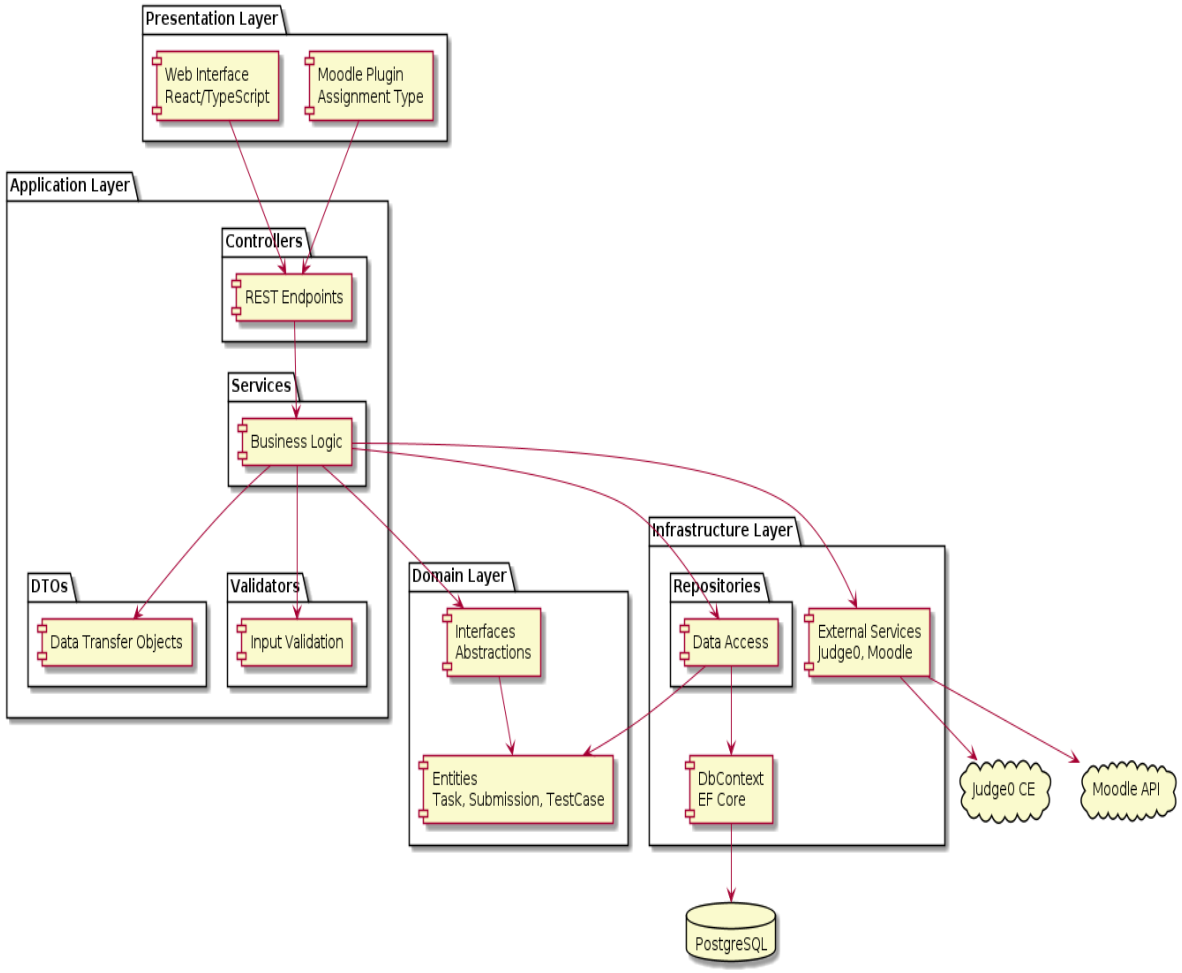
3.6.1. Общо описание

Изгледът на имплементацията описва организацията на кода, технологичния стек и правилата за имплементация. Системата следва Clean Architecture принципи с ясно разделение на слоевете.

3.6.2. Технологичен стек

Компонент	Технологии
Backend API	• ASP.NET Core 8.0 Web API • C# 12 • Entity Framework Core 8.0 • FluentValidation за валидация • Swagger/OpenAPI за документация
Web Interface	• Single Page Application (SPA) с Vanilla JavaScript • Fetch API за HTTP requests • Bootstrap 5 за UI компоненти
Moodle Plugin	• PHP 8.3+ • Moodle Assignment Plugin API • JavaScript за frontend взаимодействие
Database	• PostgreSQL 12+ • Entity Framework Core Migrations
Code Execution	• Judge0 CE (open-source) • Docker Containers • Redis за queue management

3.6.2.1. Диаграма на слоевете



3.6.3. Организация на кода (Backend)

Backend проектът е организиран в следната структура:

CodeEvaluator.API

- Controllers/ - REST API endpoints
- Middleware/ - authentication, error handling
- Program.cs, appsettings.json

CodeEvaluator.Application

- Services/ - бизнес логика:
  - TaskService - управление на задачи
  - SubmissionService - обработка на предавания
  - Judge0Service - комуникация с Judge0 API
  - MoodleAuthService - Moodle автентикация
  - Judge0PollingService - фонов service за проверка на статуси
- DTOs/ - Data Transfer Objects
- Validators/ - FluentValidation правила
- Interfaces/ - абстракции

CodeEvaluator.Domain

- Entities/ - domain models:
  - User - потребители на системата

- Task - програмни задачи
- TestCase - тестови случаи
- Submission - предадени решения
- TestResult - резултати от тестове
- ReferenceSolution - еталонни решения
- AdditionalFile - допълнителни файлове към задачи

- Enums/ - статуси и константи

CodeEvaluator.Infrastructure

- Data/ - DbContext, Repositories, Migrations
- ExternalServices/ - Judge0Client, MoodleClient
- Configuration/ - Entity configurations

3.6.4. Правила за имплементация

Naming Conventions

- PascalCase за класове, методи, properties
- camelCase за локални променливи и параметри
- Префикс I за interfaces (ITaskService)

Dependency Injection

- Всички зависимости се инжектират чрез constructor
- Services регистрирани в Program.cs
- Използване на interfaces за loose coupling

Error Handling

- Global exception handler middleware
- Custom exception types за различни сценарии
- Basic .NET logging (Console, Debug) за development

Testing

- Manual testing protocol за всички функционалности
- Structured checklist за проверка на submission flow

4. НЕФУНКЦИОНАЛНИ ИЗИСКВАНИЯ

Тази секция описва как избраната архитектура реализира атрибутите на качеството чрез употреба на различни тактики и архитектурни решения.

4.1. Производителност

4.1.1. Изисквания

- Зареждане на страници в Moodle: < 5 секунди
- Компилация на C# код: 3-5 секунди
- Пълно оценяване: < 2 минути за 10 теста
- Peak load: 50 едновременни предавания
- Concurrent users: 300-500 студенти

4.1.2. Тактики

- **Асинхронна обработка:** оценяването се извършва асинхронно, за да не блокира потребителския интерфейс
- **Connection pooling:** PostgreSQL connection pool за намаляване на латентност при достъп до база данни

- **Caching:** кеширане на често използвани данни (task definitions, test cases) в памет
- **Parallel execution:** паралелно изпълнение на тестови случаи когато е възможно
- **Resource limits:** ясни лимити за време и памет предотвратяват resource exhaustion

4.2. Сигурност

4.2.1. Изисквания

- Изолирано изпълнение на код без достъп до системни ресурси
- Автентикация чрез Moodle session
- Защита срещу SQL injection атаки
- Основна input validation

4.2.2. Тактики

- **Judge0 sandbox:** Използва готова Judge0 CE инсталация (Docker-based) за изолирано изпълнение с resource limits
- **Moodle authentication:** Автентикация чрез съществуващата Moodle сесия
- **SQL injection prevention:** Entity Framework Core с parameterized queries (по подразбиране)
- **Input validation:** FluentValidation за основна валидация на входни данни

Забележка: *Имплементацията е базирана на готови инструменти (Judge0, EF Core) и стандартни практики. Advanced security features се планират за production версии.*

4.3. Достъпност

4.3.1. Изисквания

- Стабилна работа при нормално натоварване (до 50 едновременни submissions)
- Основна error handling за graceful failures

4.3.2. Тактики

- Basic error logging с built-in .NET logging
- Забележка: Advanced monitoring и automated backups са планирани за бъдещи версии

4.4. Разширяемост

4.4.1. Изисквания

- Интеграция с други Learning Management Systems (IMoodleAuthService)
- Добавяне на нови типове тестове

4.4.2. Тактики

- **Layered architecture:** Разделение на presentation, application и data layers за организация на кода
- **Dependency injection:** Built-in DI в ASP.NET Core за основни services
- **Interface-based design:** Интерфейси за repositories (ITaskRepository, ISubmissionRepository)

- **External services:** Judge0 и Moodle са външни услуги, достъпни чрез HTTP API

4.5. Използваемост

4.5.1. Изисквания

- Функционален интерфейс за основните задачи
- Clear error messages
- Работещ UI на desktop browsers

4.5.2. Тактики

- Bootstrap за consistent styling (минимално усилие)
- Built-in validation messages
- Basic user feedback (alerts, status messages)
- Забележка: Responsive design и advanced UX са планирани за бъдещи версии

4.6. Възможност за тестване

4.6.1. Изисквания

- Basic unit tests за критични компоненти (EvaluationService, Judge0Client)
- Integration test за пълния submission flow

4.6.2. Тактики

- **Dependency injection:** позволява лесно mocking
- **Interface-based design:** улеснява създаването на test doubles
- **Separation of concerns:** всеки компонент може да се тества независимо

5. ЗАКЛЮЧЕНИЕ

Представеният архитектурен проект описва техническото решение на система за автоматично оценяване на задачи по програмиране. Целта е баланс между функционалност и реалистичен развоен цикъл чрез използване на утвърдени външни инструменти и съвременни технологии.

Системата следва многослойна архитектура с ясно разделение между Presentation, Application и Infrastructure слоеве. Presentation слоят използва Vanilla JavaScript и Bootstrap 5 за изграждане на SPA интерфейс, докато Application слоят реализира бизнес логиката чрез ASP.NET Core Web API, Entity Framework Core и FluentValidation.

Сигурността се гарантира чрез battle-tested решения: Judge0 CE осигурява sandbox среда за изпълнение на код, Entity Framework Core предпазва от SQL injection, а Moodle сесии елиминират нуждата от собствена автентикация.

Интеграцията с Moodle чрез plugin и Web Services API осигурява безпроблемно внедряване в съществуваща образователна среда. Judge0 CE позволява незабавна поддръжка на C# и лесно разширение към други езици.

MVP обхватът включва пълния процес от създаване на задача до автоматично оценяване и обратна връзка.Тестването се извършва ръчно въз основа на

структурирани списъци (checklists), а архитектурата за внедряване е максимално опростена и разчита на един или два сървъра.

Проектът е подготвен за бъдещо развитие с възможности за добавяне на автоматизирани тестове, HTTPS/TLS, ролеви достъп, мониторинг и подобрения в UX. Пътната карта започва с инсталация на Judge0 CE и PostgreSQL, разработка на backend и Moodle plugin, и изграждане на административен интерфейс.

Фазата на потребителско приемане с избрана група преподаватели и студенти ще осигури ценна обратна връзка и валидиране на архитектурните решения, като постави основа за мащабируемо институционално внедряване.