# Project guide code generation 2021/2022

## 1. Introduction

This document will guide you through the context of the project (par. 2), requirements (par. 3), user stories (par. 4), deliverables (par. 5), and the grading of the project (par. 6). This project is subject to the Student manual Project-based learning.

## 2. Context

During this term, you will participate in a project of 4 students in the structured development of a banking application in the form of a REST API, developed with the Spring Framework written in the Java programming language. The API shall be made accessible to users through SwaggerUI. Additionally, you will develop a rudimentary but fully functioning frontend interface written in JavaScript using a framework of your choice such as Vue.js or React.

The development process will start with describing an API specification, developed with OpenAPI (formerly: Swagger).[[1]](#footnote-1) The API specification consists of a definition of all API endpoints, parameters (input) and responses for every endpoint (output), and underlying model definitions etc.

The API specification will be used to generate a server stub. The server stub will function as a starting point for implementing the required functionality. The application will be hosted on a public available server.

The application is backed by unit tests and functional tests. Unit tests will be written in Java. Functional tests will be written in Cucumber suite.

## 3. Requirements

The backend will be based on the Spring Boot framework, with an in-memory database to store and retrieve information. The backend is responsible for user management, account management and transaction management. All transactions are logged to a database. The backend must contain security rules that prohibit unauthorized access. The backend is secured using JWT.

The following business rules apply:

1. Accounts can be of type Current or Savings.
2. Accounts are EUR accounts
3. Account numbers must adhere to IBAN standards, the format: NLxxINHO0xxxxxxxxx
4. The bank has its own account NL01INHO0000000001
5. One cannot directly transfer from a savings account to an account that is not of the same customer
6. One cannot directly transfer to a savings account from an account that is not of the same customer.
7. The application is secured with user accounts and roles. The roles can be of type Customer or of type Employee.
8. An employee can also be a customer.
9. The role Customer gives the user access to only their own accounts from which they can see and perform transactions.
10. The role Employee gives the user full access to all accounts (except the bank's own account) and can create and close them.
11. All money flows are done with transactions, depositing and withdrawing (at the ATM) being special cases.
12. A transaction contains timestamp, account from, account to, amount, user performing (can be customer, can be employee)
13. A customer transaction is limited by certain rules:

* Balance cannot become lower than a certain number defined per account, referred to as absolute limit
* The cumulative value of transactions occurring on a day cannot surpass a certain number defined per user, referred to as day limit
* The maximum amount per transaction cannot be higher than a certain number defined per user, referred to as transaction limit

Throughout the project, Git source control will be used for all relevant project files. Individual team members commit their work under their own name. The master branch of the repository should, at all time, represent a stable state of the project.

## 4. User stories

To make the business rules more tangible, we have translated them into user stories:

1. As a bank, I want customers to own Current and Savings accounts.
2. As a bank, I want to offer EUR accounts.
3. As a bank, I want to be able to generate IBAN accounts for customers, adhering to IBAN standards, in the format: NLxxINHO0xxxxxxxxx
4. As a bank, I want to have an account with IBAN NL01INHO0000000001
5. As a bank, I want to prevent transfers from a savings account to an account that is not of the same customer.
6. As a bank, I want to prevent transfers to a savings account to an account that is not of the same customer.
7. As a bank, I want to offer a secure banking experience, by offering user accounts with roles Customer or Employee.
8. As an employee, I can also be a customer.
9. As a customer, I want to be able to access my account details, transaction history, and user details.

As a bank, I want to make sure that other customers cannot access a customer’s account details, transaction history, and user details.

As a customer, I want to be able to perform transactions.

1. As an employee, I want to be able to access all accounts (except the bank's own account).

As an employee, I want to be able to perform transactions for customers.

As an employee, I want to be able to create and close accounts.

1. As a bank, I want to ensure that all money flows are done with transactions, depositing and withdrawing being special cases.

As a customer, I want to be able to perform a deposit and a withdrawal (at the ATM).

1. As a bank, I want to ensure that a transaction contains: timestamp, account from, account to, amount, user performing (can be customer, can be employee).
2. As a bank, I want to prevent an account balance becoming lower than a certain number, referred to as absolute limit.

As a bank, I want to limit the cumulative value of transactions occurring on a day by a certain number defined per user, referred to as day limit.

As a bank, I want to limit the maximum amount per transaction by a certain number defined per user, referred to as transaction limit.

As an employee, I want to be able to configure an absolute limit for an account, the day limit for a user, and the transaction limit for a user.

As a user, I want to see my limits and the maximum amount remaining for a single transaction.

User stories not tied specifically to a business rule:

1. As a customer, I want to be able to find the IBAN for another customer based on their name.
2. As a customer, I want to be able to find transactions between a date range, from/to a specific IBAN, or based on <, ==, > relationships to a certain balance.
3. As a customer, additional to my account balance, I want to see my total balance from all my accounts combined.
4. As a customer, I want to be able to register my user details at the bank.
5. As an employee, I want to be able to see all users for which no account has been created yet, and be able to create an account for these users.
6. As a tech savvy user, I want to be able to write my own frontend for the bank based on the information provided in the SwaggerUI pages.

## 5. Deliverables

The deliverables are as follows:

Week 1: Git repository access for examiners

* Week 4: Feature complete Swagger specification in the form of a Swagger UI URL, along with a list of endpoints each individual team member will implement.
* Week 7: First submission.
* Week 9: Second submission (retake).

The deadlines will be made available on Moodle.

## 6. Grading

A deadline for submission of the project to the examiners ([Erwin de Vries](mailto:erwin.devries@inholland.nl), [Wim Wiltenburg](mailto:willem.wiltenburg@inholland.nl), [Frank Dersjant](mailto:frank.dersjant@inholland.nl), [Mark de Haan](mailto:mark.dehaan@inholland.nl)) will be made available on Moodle. Every group will deliver:

* One or more Git repository links, containing the API source code, specification, unit tests (Java) and functional tests (Cucumber suite), and Frontend code. The work to be examined will be published to a separate branch in the Git repository (“Submission 1” for the first attempt and “Submission 2” for the retake).
* A link where the application binary is hosted. (e.g. Heroku)
* A link where the Swagger UI of the application is hosted. (e.g. Swaggerhub)

Subject to complete and timely delivery of the above mentioned materials, the project will be graded with the following tests:

* *Code Review, 1918IN241A (10%)*

The code of the project will be analyzed on feature completeness (see par. 3), code quality, adherence to framework standards, and consistency. This will lead to a grade for the entire group. **Students will be invited for the code assessment only when the group has a pass for the code review!**

* *Code assessment, 1918IN241D (40%)*

During a 10 minute assessment, individual project members will present their contribution to the code of the project. The individual contribution will be graded on design choices, explanation of the workings of the code, code quality, and adherence to framework standards. The examiners will ask questions regarding these subjects as they see fit. This will lead to an individual grade. During the code assessment for an individual student, the focus should be primarily on the Java code, with the JavaScript code as an optional extra to demonstrate the quality of the work (max 20%).

* *API design, 1918IN241B (25%)*

The API specification in the form of a Swagger UI URL will be graded on feature completeness (par. 3), proper RESTful design and legibility of the documentation, consistency, and conformity to the actual application. This will lead to a grade for the entire group.

* *API testing, 1918IN241C (25%)*

The unit tests and functional tests are graded on validity, acceptable code coverage (90%+), adequacy of functional tests, and level of complexity. This will lead to a grade for the entire group.

* *Process dossier, 1919IN241E (0%)*

The students draw up a cooperation agreement in a group context. The students by turns make the agenda and minutes of the project supervision meetings. In addition, the students reflect on the development of their (project) skills and those of their fellow students by means of the retro evaluation form.

Test criteria:  
- Active contribution to all project management meetings completeness of the following documents (on Moodle):  
- Cooperation agreement;  
- Agenda and minutes;  
- Retro evaluation form.

1. <https://swagger.io/> [↑](#footnote-ref-1)