# Software Engineering

**WS 2024/25, Assignment 02**

Prof. Dr. Sven Apel
Lukas Abelt
Sebastian Böhm

**Handout:** 18.11.2024
**Handin:** 02.12.2024 23:59 CET

## Organizational Section:

- The assignment must be accomplished by yourself. You are not allowed to collaborate with anyone. Plagiarism leads to failing the assignment.
- The use of generative AI tool for solving the assignment is not allowed and will be counted as plagiarism.
- The deadline for the submission is fixed. A late submission leads to a desk reject of the assignment.
- We provide a project skeleton that must be used for the assignment. The skeleton can be downloaded from the CMS.
- The submission must consist of a *ZIP* archive containing only the project folder (i.e., the folder included in the provided skeleton) Any violation of the submission format rules leads to a desk reject of the assignment.
- Questions regarding the assignment can be asked in the forum or via email. Please do not share any parts that are specific to your solution in the forum, as we will have to count that as attempted plagiarism.
- If you encounter any technical issues, inform us immediately.

## Task 1 [30 Points]

Your task is to implement a simple configurable chat application in Kotlin[1] with Gradle[2] as a build system using different implementation techniques. We provide an implementation using C-preprocessor directives which acts as a template for the other implementations. The logic for all features is already present in this implementation, so you can focus on the different implementation techniques rather than on the logic. If you are unfamiliar with the language we recommend taking the *tour of Kotlin*[3].

a) Implement the configurable chat application using **runtime parameters** in the directory `runtime`. We provide the interfaces for the client, server, and message classes as well as a configuration class that gets passed to the constructor of the client/server classes. Your implementation must change its behavior based on the values in the configuration object. [10 Points]

b) Implement the configurable chat application using the **decorator pattern** in the directory `decorator`. We provide the component interfaces as well as tests that specify how the concrete component and decorator classes must be named and how they can be combined. [10 Points]

c) Implement the configurable database using the **build system** for configuration in the directory `buildsystem`. Here, the chat application is configured by setting the appropriate values in the file `gradle.properties` and then building the project using Gradle's `build` task. The compiled project must behave according to the selected configuration. To achieve this, you must modify the build script `build.gradle.kts` such that it uses appropriate source files during compilation (see sheet 05 task 2). [10 Points]

[1] `https://kotlinlang.org/`
[2] `https://gradle.org/`
[3] `https://kotlinlang.org/docs/kotlin-tour-welcome.html`

**Grading** Your submission will be graded based on the following criteria:

- Your submission must be in the correct format, i.e., a *ZIP* archive with the same layout as the project skeleton (results in 0 points if violated).
- Each subtask must compile, i.e., the command `gradle build` must succeed (results in 0 points for the subtask if violated).
- Each subtask must be implemented using the specified implementation technique (results in 0 points for the subtask if violated).
- We run unit tests (the ones provided + additional tests) against your submission. Points will be awarded based on passed tests.

**Project Skeleton** You must implement your solution based on the provided project skeleton. It consists of a file `ChatApp_Preprocessor.kt` that contains an implementation of the configurable chat application using C-preprocessor directives as well as three directories, one for each subtask. The directories for the subtasks have the following structure:

```
runtime
├── gradle/
├── src
│   ├── main
│   │   └── kotlin
│   │       ├── ChatApp.kt
│   │       └── util/
│   └── tests
│       └── kotlin
│           └── ChatAppTest.kt
├── build.gradle.kts
├── gradlew
├── gradlew.bat
└── settings.gradle.kts
```

(a) Runtime parameters

```
decorator
├── gradle/
├── src
│   ├── main
│   │   └── kotlin
│   │       ├── ChatApp.kt
│   │       └── util/
│   └── tests
│       └── kotlin
│           └── ChatAppTest.kt
├── build.gradle.kts
├── gradlew
├── gradlew.bat
└── settings.gradle.kts
```

(b) Decorator pattern

```
buildsystem
├── gradle/
├── src
│   ├── main
│   │   └── kotlin
│   │       ├── base
│   │       │   └── util/
│   │       └── default
│   │           └── ChatApp.kt
│   └── tests
│       └── kotlin
│           └── */ChatAppTest.kt
├── build.gradle.kts
├── gradle.properties
├── gradlew
├── gradlew.bat
└── settings.gradle.kts
```

(c) Build system

The skeleton for each subtask already contains some interfaces and definitions. **You must not modify these**, as otherwise, your submission might not work with our tests. This may lead to getting 0 points for the respective subtask if violated. You are, however, allowed to *extend* interfaces with your own definitions. To be sure, always check whether your submission works with the provided test cases.

The `util` package contains some classes that simulate network communication. It is the same for each subtask and must not be modified. We use this simulated network instead of "real" client-server communication to simplify the implementation and testing.

The file `build.gradle.kts` contains a build script that we use to build your submission and run the tests.[4] In general, you are not allowed to modify the build script. The only exception is task c), where you have to modify the main source set depending on the configuration. The build script should also enable you to import the project skeleton into any IDE with KOTLIN support (e.g., INTELLIJ). For the assignment, we use *KOTLIN 2.0.21* which is also specified in the build script. You can also run the tests included with the project skeleton using the build script. To run all tests execute the command `gradle test`.

The files included with the skeleton also contain some documentation that gives further information regarding how and where to implement your solution.

**Feature Model** The configurable chat application we implement follows the following feature model:

[4] `https://docs.gradle.org/current/userguide/userguide.html`

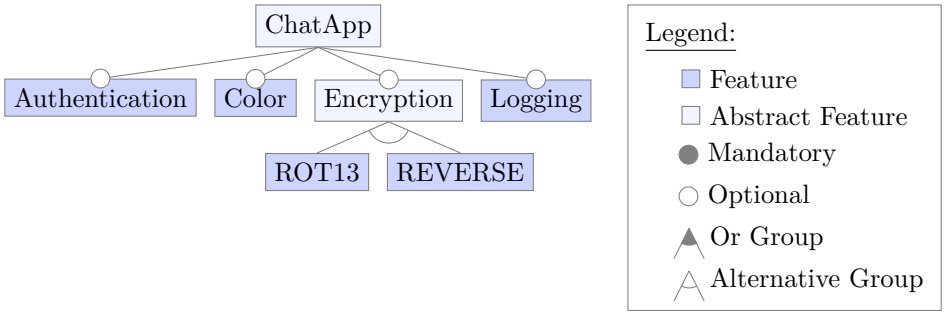


Figure 2: Feature model for the ChatApp.

The following table explains each feature in more detail:

Table 1: Explanation for all features.

| | |
|---|---|
| *Authentication* | Requires that a client must authenticate before it can send messages. |
| *Color* | Text messages can be formatted with a color. |
| *Encryption* | All strings in a message are encrypted with one of the following two (joke) encryption methods. |
| *ROT13* | Apply ROT13 to all alphabetic characters (a-zA-Z). |
| *REVERSE* | Reverse the string. |
| *Logging* | Log important operations. |