

ФУНКЦИОНАЛНО ОПИСАНИЕ

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: 07 януари 2026

1. ВЪВЕДЕНИЕ

Системата за автоматично оценяване на задачи по програмиране е предназначена за студенти от първи курс на ФПМИ към ТУ-София. Интегрира се с Moodle 3.8.9 и осигурява автоматизирано тестване на C# програми с незабавна обратна връзка.

Предметна област: Висше образование - обучение по програмиране на начално ниво (C#)

Проблем: Ръчната проверка на задачи води до забавена обратна връзка (3-7 дни), липса на многократни опити и висока натовареност на преподавателите.

Цел: Автоматизиране на оценяването за осигуряване на незабавна обратна връзка (секунди), обективност и намалена натовареност.

Очаквани резултати: Съкращаване на времето за обратна връзка от дни до секунди, освобождаване на 3-17 часа седмично (за 100 человека - всеки по 2-10 минути) за преподавателите, повишена обективност и детайлна аналитика.

2. ОБХВАТ НА ПРОЕКТА

2.1. Обща информация

2.1.1. Предметна област

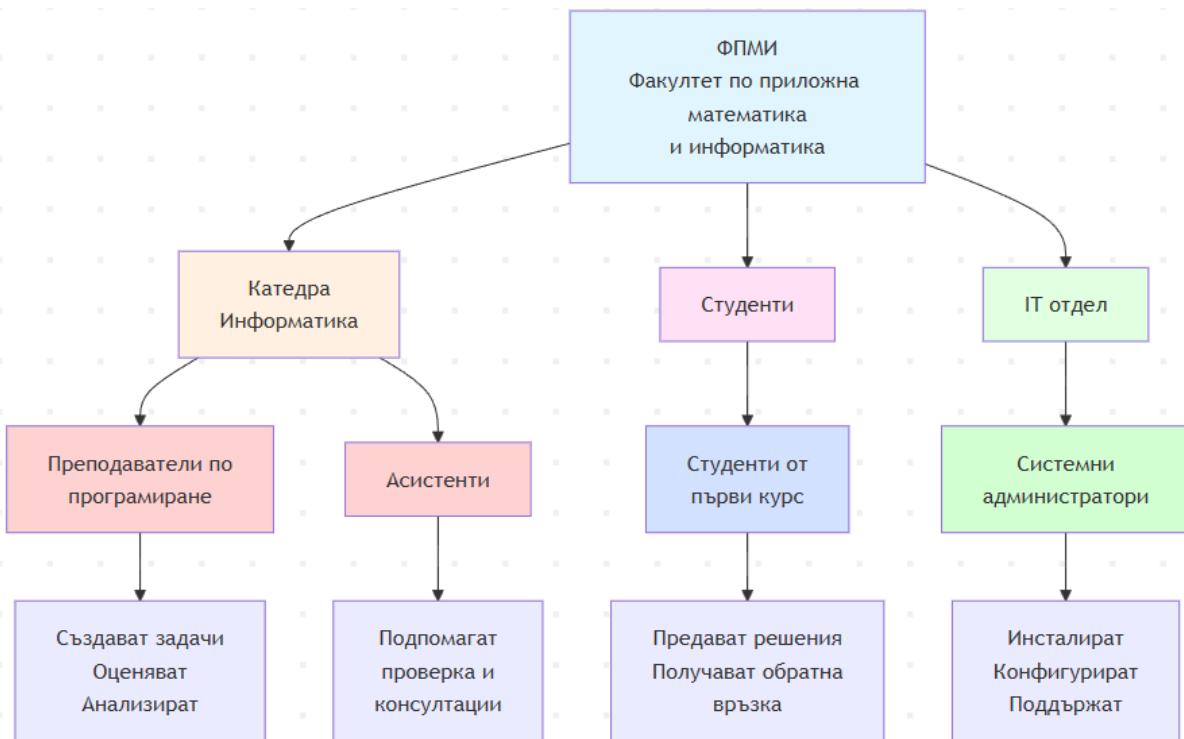
Обучението по програмиране във ФПМИ включва теоретично обучение и практическо упражнение чрез решаване на задачи с нарастваща сложност. Студентите използват C# за изучаване на основи на императивното програмиране.

Нови термини:

- **Assignment** - задача в Moodle за предаване на решения
- **Sandbox изпълнение** - изолирана среда за код, ограничаваща системния достъп
- **Тестов случай** - двойка входни данни и очакван изход
- **Еталонно решение** - коректна програма от преподавателя

Източници: Judge0 CE Documentation (<https://ce.judge0.com/>), Moodle Web Services API

2.1.2. Организационна структура



Ролеви взаимодействия:

- Преподаватели:** Дефинират задачи, наблюдават напредък, анализират статистика
- Студенти:** Получават задачи, разработват решения, получават автоматична обратна връзка
- Администратори:** Инсталират и поддържат системата, управляват права

2.1.3. Необходимост от софтуер

Проблеми: Забавена обратна връзка (дни), непоследователност в оценяването, ограничени опити, висока натовареност на преподавателите, затруднен мониторинг, липса на аналитика.

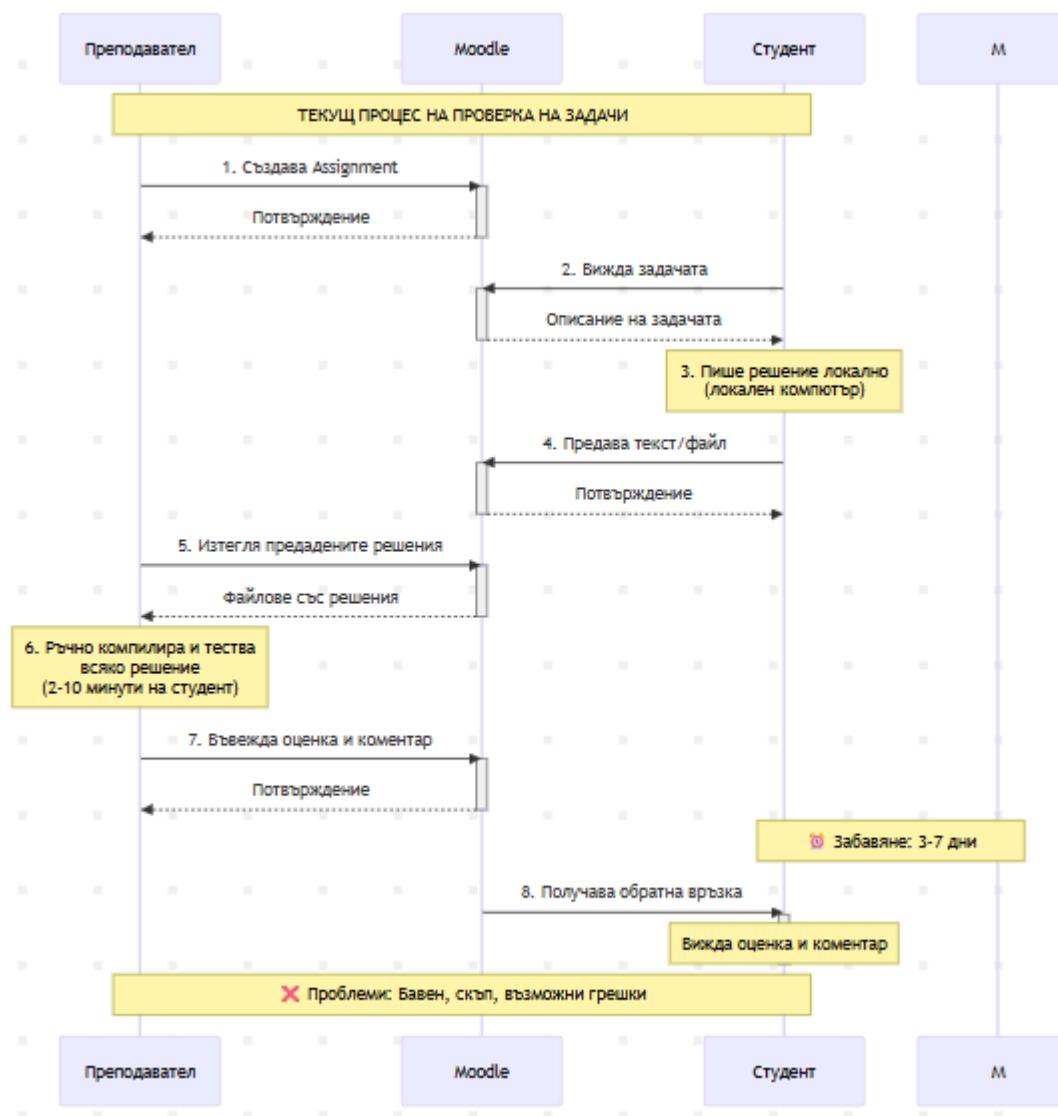
Ползи: Незабавна обратна връзка, многократни опити, обективно оценяване, освобождаване на 10-15 часа седмично, детайлна статистика, повишена ангажираност на студентите.

2.1.4. Целеви групи

- Преподаватели (основна):** Експерти, нужди: гъвкави инструменти, подробна аналитика
- Асистенти:** Подпомагат преподавателите, достъп до резултати
- Студенти :** бърза обратна връзка

2.2. Бизнес процеси в организацията

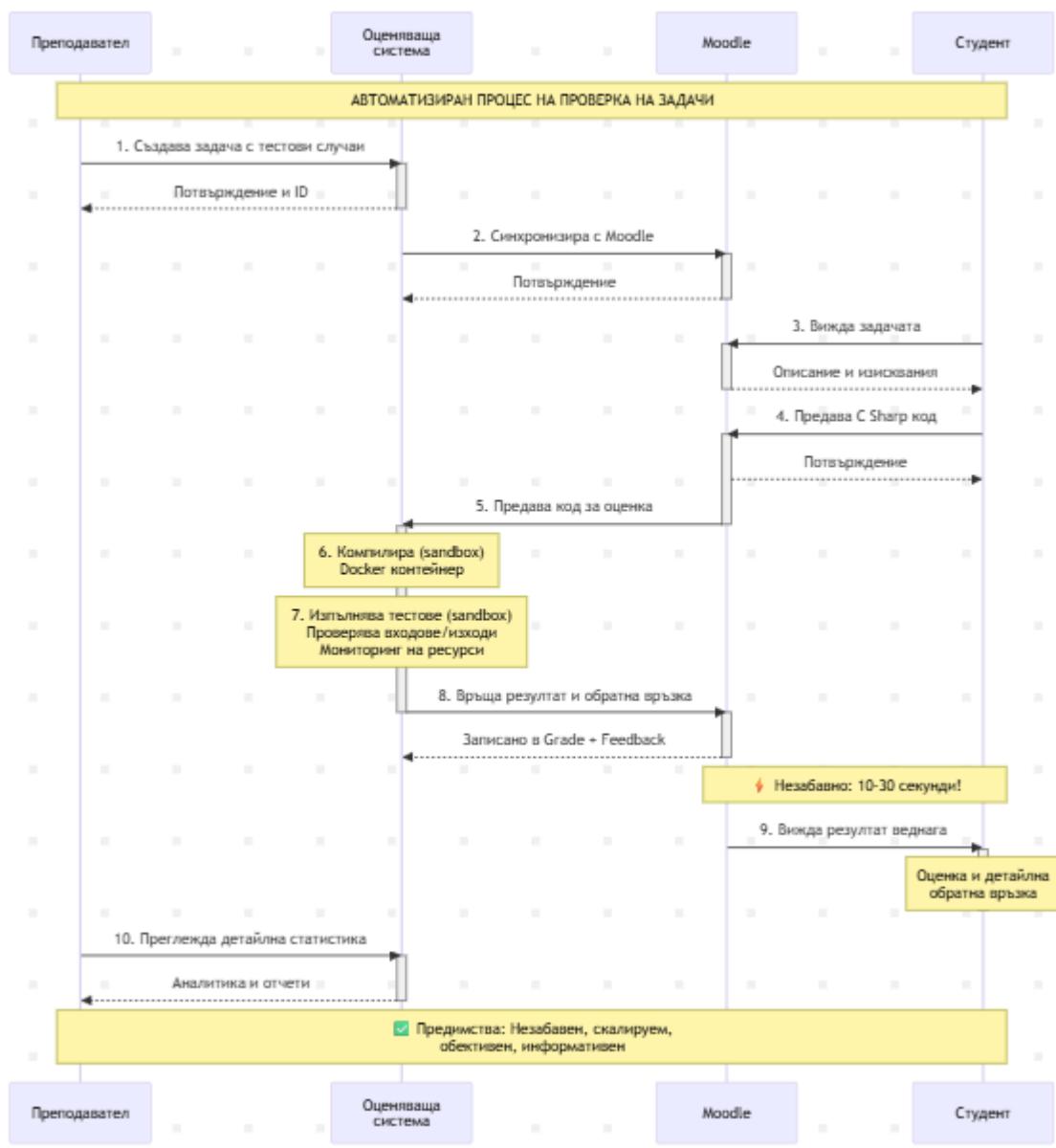
2.2.1. Текущ процес (AS-IS)



Стъпки: Преподавател създава Assignment → Студент предава код → Преподавател изтегля → Ръчно компилира и тества (2-10 мин/студент) → Въвежда оценка → Студент получава обратна връзка след 3-7 дни.

Проблеми: Бавен, скъп (3-17 часа за 100 студента), възможни грешки, нестандартизиран.

2.2.2. Целеви процес (ТО-ВЕ)



Стъпки: Преподавател създава задача с тестове → Синхронизира с Moodle → Стudent предава код → Автоматично компилиране и изпълнение (sandbox) → Незабавен резултат → Преподавател преглежда детайлна статистика.

Предимства на новия процес:

1. Незабавен (секунди)

- Студентът получава резултата за 10-30 секунди вместо да чака 3-7 дни
- Веднага вижда дали решението му работи правилно

2. Скалируем

- Системата може да обработва много студенти едновременно (стотици)

- Ако има 100 студента, които предават по едно време, всички получават резултат едновременно
- Не зависи от броя преподаватели - 1 преподавател може да управлява 500+ студента

3. Обективен

- Еднакви критерии за всички студенти
- Няма субективност - компютърът не може да бъде пристрастен
- Всеки код се оценява по точно същите тестове
- Преподавател А и преподавател Б ще дадат абсолютно същата оценка за един код

4. Информативен

- Системата дава детайлна обратна връзка:
 - Кои тестове са минали
 - Кои тестове са failed
 - Къде точно е грешката
 - Време за изпълнение
 - Използвана памет
 - Компилационни грешки с точни редове
- Преподавателите получават статистика и аналитика за целия курс

2.3. Логически модел на данните

Логическият модел описва основните информационни обекти, с които работи системата, представени по начин достъпен и за неспециалисти. Детайлната техническа схема на базата данни с всички служебни полета (ID-та, външни ключове) и точни типове данни (VARCHAR, DECIMAL и др.) се намира в архитектурния проект.

2.3.1. Основни типове данни

Задача (Task)

- Заглавие на задачата
- Описание на задачата
- Максимални точки, които може да се получат
- Лимит на време за изпълнение (в милисекунди)
- Лимит на памет за изпълнение (в мегабайти)
- Връзка към съответната задача в Moodle
- Автор на задачата (преподавател)

Тестов случай (TestCase)

- Име на теста
- Входни данни за проверка
- Очакван изходен резултат
- Видимост за студентите (публичен или скрит тест)

- Тегло на теста в общата оценка
- Към коя задача принадлежи

Предаване на решение (Submission)

- Предаден програмен код
- Дата и час на предаване
- Статус на обработка (изчакващо, в процес, завършено, грешка)
- Получена оценка
- Обратна връзка към студента
- Кой студент е предал
- За коя задача е предадено

Резултат от тест (TestResult)

- Статус на изпълнение (успешен, неуспешен, timeout, грешка)
- Реално използвано време за изпълнение
- Реално използвана памет
- Получен изход от програмата
- Към кое предаване принадлежи
- Кой тестов случай е изпълнен

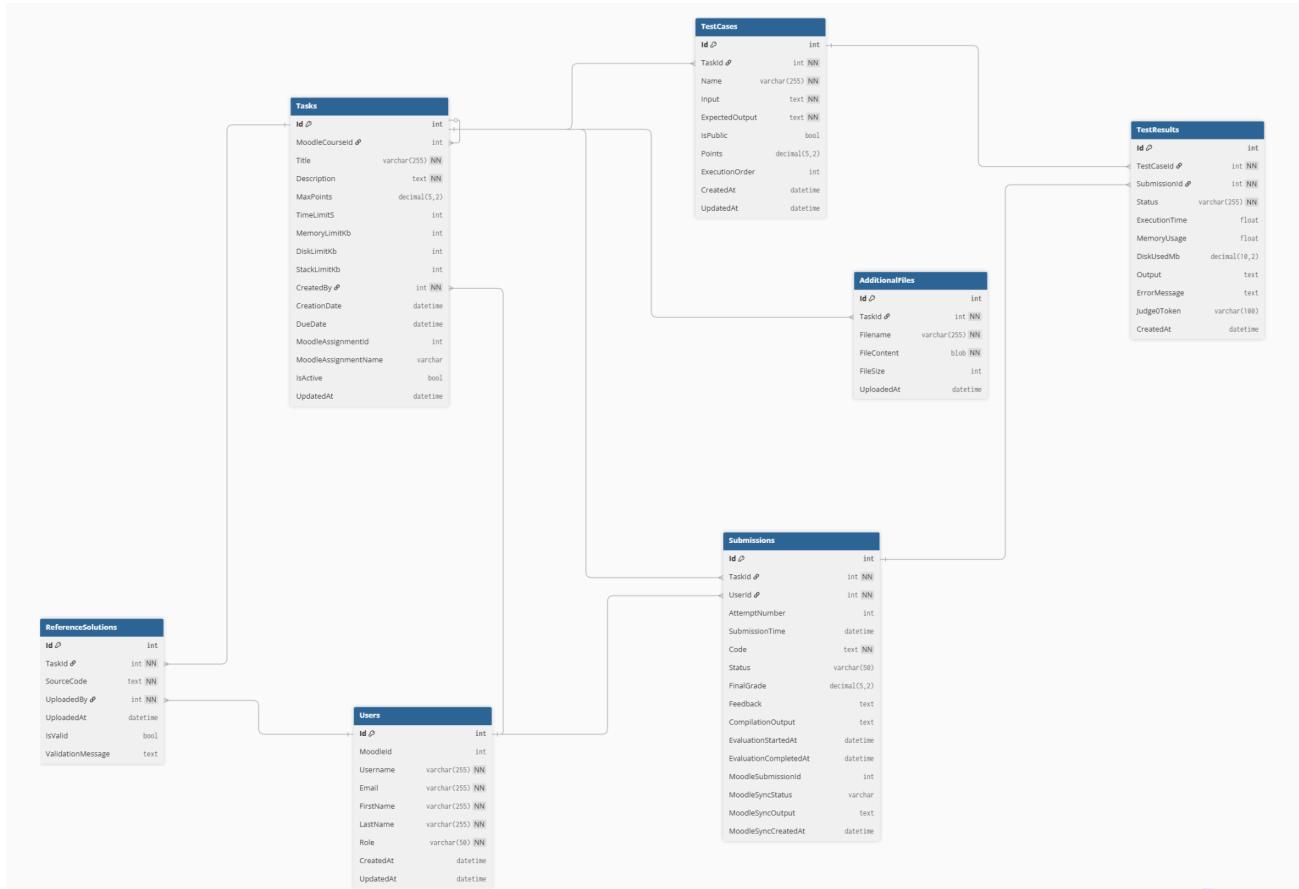
Потребител (User)

- Потребителско име
- Email адрес
- Роля в системата (студент, преподавател, администратор)
- Връзка с профила в Moodle

Други таблици: Еталонно решение (ReferenceSolution), Допълнителни файлове (AdditionalFile)

2.3.2. Диаграма на връзките

Забележка: Диаграмата по-долу показва основните връзки между информационните обекти в опростен вид. Подробна схема на релационната база данни с всички технически детайли (първични ключове, външни ключове, индекси, точни типове данни) се намира в архитектурния проект.



Основни връзки между данните:

- Един преподавател създава много задачи
- Една задача съдържа много тестови случаи
- Една задача получава много предавания от студенти
- Един студент предава много решения (за различни задачи)
- Едно предаване се тества срещу много тестови случаи
- Всеки тестов случай генерира отделен резултат за всяко предаване

3. ОБЩИ И СПЕЦИФИЧНИ ЦЕЛИ НА ПРОЕКТА

3.1. Обща цел

Трансформиране на процеса на обучение по програмиране чрез автоматизирана система за оценяване с незабавна, обективна и детайлна обратна връзка.

3.2. Специфични цели

За студентите:

- Незабавна обратна връзка (10-60 сек)
- Неограничени опити с история
- Ясни критерии и детайлна информация

- Достъпност 24/7

За преподавателите:

- Автоматизация (3-17 часа седмично спестени)
- Детайлна аналитика и статистика
- Гъвкавост при дефиниране на задачи
- Пълен контрол и достъп до решения

За процеса:

- Стандартизация и обективност
 - Повишена ангажираност
 - Скалируемост
 - По-високо качество на обучението
-

4. ТЕКУЩО СЪСТОЯНИЕ

4.1. Съществуващи решения

Moodle Assignment Module - Вграден, но без автоматично оценяване.

Judge0 CE - Sandbox API за 60+ езика, но без Moodle интеграция и UI.

VPL (Virtual Programming Lab) - Moodle plugin, но със сложна конфигурация и старяла технология.

Codeboard - Модерен интерфейс, но платено и зависимо от външни услуги.

4.2. Обосновка за нова система

Нито едно решение не отговаря напълно на нуждите - пълна интеграция с Moodle 3.8.9, гъвкавост, контрол върху данни, модерни технологии, дългосрочна поддръжка.

Сравнителна таблица на съществуващи решения:

За да илюстрираме предимствата на новата система, представяме сравнение със съществуващите решения:

Критерий	Moodle Assignm.	Judge0 CE	VPL	Codeboard	Нашето решение
Автоматично оценяване	X	✓	✓	✓	✓
Moodle интеграция	✓	X	✓	X	✓
Детайлна статистика и аналитика	X	X	~	~	✓
C# поддръжка	н/п	✓	✓	✓	✓
Sandbox изпълнение	н/п	✓	✓	✓	✓
Безплатно/Open-source	✓	✓	✓	X	✓
Контрол върху данни (on-premise)	н/п	✓	~	X	✓
Лесна конфигурация	✓	~	X	✓	✓
Модерни технологии	~	✓	X	✓	✓
Копиране на задачи между курсове	✓	н/п	~	X	✓

Легенда:

- ✓ = Пълна поддръжка
- ~ = Частична поддръжка или ограничена функционалност
- X = Няма поддръжка
- н/п = Неприложимо

Извод: Нито едно от съществуващите решения не предлага комбинацията от пълна Moodle интеграция, детайлна статистика, контрол върху данните и лесна конфигурация, която нашата система осигурява.

5. ФУНКЦИОНАЛНИ ИЗИСКВАНИЯ

5.1. Управление на задачи (преподаватели)

FR-1.1: Създаване на задача

- Въвеждане на име, описание, максимални точки
- Конфигуриране на ограничения (време, памет, диск)
- Свързване с Moodle Assignment
- **Вход:** Име, описание, точки, лимити, Moodle ID
- **Изход:** Идентификатор, потвърждение

FR-1.2: Добавяне на тестови случаи

- Множество тестове с име, вход/изход
- Задаване на тегло и видимост
- Допълнителни файлове (readonly)
- **Вход:** Име, входен/изходен файл, точки, видимост
- **Изход:** Списък тестове, опции за редакция

5.2. Предаване и оценяване (студенти)

FR-2.1: Автентикация

- Автоматична автентикация чрез Moodle session
- Показване на активни задачи

FR-2.2: Предаване през Moodle

- Текстово поле за C# код, предоставено от Moodle Plugin, който комуникира с backend API
- Валидация на формат
- **Вход:** C# изходен код
- **Изход:** Идентификатор, статус

FR-2.3: Автоматично оценяване (КРИТИЧЕН)

Процес на оценяване:

Стъпка 1: Компиляция в sandbox среда

- Програмата се компилира изолирано без достъп до системни ресурси
- При компилационна грешка процесът спира и се връща пълното съобщение от компилатора
- Компиляцията и изпълнението се извършват чрез Judge0 CE API, използвайки изолирана Docker среда с контрол на ресурси

Стъпка 2: Изпълнение на тестови случаи

- За всеки тестов случай програмата се стартира отново в нова изолирана среда
- Входните данни се подават през стандартен вход
- Програмата генерира изход през стандартен изход
- Изпълнението се следи за превишаване на лимити (време, памет, диск)

Стъпка 3: Сравнение на резултатите

- Изходът от програмата се сравнява byte-by-byte с очаквания изход
- **Точно съвпадение:** тестът е успешен (Pass)
- **Различие в изхода:** тестът е неуспешен (Fail)
- **Превищено време:** тестът е timeout (Time Limit Exceeded)
- **Превищена памет:** тестът е memory limit (Memory Limit Exceeded)
- **Превищено дисково пространство:** тестът е disk limit
- **Runtime грешка:** тестът е runtime error

Стъпка 4: Изчисляване на оценка

- Преброяване на успешни тестове (status = Accepted)
- **Формула:** (брой успешни тестове / общ брой тестове) × максимални точки
- Закръгляване до 2 десетични знака
- Пример: 7 от 10 теста минават, максимум 10 точки → $(7/10) \times 10 = 7.00$

Стъпка 5: Генериране на обратна връзка

- Създаване на текстово съобщение за студента (виж FR-2.4 за детайли)
- Включване на информация за всеки неуспешен тест
- При грешка - конкретно описание

Стъпка 6: Връщане на резултата в Moodle

- Оценката се записва автоматично в поле Grade на Assignment
- Обратната връзка се записва във Feedback поле
- Студентът вижда резултата веднага в Moodle

Вход: C# изходен код

Изход: Оценка (число от 0 до max_points) и обратна връзка (текст) в Moodle

FR-2.4: Обратна връзка

Типове съобщения:

- Компиляционна грешка → пълно съобщение от компилатора
- Неуспешен тест → "Неуспешно изпълнение на тест '<име>'"
- Timeout → "Програмата не преминава за достатъчно кратко време. Време: Xms, Лимит: Yms"
- Изчерпан диск → "Изчерпване на дисково пространство"
- Успех → "Поздравления! Всички тестове преминати! Оценка: X/Max"

FR-2.5: История на предавания

- Списък с дата и оценка
- Преглед на код и обратна връзка

FR-2.6: Примери за оценяване

Системата работи по следния начин при различни сценарии:

Пример 1: Програма с компилационна грешка

- Предаден код:

```
```csharp
Console.WriteLine("Hello; // липсва затваряща кавичка
```

```
- Резултат: Оценка **0.00**
- Обратна връзка:

```
```
```

Компиляционна грешка:  
error CS1010: Newline in constant  

```
```
```

Пример 2: Програма с частично верни резултати

- Задача: 5 тестови случая, максимум 10 точки
- Резултат: 3 теста минават (Pass), 2 не минават (Fail)
- Оценка: **(3/5) × 10 = 6.00** точки
- Обратна връзка:
``

Успешно преминати тестове: 3 от 5

Неуспешно изпълнение на тест 'Тест 3 - Големи числа' -
резултатът се различава от очакваното.

Неуспешно изпълнение на тест 'Тест 5 - Граничен случай' -
резултатът се различава от очакваното.

``

Пример 3: Програма с timeout (бездържавен цикъл)

- Предаден код съдържа бездържавен цикъл: `while(true) {}`
- Резултат: Оценка **0.00**
- Обратна връзка:
``

Програмата не преминава тестът 'Тест 1' за достатъчно кратко време.

Време за изпълнение: 5000ms

Максимално позволено време: 5000ms

``

Пример 4: Успешна програма (всички тестове минават)

- Код: Коректно решение, минава всички тестове
- Резултат: Оценка **10.00** (пълна оценка)
- Обратна връзка:
``

Поздравления! Всички тестове са преминати успешно!

Оценка: 10.00/10.00

``

Пример 5: Програма с изчерпване на дисково пространство

- Предаден код създава много голям файл
- Резултат: Оценка **0.00**
- Обратна връзка:
``

Програмата не преминава тестът 'Тест 2' поради изчерпване
на позволеното дисково пространство.

Използвано: 256MB

Максимално позволено: 256MB

``

Пример 6: Програма с runtime грешка

- Предаден код: Деление на нула `int x = 10 / 0;`
- Резултат: Оценка **0.00**
- Обратна връзка:
``

Runtime грешка в тест 'Тест 1':

System.DivideByZeroException: Attempted to divide by zero.

5.3. Статистика (преподаватели)

FR-3.1: Предавания на студент

- Всички опити с хронология
- Преглед на код и резултати

FR-3.2: Статистика за задача

Преподавателят вижда детайлна статистика за всяка задача, организирана в три категории:

A. Обща информация

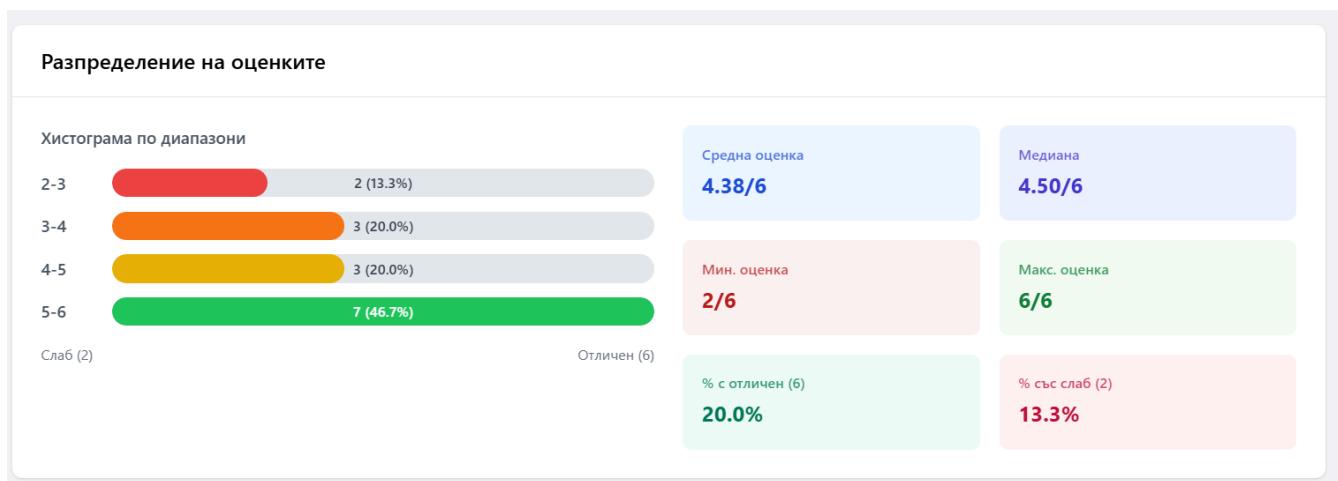
- Име на студента
- Потребителско име
- Имейл адрес
- Брой предавания за задачата
- Средна оценка (в проценти)
- Бутон "Виж детайли" за преглед на конкретните предавания

| ИМЕ | ПОТРЕБИТЕЛ | ИМЕЙЛ | ПРЕДАВАНИЯ | СРЕДНА ОЦЕНКА | ДЕЙСТВИЯ |
|------------------|------------|---------------------|------------|---------------|-----------------------------|
| SD Stu Dent | student4 | stu.dent4@test.test | 1 | 100% | Виж детайли |
| SD Stu Dent2 | student2 | stu.dent2@test.test | 2 | 100% | Виж детайли |
| SD Stu Dent3 | student3 | stu.dent3@test.test | 1 | 100% | Виж детайли |
| SD Stu Dent5 | student5 | stu.dent5@test.test | 2 | 100% | Виж детайли |
| SD Stu Dent6 | student6 | stu.dent6@test.test | 3 | 100% | Виж детайли |
| SD Stu Dent7 | student7 | stu.dent7@test.test | 6 | 100% | Виж детайли |
| SD Stu Dent8 | student8 | stu.dent8@test.test | 2 | 100% | Виж детайли |
| ST Student Three | student3 | student3@test.local | 21 | 23.33% | Виж детайли |

B. Разпределение на оценките

- Хистограма с брой студенти по оценки (2-3, 3-4, 4-5, 5-6)
- Средна оценка за задачата
- Медиана на оценките
- Минимална оценка
- Максимална оценка
- Процент студенти с отличен (6)
- Процент студенти със слаб (2)

Пример за хистограма на оценки:



5.4. Интеграция с Moodle

FR-4.1: Синхронизация на потребители

- Автоматично извличане на студенти от курс
- Присвояване на роли

FR-4.2: Двупосочна синхронизация на оценки (КРИТИЧЕН)

- Автоматично обновяване на оценка при предаване
- Синхронизация на обратна връзка

FR-4.3: Moodle Plugin

- Нов тип Assignment "Code Evaluation"
- Конфигурация от Moodle интерфейс

5.5. Сигурност и sandbox

FR-5.1: Изолирано изпълнение (КРИТИЧЕН)

- Docker-базирана sandbox среда, предоставена от Judge0 CE, с вградени ограничения за време, памет и диск
- Без достъп до файлова система извън работната директория
- Автоматично спиране при изчерпване на ресурси

FR-5.2: Ограничения за ресурси

- Време (default: 5000ms)
- Памет (default: 256MB)

6. ТЕХНИЧЕСКИ ИЗИСКВАНИЯ

6.1. Архитектура

Модулна структура:

- Web Interface (JavaScript/HTML/CSS)
- Moodle Plugin (PHP)
- Backend API (REST - .Net/C#)
- Code Execution (Judge0 CE)
- Database (PostgreSQL)

Скалируемост: Хоризонтално скалиране, асинхронна обработка (queue)

6.2. Инфраструктура

Хардуер:

- Web Server: 4 cores, 8GB RAM, 100GB SSD
- **Judge0 CE (локална инсталация)** – използва се чрез REST API, хостван на Application Server
- Database: 4 cores, 8GB RAM, 500GB SSD

Софтуер:

- Docker & Docker Compose
- Web Server: Nginx/Apache
- PHP 8.3+, PostgreSQL 12+
- Judge0 CE

Мрежа:

- HTTPS (SSL)
- Firewall с зони: Public (Web), Internal (API), Isolated (Judge0)

6.3. Интеграция

Moodle API:

- mod_assign_get_assignments, mod_assign_save_grades
- core_user_get_users, core_course_get_courses

Judge0 API:

- POST /submissions, GET /submissions/:token
- Language ID за C#: 51

6.4. Потребителски интерфейс

- Responsive design, интуитивна навигация
- Следване на Moodle UI conventions

6.5. Производителност

- Зареждане на страници в Moodle: < 5 секунди
- Компилация на C# код: 3-5 секунди (включва Moodle → Backend → Judge0)
- Пълно оценяване: < 2 минути за 10 теста (до 5 сек/тест + overhead)
- Капацитет: 300-500 студенти, 50-100 активни задачи едновременно
- Peak load: Системата трябва да издържа на 50 едновременни предавания без забележимо забавяне

6.6. Сигурност

- SSO чрез Moodle
- SQL injection/XSS/CSRF protection

- Изолирани Docker containers без мрежа