IZMIR UNIVERSITY OF ECONOMICS
FACULTY OF ENGINEERING

CE316 Design Document Team 10

OmnIDE

İZMİR EKONOMİ ÜNİVERSİTESİ

2024-2025 SPRING SEMESTER

EDİZ ARKIN KOBAK/20220602054 /SECTION-2 /CE.

BEYAZIT TUR/20220602074 /SECTION-2 /CE.

MERT KOĞUŞ/20210602037 /SECTION-2 /CE.

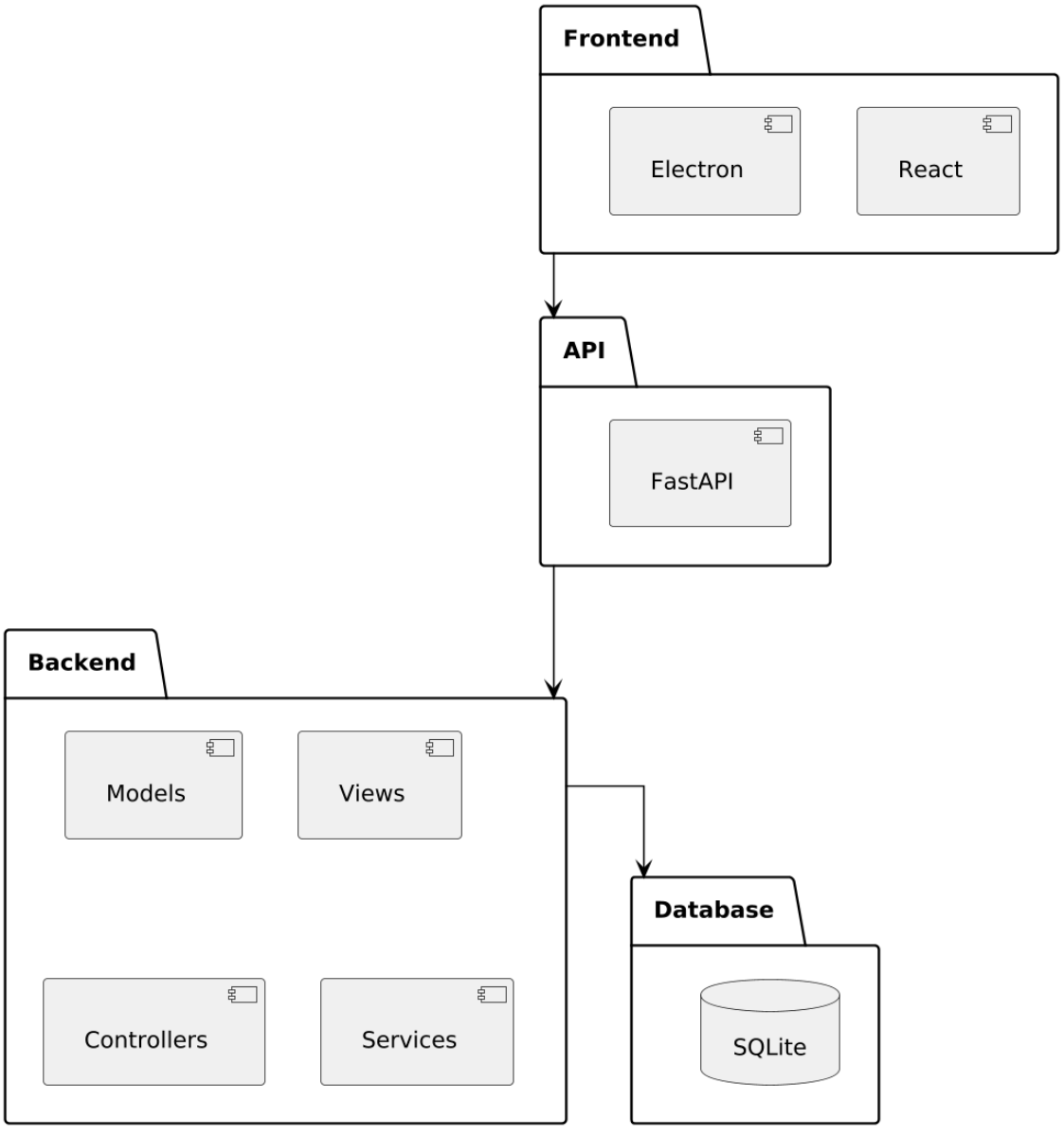
TABLE OF CONTENTS

1. Introduction

The Integrated Assignment Environment (IAE) is a standalone desktop application designed to efficiently manage and evaluate programming assignments across multiple languages. The system provides functionality for lecturers to create assignments, configure language-specific environments, process student submissions, and automatically evaluate code correctness. It ensures robust handling of different programming languages, supports automated testing with custom inputs, and maintains a user-friendly interface. The system will be developed using Python for the backend (handling file operations, zip operations, and data analysis) and React with TypeScript, Electron, Tailwind CSS, and Vite for the frontend and desktop application framework. SQLite will be used for local data storage.

2. Software Architecture

2.1 High-level Architecture Diagram



2.2 Architecture Design

The system is designed with a modular architecture that separates concerns into distinct layers, ensuring flexibility, scalability, and maintainability. The main components of the architecture include frontend components, backend services, file operations, and persistent storage. Each module is responsible for a specific task, with well-defined interfaces for communication between them.

2.3 Technologies and Tools Used

Frontend:

- React: Used for creating dynamic, component-based user interfaces
- TypeScript: Type-safe JavaScript for better code maintainability
- Material UI and Tailwind CSS: For responsive and modern UI components
- Electron: Framework for creating cross-platform desktop applications
- Vite: Modern frontend build tool for React

Backend:

- Python 3.9+: Core programming language for backend logic
- FastAPI: Modern, high-performance web framework for building APIs
- SQLAlchemy: SQL toolkit and Object-Relational Mapping (ORM) library
- Pydantic: Data validation and settings management
- Uvicorn: ASGI server implementation for running FastAPI

File Operations:

- Python's built-in zipfile: For compressing and extracting student submissions
- subprocess: For executing code in various programming languages
- pandas: For data processing and analysis

Database:

- SQLite: Lightweight file-based database used for local storage

Development Tools:

- Git: For version control
- ESLint: For JavaScript/TypeScript code quality
- Pytest: For testing Python components

Installer Packaging:

- Electron Builder: For packaging the application into installable formats

3. Structural Design

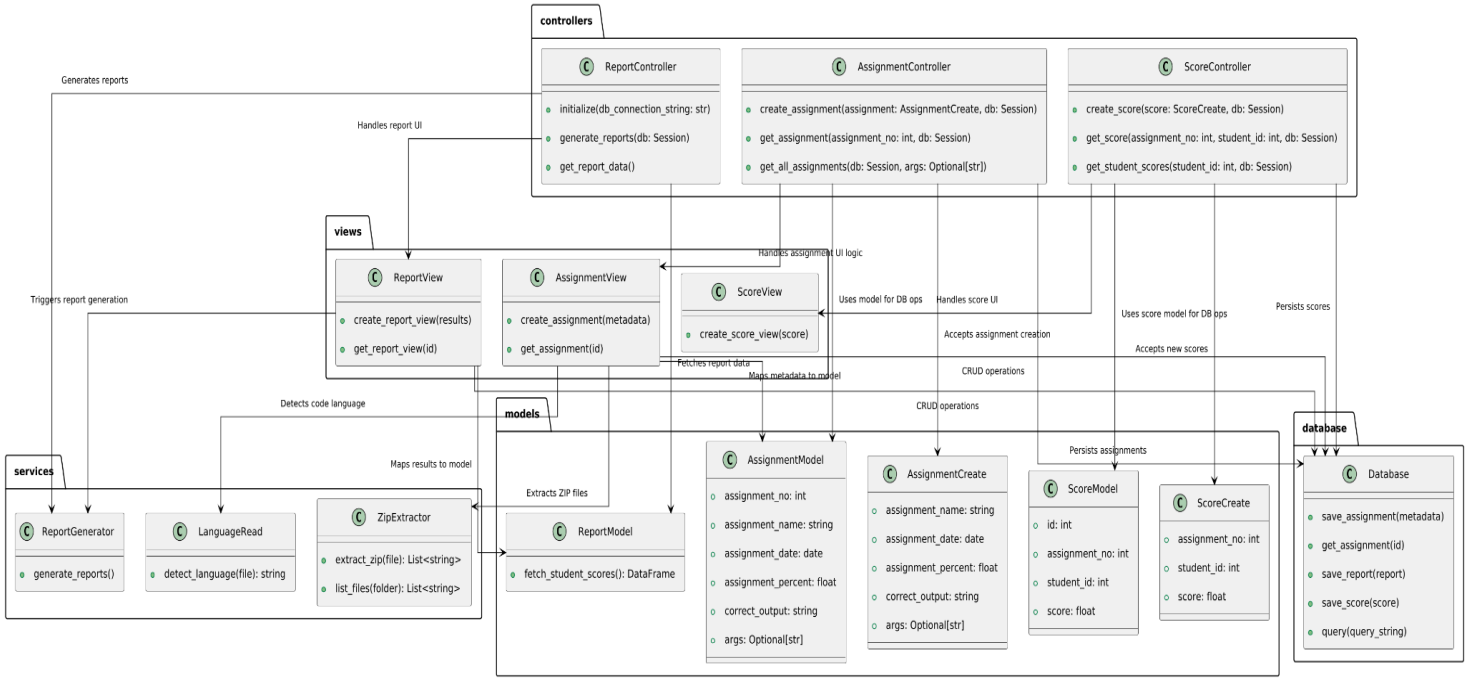
The structural design includes the primary classes that represent projects, language configurations, student submissions, evaluation results, and various system controllers. This section outlines these key components based on the functional requirements, explains the design principles followed, and details the interaction between system classes as depicted in the class diagram..

3.1 Design Principles

The system follows key design principles to ensure modularity and maintainability. The **Single Responsibility Principle** ensures each module and controller handles a specific function, minimizing complexity. **Separation of Concerns** is maintained by clearly dividing the UI, business logic, and data access layers, making the system easier to manage and scale. The backend follows the **Model-View-Controller (MVC)** pattern, which separates data, logic, and presentation for better organization and maintainability.

On the frontend, a **Component-Based Architecture** is used with React, promoting reusable, independent components for a flexible and consistent UI. Communication between the frontend and backend occurs via **RESTful API endpoints**, ensuring efficient interaction. These principles ensure the system is scalable, organized, and easy to maintain, adhering to best practices in modern web development.

3.2 System Components and Class Diagram



The architecture consists of several main components:

- **AssignmentController:** Manages assignment-related logic, including creation and retrieval of assignments.
- **ReportController:** Handles the generation and retrieval of reports.
- **ScoreController:** Manages student score operations, including creation and retrieval.
- **AssignmentView:** Displays the UI for assignments and handles user interactions.
- **ReportView:** Displays the UI for reports and triggers report generation.
- **ScoreView:** Displays the UI for student scores.
- **ZipExtractor:** Extracts and lists files from ZIP archives.
- **ReportGenerator:** Generates reports based on student data and execution results.
- **LanguageRead:** Detects the programming language of student code, compiles and runs the code.
- **AssignmentModel:** Represents assignment data and stores it in the database.
- **AssignmentCreate:** Stores metadata for creating new assignments.
- **ReportModel:** Fetches and processes data for report generation.
- **ScoreModel:** Represents student score data and stores it in the database.
- **ScoreCreate:** Stores score information for new student scores.
- **Database:** Manages all database operations, including saving and querying data.

3.3 Database Design

Data persistence is handled by the DatabaseManager class using SQLite. The database stores:

- Project metadata (Metadata)
- Evaluation results (ExecutionResult)
- Generated reports (Report)
- Configuration files and internal records for versioning and updates

Each entity is stored in its dedicated table with appropriate foreign key relationships to maintain integrity and traceability across the system.

3.4 File Handling

3.4.1 ZIP File Handling

The system will process student submissions in ZIP format. A dedicated service will handle the extraction of ZIP archives, ensuring that the file structure is verified and meets the necessary requirements. The extracted files will be organized based on unique student identifiers, making them easy to track. These files will then be prepared for compilation and execution, with the appropriate organization for further processing.

3.4.2 Configuration Files (Import-Export)

The system will support the import and export of configurations using structured file formats. For exporting, the system will convert database configuration entries into predefined structured formats, preserving critical configuration details such as compiler paths, arguments, and language-specific settings. When importing, users can upload these structured configuration files, and the system will parse them to update the database configurations, ensuring smooth integration and consistency with the system's configuration requirements.

3.4.3 CSV Exporting Analysis

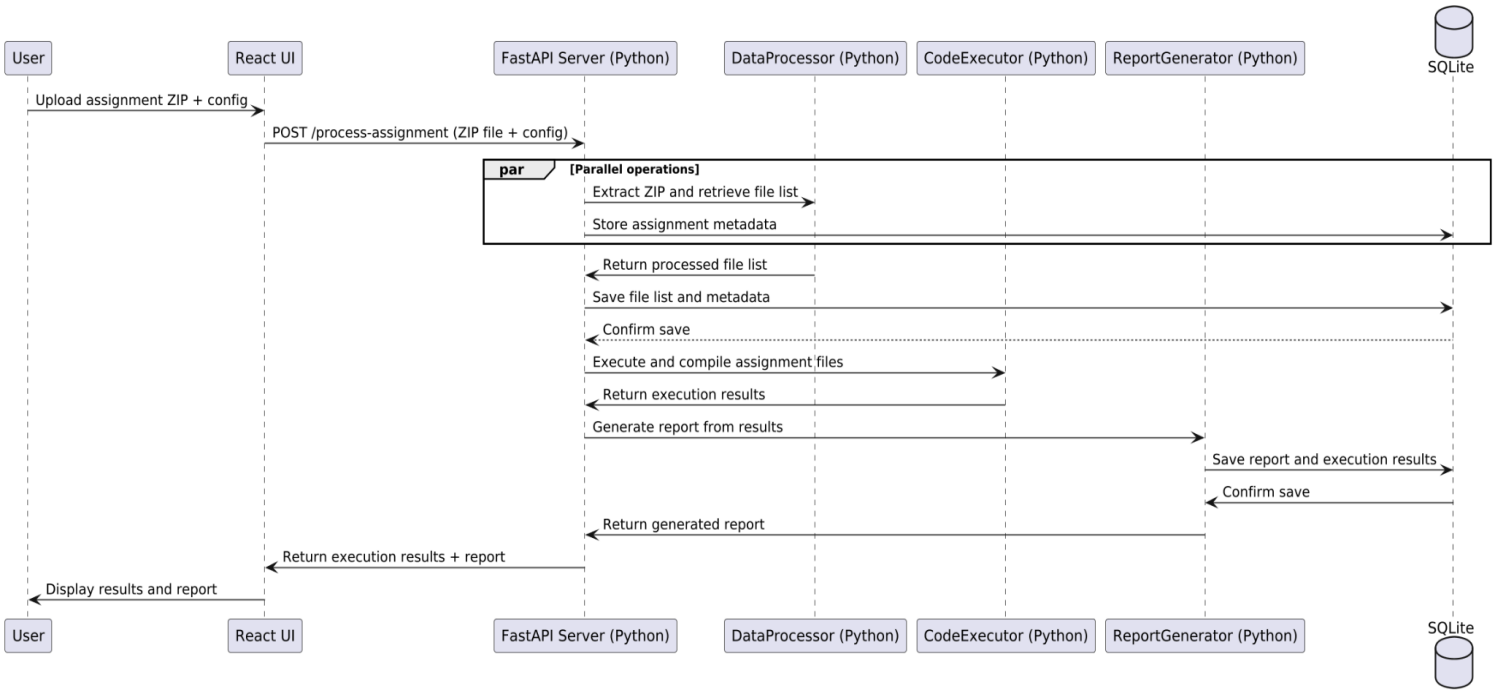
To support data analysis and external reporting, the system will allow the export of evaluation results in CSV format. The exported CSV files will include essential details such as student ID, assignment ID, score, execution time, and submission status. The system will leverage Python's pandas library to structure and process the exported data, enabling further analysis. Additionally, the system will support exporting aggregated statistics, providing valuable insights into student performance and overall assignment trends.

---

4. Behavioral Design

While the structural design includes the primary classes that represent projects, configurations, and submissions, the following methods are required to add functionality to the software.

4.1 Sequence Diagram



4.2 Creating Projects

In OmnIDE, projects represent assignments, encompassing both their configurations and associated student submissions. The process of creating a project follows these steps:

1. The user enters project details through a form in the React frontend.
2. The form data is sent as a JSON object to the /api/assignments endpoint in the FastAPI backend.

3. The AssignmentController validates the received data and creates a new entry in the database to store the project details.
4. Upon successful creation, the frontend redirects the user to the project configuration page for further customization.
5. The project settings are saved and can be modified at any time through the configuration page.

This process ensures that projects are created, configured, and stored efficiently, with the flexibility to adjust settings later as needed.

This functionality will meet **Requirement 3.**

Requirement 3 (user requirement): The user must be able to create a project that uses an existing or a new configuration.

4.3 Creating, Editing and Removing Configurations

The system shall provide full support for users to create, edit, and remove configurations related to file compilation and interpretation processes. These configurations primarily include the compiler or interpreter paths associated with specific file extensions (e.g., .py, .java, .cpp). Configuration management will facilitate seamless execution of various file types by automatically resolving the execution path based on previously defined settings.

When users interact with the UI, they can select or input a custom compiler or interpreter path corresponding to a specific file type. This configuration data is transmitted to the backend through a REST API. The ScoreController receives the request and delegates the task to the appropriate service layer for processing.

The configuration is stored and managed in an SQLite database under a table named config, where each file extension (e.g., .py) is mapped to its corresponding compiler or interpreter path (e.g., /usr/bin/python3).

During file compilation or execution, the system automatically retrieves the relevant configuration from the database and uses it to run the file via Python's subprocess.run() method.

If no configuration is found for a given file type, the system will prompt the user to define a new one, which will then be saved in the database for future use. This ensures efficient execution while minimizing redundant user input.

The system shall also allow users to export all their existing configurations into a single JSON file. This file will contain all defined compiler or interpreter paths as key-value pairs, where the key represents the file extension (e.g., ".py") and the value is the corresponding path. Additionally, the system shall support importing configurations from a JSON file in the same format. During the import process, the system will parse the JSON file and store the provided paths in the database.

This functionality will be fully implemented in Python, utilizing the built-in json library for handling JSON data and sqlite3 for database operations.

Requirement 4 (user requirement): The user must be able to create, edit and remove a configuration.
Requirement 5 (user requirement): The user must be able to import and export configurations.

4.4 Processing Student Submissions

The user inputs the folder path containing student submissions for an assignment through the frontend user interface. This path is transmitted to the backend via a **FastAPI-based REST API**, where it is processed. Upon receiving the path, the backend utilizes Python's built-in **zipfile** module to automatically extract ZIP files into a designated directory, organized under the assignment's name within the application's storage structure. Extracted files are renamed or saved based on student identification numbers, facilitating systematic evaluation.

The system employs the **subprocess** module in Python to execute and evaluate code submissions across multiple programming languages, including Java, C#, C, and Python. The **subprocess.run()** function enables the backend to invoke external compilers and interpreters, capturing both output and error messages for each execution. For instance, Java code is compiled using the javac compiler and executed with the java interpreter; C# code is compiled with the csc compiler; and C code is compiled with gcc before execution. Python scripts are directly executed using the Python interpreter.

Requirement 6 (user requirement): The software must be able to process ZIP files for each student.
Requirement 7 (user requirement): The software must be able to compile or interpret source code using the configuration of the project.

4.5 Comparing Outputs

The system integrates a FastAPI-based REST API, Python, and React to provide a comprehensive platform for code evaluation and analytics. Student submissions are uploaded through the frontend and processed by the backend, where ZIP files are extracted using Python's built-in zipfile module. Based on the assignment settings, the backend identifies the programming language for each submission and uses Python's subprocess module to compile or execute the code. Standard output and error streams are captured during execution for further analysis.

Output comparison is performed by executing the student code with predefined test inputs and comparing the captured output against the expected output. Various comparison methods are supported, including exact matching, whitespace-insensitive matching, and pattern matching where appropriate. A similarity score is generated to quantify the degree of match, and a pass/fail status is assigned based on the results. The evaluation results are stored in a lightweight SQLite database for historical tracking, reporting, and visualization.

The React frontend communicates with the FastAPI backend via RESTful API calls to retrieve evaluation results and present them to users. Using libraries like React Table and Chart.js, the system displays detailed feedback and performance insights in an interactive format. Over time, advanced analytics and predictive modeling can be applied using tools such as pandas and scikit-learn to identify student learning patterns and forecast future performance.

Requirement 8 (user requirement): The software must be able to compare the output of the student program and the expected output.

4.6 Project Load and Save Behavior

The system allows users to save, open, export, and import projects with ease. When a project is created or modified, all configurations, such as evaluation settings and testing criteria, are

stored in the SQLite database, ensuring data consistency. Users can open existing projects from the database, retrieving the stored configurations and evaluation results.

In addition, the system supports exporting project data in portable formats for backup or sharing purposes. It also enables importing project data from external sources, offering flexibility for collaboration or migration between different environments. These functionalities are accessible through the FastAPI backend and the React frontend for a seamless user experience.

Requirement 10 (user requirement): The user must be able to open and save projects to operate on them at any time.

---

5. Graphical User Interface

The user interface (UI) of the system is implemented using **React**, coupled with **Material UI** and **Tailwind CSS** for responsive and modern design. The UI is hosted within an **Electron container**, enabling the application to function as a cross-platform desktop application. This ensures compatibility across different operating systems while maintaining a native desktop experience.

Menu Bar:

- File
    - New Project
    - Open Project
    - Save Project
    - Exit
- Configuration
    - New Configuration
    - Manage Configurations
    - Import/Export
- Evaluation
    - Process Submissions
    - View Results
- Help
    - User Manual
    - About

Main Views:

1. **Project Dashboard**: This view lists all projects with their respective statuses, providing quick insights into each project's current state.

2. **Configuration Editor**: A dedicated interface for managing and modifying language-specific configurations, such as compiler settings, arguments, and evaluation parameters.

3. **Submission Processing**: A section where users can upload student submissions and initiate the processing of these files.

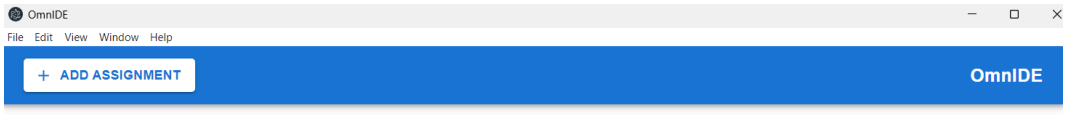
4. **Results Viewer**: This view allows users to view detailed evaluation results, including the status and performance of each student submission.

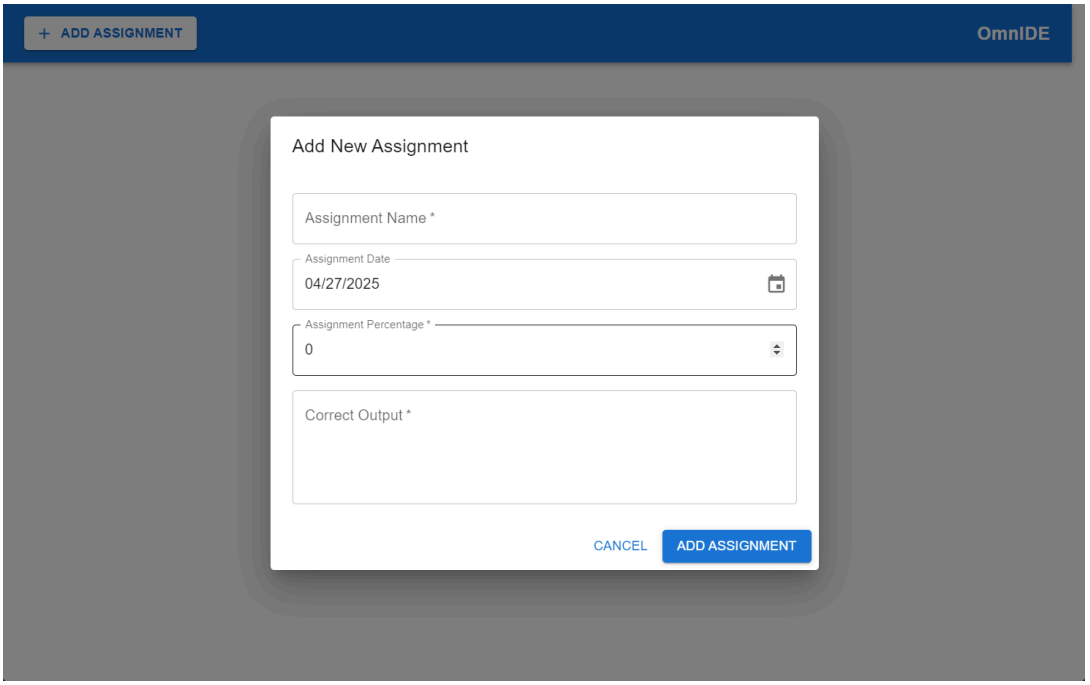
The frontend, built with **React.js**, displays student results in an interactive and user-friendly interface. It fetches processed data, such as submission status, correctness, and execution feedback, through asynchronous API calls to a **FastAPI backend**, which interacts with a lightweight file-based database like **SQLite** or **LiteDB**. The backend leverages Python to handle the execution and evaluation of student code, while the frontend uses the **Material-UI (MUI)** library for consistent design and responsive layout. For enhanced data visualization, **React Table** and **Chart.js** are employed, enabling instructors to efficiently track and interpret individual or batch student performance.

Requirement 9 (user requirement): The software must be able to display the results of each student file.

5.1 Sample User Interfaces

These screenshots from the program will meet the following Functional Requirements.





5.2 Help Manual

The system includes a user manual accessible via a "Help" menu in the frontend, built with React. This manual provides clear instructions on how to use the platform, including submitting code, viewing results, and managing projects. The frontend will display the manual in a user-friendly, searchable format, with easy navigation between sections.

Content for the manual can be stored as static files or fetched from the backend, allowing for easy updates. The backend may also offer features like context-sensitive help or user-specific topics for a more personalized experience.

Requirement 2 (system requirement): The software must have a manual that will be displayed with a “Help” menu item.

6. Installer and System Requirements

The application will be packaged as a Windows desktop application using **Electron Builder**, with build settings defined in the package.json. During the build process, all necessary frontend and backend components are bundled into a single installer, with output placed in the dist_electron directory.

The installation process is straightforward: users download the Windows installer, run the setup, and the application is installed with all required files. External dependencies such as Python interpreters and language compilers must be installed separately by the user.

Requirement 1 (system requirement): The software must be deployed to target Windows computers with an installer

---