Project for **Advanced Topics of Software Engineering**
Winter semester 2020/21
Prof. Dr. Alexander Pretschner, Claudius Jordan, Daniel Elsner

Technische Universität München
Fakultät für Informatik

Project Specification – OnlineIDE

Motivation

An integrated development environment (IDE) provides programmers typically with at least a source code editor, project file management, and capabilities to compile and/or interpret source code. Most modern IDEs furthermore have build automation tools, a debugger, and version control systems integrated. IDEs can be language-specific or support multiple languages. Intentions behind using an IDE can be for example:

- Reduction of the necessary configuration, e.g. link and install multiple development utilities, by using one cohesive unit
- Graphical user interface (GUI) which can simplify and sometimes accelerate the development process
- Tighter integration of code with productivity tools, e.g. continuous parsing of code providing instant feedback or GUI builders

However, when using an IDE you will often still need certain compilers or interpreters installed to work with it properly, and you may need to update your IDE regularly in order to stay up-to-date. Therefore, it can be desirable to even reduce this effort with a **web-based IDE**, where necessary tools, programming languages or features come prepackaged and always at the newest version.

OnlineIDE

You and your team will implement your own web-based IDE (**OnlineIDE**), where users can write source code in a web application, organize source code files, and compile the code and display output (e.g. error messages) from the compiler operated on a remote server (see big picture in Fig. 1). The users do not need to install any programs or files nor configure their development machines to start new projects. Find some inspiration at https://ide.cs50.io/, https://aws.amazon.com/cloud9/, https://www.gitpod.io/ or https://theia-ide.org/.

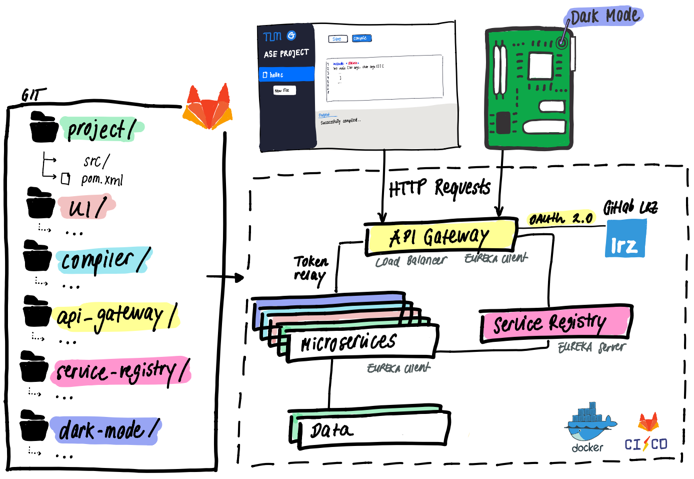

Figure 1: Big picture of the semester project, the OnlineIDE

Requirements & Grading

Requirements

- Source Code Editor (*UI* service) – **2 points**
  - A user can see syntax highlighting for the respective programming language when editing a source file in the code editor.
  - A user can compile single source code files and see the output of the compilation process (e.g. errors).
  - When a user edits a source file, he or she needs to save the changes first before being able to compile the code.
  - A user can create, list, update and delete (source code) projects through the web GUI.
  - A user can create, list, update and delete source code files inside a project through the web GUI.
  - A user can share a project with another user.
- Compilation (*compiler* service) – **1 point**
  - A user can compile single source code files containing C or Java code (both, C and Java must be implemented).
  - A user can retrieve the standard output and errors of the compilation process.
- Project File Management (*project* service) – **1 point**
  - A user can create, list, update and delete (source code) projects which have a unique name and are persisted in a database.
  - A user can create, list, update and delete source code files inside a project; the source code files are persisted in a database (you do not have to store the files to disk, but simply store their content as a string in the database).
- Authentication/Authorization (*api-gateway* service) – **2 points**
  - Users can authenticate themselves via their GitLab LRZ account (OAuth 2.0).
  - User requests accessing projects, project files, or user information need to be authenticated and authorized.
  - Users can share projects with other users registered with GitLab LRZ thereby granting them full read/write permission.
- Microservices Architecture (*api-gateway* and *service-registry* service) – **1 point**
  - All services need to be independently executable and deployable microservices written with Spring Boot.
  - An API gateway serves as a single entry point for all clients and proxies/routes requests to the appropriate services, if authorized.
  - Service discovery is performed by a central service registry entity.
  - Load balancing is performed to distribute load across microservice instances.
- Dark Mode Toggle Button (*dark-mode* service) – **1 point**
  - A user can use a *dark mode* button in the form of a hardware switch on the microcontroller, to toggle a dark color scheme for the syntax highlighting.
  - When tapping the hardware switch, an interrupt is triggered which sends an HTTP request to the *dark-mode* service, which stores the dark mode status in memory (no need for other persistence).

  - For the sake of simplicity, the dark mode is not triggered for a specific user account, but rather for the entire OnlineIDE.
  - The dark mode status is regularly fetched from the OnlineIDE's frontend; there is *no need* to implement bidirectional streaming (e.g., through websockets).
  - The final version of your C code is flashed to the microcontroller before returning it to us. This includes the static IP address of your Google VM, as this is where the HTTP request is sent.

- Continuous Integration & Deployment – **2 points**
  - Regression tests are continuously run in a continuous integration environment on each commit (at least one unit, integration, and end-to-end test).
  - Each microservice lives in its own (Docker) container, to simplify scaling of specific services.
  - Containers of microservices and databases can be orchestrated and scaled independently.
  - The whole OnlineIDE microservice orchestration is continuously deployed to a Google VM and accessible via the browser.

Generally, we will guide you through all these requirements in the weekly project exercises. If you are unsure what some of these requirements mean in practice, please inspect our reference implementation of the OnlineIDE: https://ide.sse.in.tum.de/

Grading Scheme

The requirements are structured by the components (which can often be directly mapped to a microservice) and give the amount of points indicated behind them (e.g., 2 points for *Source Code Editor*). Importantly, these points are only given if **all** requirements for that component are fulfilled. That means for each component you get either full points or none. Overall, this leads to a total of 10 possible points. The grading scheme is as follows:

- If you reach at least 6 points, your final grade will improve by one grade step (0.3/0.4, e.g., 1.7 to 1.3).
- If you reach at least 8 points, your final grade will improve by two grade steps (0.6/0.7, e.g., 2.3 to 1.7).
- If you reach 10 points, your final grade will improve by three grade steps (1.0, e.g., 2.3 to 1.3).
- You cannot get a better final grade than 1.0.
- The bonus is only applicable if you pass the final or retake exam.
- We will check the contribution of each team member individually during the final technical demo (see below). If the contribution is questionable or differs significantly across team members, we will adjust the bonus for each individual (i.e., some team members might get less bonus than others).

If we cannot build (i.e., compile) or execute your project, we will not further consider it.

Cheating

We will manually review every team's code and detect any attempt of cheating or copying code. If we find any code parts to be copied from other project teams, **both** project teams will be disqualified for the bonus.

Deliverables

- Project git repository containing all source code and documentation necessary to run and deploy the OnlineIDE
- Short document or slide (PDF) containing information about team members and their individual contributions (not more than 1-2 pages)
- Short technical demo including the deployed OnlineIDE and short code walk-through (no slides)

Organization

- You should form groups of 3-5 students that work together.
- There will be a submission deadline announced towards the end of the semester. Project teams can work on their repositories only until then, everything that is added afterwards will not be considered.

Technology Roadmap / Project Schedule

In addition to the practical exercises, there are project exercises on each exercise sheet. The list below contains all technologies covered in the latter. Code samples can be found in the public repository of the course: https://gitlab.lrz.de/i4/teaching_public/winter-semester-2020/advanced-topics-of-software-engineering-2020.

- **Exercise Week 1**
  - Git and GitLab, Team up
- **Exercise Week 2**
  - Dark mode toggle button on a microcontroller
- **Exercise Week 3**
  - Spring Boot introduction
- **Exercise Week 4**
  - Persisting data with Spring Boot, Spring Data, PostgreSQL and Java Persistence API
- **Exercise Week 5**
  - Code compilation REST API with Spring Boot and Spring MVC
- **Exercise Week 6**
  - Advanced Spring Boot configuration: Spring Environment, Spring Profiles
- **Exercise Week 7**
  - Serving a packaged Angular Application with Angular CLI and Spring Boot
- **Exercise Week 8**
  - Testing the compilation REST API with Spring Boot (unit, integration, and end-to-end) and setting up GitLab CI for Git repository
- **Exercise Week 9**
  - Spring Security for authentication and authorization – Securing a REST API with CSRF protection and session cookies
- **Exercise Week 10**
  - Using OAuth 2.0 with Spring Security to secure an Angular Application
- **Exercise Week 11**
  - GitLab API for sharing projects with OAuth 2.0
- **Exercise Week 12**
  - API Gateway (Netflix Zuul) and Service Discovery (Netflix Eureka) pattern with Spring Cloud microservices
- **Exercise Week 13**
  - Microservice deployment with Container-per-Service pattern with Docker, Docker Compose, GitLab CI and Google Cloud Platform (GCP)