Bookly - Software Architecture Document

Version 6.0

Revision history

Date	Version	Description	Author
28/11/2019	1.0	Initial Documentation	Alexandra Stober
2/12/2019	2.0	Architecture	Jeanne Helm
30/4/2020	3.0	Deployment Images	Jeanne Helm
29/5/2020	4.0	Update Schemes and pictures	Jeanne Helm
29/5/2020	5.0	Implementation View	Jeanne Helm
29/5/2020	6.0	Patterns	Alexandra Stober

Table of Contents

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This document describes the technical architecture of the bookly project, including module structure and dependencies as well as the structure of classes.

1.3 Definitions, Acronyms and Abbreviations

Abbreviation	Description
API	Application programming interface
MVC	Model View Controller
REST	Representational state transfer
SDK	Software development kit

Abbreviation	Description
SRS	Software Requirements Specification
UC	Use Case
VCS	Version Control System
N/A	Not Applicable

1.4 References

Reference	Date
Bookly Blog	28/10/2019
GitLab Repository	28/10/2019
YouTrack	28/10/2019
SonarQube	29/5/2020
bookly.online	29/5/2020
dev.bookly.online	29/5/2020
keycloak.bookly.online	29/5/2020

1.5 Overview

This document contains the architectural representation, goals and constraints as well as logical, deployment, implementation and data views.

2. Architectural Representation

Our project bookly uses the classic MVC structure as follows:

booklyMVC

As special feature we are using Keycloak as identity management server. It handles automatically the authentication and authorization when a visitor is trying to access some restricted site / data.

booklyMVCKeycloak

3. Architectural Goals And Constraints

As our main technology we decided to use Spring MVC, which is a framework that takes not only care of the backend but also of the frontend. Besides the controller and model language is Java, so that we do not have to care about serialization.

4. Use-Case View

This is our overall use-case diagram:

Overall use-case diagram

5. Logical View

5.1 Overview

We split our architecture according to the MVC architecture as follows:

Spring uses a Dispatcher Servlet that accepts requests and forwards to the view resolver. This resolver serves our view files. See steps 1, 6, 7 and 8. This is our controller according to the MVC model. Controller

The backend serves as the model according to the MVC model. Model

The frontend serves as the view according to the MVC model. View

5.2 Architecturally Significant Design Packages

We have a backend and a frontend module. The backend module contains our model. The frontend module contains our view. The Spring MVC framework is realized. The controller cannot directly access the database.

6. Process View

N/A

7. Deployment View

This is our deployment view: DeploymentView

This is our deployment process. Our code is hosted on GitLab. To be deployed, it is build as JAR (or as Dockerimage). The Files will be copyed on our server and for stage run exposed on port 8080 or on dev on port 7070. We are using the Apache2 webserver as reverse proxy and link the ports on the intended domain. If a visitor wants to log in, every authentication request is redirected to the running Keycloakinstance. DeploymentProcess

8. Implementation View

You can find the stage system here: Stagesystem

You can find the current state of the dev system here: Devsystem

As we are using swagger for the backend implementation the API can be tested and seen here:

Stage: StageSwagger

Dev: DevSwagger

Here you can see our Swagger API:

swagger_basic_error_controller

swagger_edit_page_controller

swagger_friendship_book_controller

swagger_page_controller_and_user_controller

swagger_vue_controller1

swagger_vue_controller2

swagger_vue_controller3

swagger_models_8.JPG

9. Data View

Our data view is modelled as followed:

DataView

10. Size and Performance

N/A

11. Quality/Metrics

To ensure a high quality we are using continuous integration. It automatically builds, tests, measures and deploys the application, if the respective previous step has not failed. This happens periodically and when changes are pushed to a branch.

Pipelines

Our first pipeline is executing our tests. The results are displayed in the log and are summed up in a tab next to the pipeline as well (See tab "Tests"). The test pipeline also calculated the overall test coverage and is creating a badge. For example the badge for the master branch: coverage 29.00%

After that our whole project is measured. We are using SonarQube that analyzes our code. If one wants a specific metric report of a branch or commit one should repeat this specific pipeline. Sonarqube is also able to generate some badges for the last scan:

quality gate passed

After measuring the last pipelines are building and deploying.

For serving a most current documentation of our API, we are using Swagger. It is an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful web services. It is accessible at PATH/swagger-ui.html. It's also possible to test an API and see all possible responses.

12. Patterns

Behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns.We decided to go with the Null Object Design Pattern. The intent of a Null Object is to encapsulate the absence of an object by providing a substitutable alternative that offers suitable default do nothing behavior.

As we have various pictures and stickers, we decided to use the Null Object Design Pattern to implement our DummyImage. It allows us the abstract handling of null away from the client. Apart from that the refactoring enables us to get rid of some code duplication.

Before Design Pattern:

FriendshipBookServiceWithoutSOFA

Excerpt before Design Pattern: FriedshipBookBeforeDesignPatter

After Design Pattern: dtoWithNullObjectDesignPattern Excerpt after Design Pattern: FriedshipBookAfterDesignPatter

The Pattern can be found in the highlighted classes: OverallMarkedPatter