BOOKLY

Installation Guide

1. Prerequisites

- Maven
- npm
- Docker
- Keycloak on Port 8180 (accessible) or use our running instance at keycloak.bookly.online

2.Installation

2.1 Clone code from directory via URL or HTTPS:

URL: git@gitlab.com:project_bookly/bookly.git

HTTPS: https://gitlab.com/project_bookly/bookly.git

Change the application properties of the backend

-> backend/src/main/resources/application.properties

```
 server.port=8080

keycloak.auth-server-url=https://keycloak.bookly.online/auth
spring.datasource.hikari.auto-commit=true
#database
spring.jpa.database=POSTGRESQL
spring.datasource.platform=postgres
spring.datasource.url=jdbc:postgresql://localhost:5432/bookly-dev
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=booklyUser
spring.datasource.password=booklyPW
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

2.2 Build the project in the root folder of the project.

```
mvn install -DskipTests
```

2.3 Go to the backend folder

```
Execute: cd backend
```

2.4 Launch bookly with.

```
docker-compose up
```

2.5 Go to localhost:8080 to use bookly.

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

BOOKLY - Software Requirements Specification

Table of Contents

1. Introduction

Do you still remember the friendship book from your childhood days? Whoever had the most pages and the greatest entries was highly regarded. It was handcrafted, painted, drawn and designed.As books become less important with digitalization, we want to maintain the old charm of friendship books and bring back the fun of designing and collecting.

1.1 Purpose

The purpose of this document is a general description of the bookly project. It explains our vision and all features of the product. Also it offers insights into the system of back- and frontend, the interfaces in both ends for communication and the constraints of the product.

1.2 Scope

This document is designed for internal use only and will outline the development process of the project.The idea shall be realized within a website. Depending on the progress, the same or reduced functions should also be implemented as an Android app.

Subcomponents - Account - Registering, managing, deleting account, Login and Logout to our Application. - Connecting People - Opportunity to share the friendship book with friends by sending a link for editing a page. - Overview - View of the friendship book with all pages and entries. - Edit friendship book - Edit cover, delete friendship book pages, add page (data input only) - Static Pages - FAQ - Impressum - DSGVO - Kontakt - Friend Entry - Possibility to create new entries and to share them with friends - Individual Themes - Choose your own theme and design it by extending it with stickers, pictures and text.

1.3 Definitions, Acronyms and Abbreviations

Term	
SRS	Software Requirements Specification
JSON	JavaScript Object Notation
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
API	Application Programming Interface
FAQ	Frequently Asked Questions
REST	Representational State Transfer

1.4 References

Title	Date
Blog	17/10/2019
Gitlab	17/10/2019
Use Case Diagram	17/10/2019
Youtrack	02/12/2019
SonarQube	29/5/2020
bookly.online	29/5/2020
dev.bookly.online	29/5/2020
keycloak.bookly.online	29/5/2020

1.5 Overview

The next chapters provide information about our vision based on the use case diagram as well as more detailed software requirements.

2. Overall Description

2.1 Product perspective

The product is supposed to be an open source. It is a web based system implementing client-server model. Bookly provides a friendship book from the old days.

The following are the main features that are included in the bookly website.

- User account/interface: The system allows the user to create their accounts in the system and provide features of updating and viewing profiles. You can view your friendship and invite friends to design a page.
- Number of users being supported by the system: Though the number is precisely not mentioned but the system is able to support a large number of online users at a time.

2.2 Product functions

At registration, the data provided by the user is stored in the backend. It is needed to log in, edit the profile and also provides the basis for a permission-system. According use cases are: - User System: login, register, invite friends, logout, edit, close account

A user can give permissions to other users so they're able to design a new page in the friendship book. Users are able to design their own friendship book covers. According use cases are: - friendship book: design cover, upload custom images, edit title, delete/add pages - page: add entries, add cover image, add stickers, share page

Additional content: - footer and header for navigation and additional information

Our Use-Case-Diagram

UseCaseDiagram

2.3 User characteristics

It is basically for everyone who feels nostalgic and wants to dwell in the memories of their childhood. Nevertheless, our main target group consists of children.

2.4 Dependencies

The project has the dependencies that are due to our choice in technology. Requirements subsets - VM with debian buster - Apache2 webserver - Wordpress - Domain - Docker - Java (including Jacoco, Maven, Lombok, SpringBoot, Hibernate JPA, Swagger, Cucumber, JUnit, Mockito) - VueJS (including Bootstrap) - GitLab (including CI) - IntelliJ IDE - YouTrack - Keycloak (including OAuth2 Module for SpringBoot and VueJS) - PostgreSQL (including PgAdmin) - SonarQube - H2

2.5 Constraints

Our database limit is under 15 GB. We have 2 Servers with 2GB RAM each. The metrics server and the blog are seperated from the application server.

3. Specific Requirements

This section contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features. In the following sections our standard requirements for our functions are described. If they vary, these functions will be explained explicitly.

3.1 Functionality – Data Backend

The backend is needed to separate the user interface from the data storage. It verifies if the correct permissions are present to request data or to ensure that incoming data is properly parsed and saved correctly. For security reasons data is filtered by the backend. It is then packed in the right format which the next chapter describes. The data is kept inside a database and maintained by the backend.

3.1.1 User system

For the identity management we are using Keycloak. There we have the opportunity to login/logout/register/delete an user. Additional information about a user that uses our application we store the data in our application too and are synchronising the data. The According use cases:

- Operate Account
- Navigate Header
- Navigate Footer

3.1.2 Friendship book

A friendship book consists of a cover and friendship book entries. These entries can be created by the friendship book owner and can be edited public by a friend. According use cases are:

- Manage book pages/entries
- Manage book
- Share code to invite friends
- Manage page decorations
- Manage cover decorations

3.1.3 Read data given over API endpoints

For the communication between both sides (frontend and backend) a universal data format is needed, therefore JSON is used. The frontend sends data in JSON to the backend in form of a request and waits for a response from the backend which also answers with JSON.

3.1.4 Parse data

Incoming data needs to be checked if the sent values represent the correct data type and if the user that sends the request has the permissions to do so.

3.1.5 Provide data

After data is requested from the frontend and the user is authorised to do so, the backend sends data. In addition, the response contains a HTTP status code even if the request failed so that the frontend knows if it just received data or an error.

3.2 Functionality – User Interface

The frontend provides an user interface for the users to interact with and is able to request data from the data backend.

3.3 Usability

We will build the user interface intuitive, so that a new user does not necessarily need an explanation. If questions arise our interface provides a comprehensive FAQ.

3.4 Reliability

In the following we describe the availability, MTBF and MTTR, accuracy and bug classes we strive for.

3.4.1 Availability

Since we are trying to focus on a bug free application rather than caring about hosting it on our own, the availability depends on the hosting provider we choose. Due redundancy and other security arrangements, most providers can ensure an uptime over 99.9% of the time its hosted at their datacenter.

3.4.2 MTBF, MTTR

If the application fails due an hardware issue, then the mean times are up to our hosting provider. Since the ensured uptime of most hosting providers is 99.9%, they try to fix the issue within a few minutes. However, if the application fails due a bug in our code, we can revert the code to a previous version that worked fine. This shouldn't take more than one or two hours from the point on we noticed.

3.4.3 Accuracy

- N\A

3.5 Performance

In general, we try to keep to user experience fluent and response times low. High peaks can still appear when the hosting provider is currently having issues.

3.5.1 Response time

Should be as low as possible. Maximum response time is 3 seconds. Average response time should be less than 1 second.

3.5.2 Throughput

Should be as large as possible.

3.5.3 Capacity

The current system should be capable to manage 100k of registered users and up to 1k users at the same time.

3.6 Supportability

Our frontend, backend and each functionality will be clearly separated and we try to stick to naming conventions which are common in the used technologies. Furthermore we aim to keep our code clean which we can't guarantee though. Thereby we make it easy to understand our infrastructure and avoid possible confusion when one needs to edit older parts of the application.

3.7 Design Constraints

We are focused on building a modern-looking application using modern technologies. Of course there are other smaller libraries and frameworks used than the ones that are listed, but they represent just a small fraction of the whole project and aren't worth mentioning.

3.7.1 Development tools

- Git: version control system
- JetBrains IntelliJ: Spring MVC backend development (including Swagger) and Vue.js frontend development
- YouTrack: Project planning tool
- Database: H2 (local), Postgres (prod)
- Browser: Firefox, Chrome

3.7.2 Spring Boot

Spring Boot is built on top of the Spring framework and provides the developer with helpful features to create and run web applications. In our case, a REST Web Service which represents the interface between our front- and backend. As we want to benefit from the newest features of Java 8, the platform this service will be hosted on needs to support Java 8.

3.7.3 Supported Platforms

Since bookly will be a web application the user only needs a modern web browser and a stable internet connection. With modern web browser we mean the current versions of Mozilla Firefox or Google Chrome.

3.8 Online User Documentation and Help System Requirements

We want to provide a small F.A.Q. for possible questions that can come up when using our application. Since it can be frustrating for children if they don't know what to do we will include step-by-step instructions and enough pictures to show the user exactly what to click at.

3.9 Purchased Components

- N\A

3.10 Interfaces

3.10.1 User Interfaces

Our User Interface will provide one page for each implemented functionality. To navigate between these sites the user will find a menu bar at the top.

3.10.2 Hardware Interfaces

- N\A

3.10.3 Software Interfaces

Our backend implements a REST-API, whose URLs our frontend can invoke with Http-Requests. We will prepare base URLs for data concerning the user. For each friendship book base URL one can choose to either leave it public or private. These data will be processed by our backend and then passed to our database. The connection between our backend and the database will be managed by H2.

3.10.4 Communications Interfaces

Each HTTP-Request and Response contains a JSON. By interpreting its content our system will be able to transfer all needed data between front- and backend.

3.11 Licensing Requirements

Our project runs under the MIT License. This way everyone is allowed to create his own version.

3.12 Legal, Copyright and other Notices

- N\A

3.13 Applicable Standards

- N\A

4. Supporting Information

For a better overview, watch the table of contents and/or references.

Test plan

1. Introduction

1.1 Purpose

The purpose of the Iteration Test Plan is to gather all of the information necessary to plan and control the test effort for a given iteration. It describes the approach to testing the software. This Test Plan for vnv supports the following objectives: - Identifies the items that should be targeted by the tests. - Identifies the motivation for and ideas behind the test areas to be covered. - Outlines the testing approach that will be used. - Identifies the required resources and provides an estimate of the test efforts.

1.2 Scope

This document describes the used tests, as they are unit tests and functionality testing.

1.3 Intended Audience

This document is meant for internal use primarily.

1.4 Document Terminology and Acronyms

- **SRS** Software Requirements Specification
- **n/a** not applicable
- **tbd** to be determined

1.5 References

Reference
Blog
SAD
Function Points
UC1 Create Book
UC2 Read Book
UC3 Update Book
UC4 Create Page
UC5 Read Page
UC6 Update Page
UC7 Share Link Visibility Invite
UC8 Manage Page Decorations
UC9 Manage Cover Decorations
UC10 Navigate Footer
UC11 Navigate Header
UC12 Operate Account

2. Evaluation Mission and Test Motivation

2.1 Background

By testing our project, we make sure that all changes to the sourcecode do not break the functionality. Also by integrating the test process in our deployment process, we make sure that only working versions of our project getting deployed. So the web application is always available.

2.2 Evaluation Mission

Our motivation in implementing tests came at an early stage to recognize the need for errors and to ensure the functionality and thus the outstanding quality of the software.

2.3 Test Motivators

Our testing is motivated by - quality risks - technical risks, - use cases - functional requirements

3. Target Test Items

The listing below identifies those test items (software, hardware, and supporting product elements) that have been identified as targets for testing. This list represents what items will be tested.

Items for Testing: - java backend - field test - database operations

4. Outline of Planned Tests

4.1 Outline of Test Inclusions

Unit testing the Java backend, functional testing of the Web frontend and database integrity

4.2 Outline of Other Candidates for Potential Inclusion

Stress testing the application

4.3 Outline of Test Exclusions

What we are not planning to do is attack simulation.

5. Test Approach

5.1 Initital Test-Idea Catalogs and Other Reference Sources

n/a

5.2 Testing Techniques and Types

5.2.1 Database Integrity Testing

Technique Objective	The connection with the database shall be opened. Data shall be created, deleted, updated or read without any problem.
Technique	Authentication will be mocked. Any service method or database request will be tested by using the local runtime database H2
Oracles	The database connection is possible and the database user has all privileges he needs.
Required Tools	JUnit + Mockito, H2
Success Criteria	successful scenarios, all tests will pass, no strange behaviour will occur
Special Considerations	-

5.2.2 Function Testing

Technique Objective	Every service request shall be done correctly. Possible exceptions are caught correctly.
Technique	Cucumber tests are doing the integration testing, Unit tests for the backend are done by JUnit and Mockito
Oracles	user enter valid data, for example a valid username and a valid password

Required Tools	Cucumber + JUnit
Success Criteria	successful scenarios, all tests will pass, no strange behaviour will occur
Special Considerations	-

5.2.3 Business Cycle Testing

n/a

5.2.4 User Interface Testing

n/a

5.2.5 Performance Profiling

n/a

5.2.6 Load Testing

n/a

5.2.7 Stress Testing

n/a

5.2.8 Volume Testing

n/a

5.2.9 Security and Access Control Testing

n/a

5.2.10 Failover and Recovery Testing

n/a

5.2.11 Configuration Testing

n/a

5.2.12 Installation Testing

n/a

5.2.13 Field Testing

Technique Objective	The whole application shall be tested by a real user.
Technique	Before the field test will start, the developing team will test the application by themselves, too. Each user can then try out the website on his own. We can ask or answer further questions while he/she is testing the application
Oracles	user uses the website exactly as we do, user enters data in the expected format
Required Tools	Multiple computers, chatting tool for remote support, support from parents or teachers, mail-address required
Success Criteria	Feedback, finding bugs, no strange behaviour will occur
Special Considerations	-

6. Entry and Exit Criteria

6.1 Test Plan

6.1.1 Test Plan Entry Criteria

Building a new version of the software with Gitlab-CI will execute the testprocess.

6.1.2 Test Plan Exit Criteria

When all tests pass without throwing an exception.

6.1.3 Suspension and Resumption Criteria

It will be possible to skip the tests for faster deployment during development. Apart from that the tests will always run till the end.

7. Deliverables

7.1 Test Evaluation Summaries

Test evaluation will be available online every time the tests are run in gitlab. The report is generated with the help of surefire and jacoco. Detailed test results are available within every pipeline. TestsWithinPipeline

7.2 Reporting on Test Coverage

Test coverage numbers will be generated by jacoco and gitlab. The current coverage status of the master branch: coverage 29.00%

7.3 Perceived Quality Reports

We are using sonarqube for metrics. After every push to gitlab a pipeline is triggered that is updating the metrics on sonarqube.

quality gate passed

If one wants a specific metric report of a branch or commit one should repeat the specific pipeline. PipelineMetric

7.4 Incident Logs and Change Requests

With the help of the above tools we will integrate a check into gitlab for pull requests.

7.5 Smoke Test Suite and Supporting Test Scripts

n/a

7.6 Additional Work Products

7.6.1 Detailed Test Results

Detailed test results are available within every pipeline. TestsWithinPipeline

The test coverage of the whole project can be found here: Coverage

7.6.2 Additional Automated Functional Test Scripts

n/a

7.6.3 Test Guidelines

In general all testable code should be tested. Due to time constraint this is of course not always possible. Therefore we set a bound of 30% for coverage

7.6.4 Traceability Matrices

n/a

8. Testing Workflow

Every developer can run the tests in his IDE and is required to do so on a regular basis. Furthermore tests will be run automatically on

9. Environmental Needs

This section presents the non-human resources required for the Test Plan.

9.1 Base System Hardware

The following table sets forth the system resources for the test effort presented in this Test Plan.

Resource	Quantity	Name and Type
Integration Server	1	Debian Buster Server
Server Name		bookly.online
Development Server	1	Debian Buster Server
Server Name		dev.bookly.online
Metrics Server	1	Debian Buster Server
Server Name		sonarqube.bookly.online
Database	3	PostgreSQL & H2
Database Name		Production: PostgreSQL booklyDev
Database Name		Development: PostgreSQL booklyProd
Database Name		H2 on localhost

9.2 Base Software Elements in the Test Environment

The following base software elements are required in the test environment for this Test Plan.

Software Element Name	Version	Type and Other Notes
macOS Catalina	10.15.4	Operating System
Windows	10	Operating System
Firefox	76	Internet Browser
Chrome	81.0.4044.138	Internet Browser
Geckodriver	0.26.0	Application
PostgreSQL	11.7	Database
H2	1.4.200	Database

9.3 Productivity and Support Tools

The following tools will be employed to support the test process for this Test Plan.

Tool Category or Type	Tool Brand Name
Code Hoster	gitlab.com
Test Coverage Monitor	Maven Surefire Plugin and Jacoco
CI Service	Gitlab CI
Metrics Tool	Sonarqube

9.4 Test Environment Configurations

In order to be able to run the tests you need to have a database set up and running with the specification in the spring database file.

10. Responsibilities, Staffing, and Training Needs

10.1 People and Roles

This table shows the staffing assumptions for the test effort.

Human Resources

Role	Minimum Resources Recommended (number of full-time roles allocated)	Specific Responsibilities or Comments
Test Manager	1	Provides management oversight. Responsibilities include: planning and logistics agree mission identify motivators acquire appropriate resources present management reporting advocate the interests of test evaluate effectiveness of test effort
Test Designer	1	Defines the technical approach to the implementation of the test effort. Responsibilities include: define test approach define test automation architecture verify test techniques define testability elements structure test implementation
Tester	1	Implements and executes the tests. Responsibilities include: implement tests and test suites execute test suites log results analyze and recover from test failures document incidents
Test System Administrator	1	Ensures test environment and assets are managed and maintained. Responsibilities include: administer test management system install and support access to, and recovery of, test environment configurations and test labs
Database Administrator, Database Manager	1	Ensures test data (database) environment and assets are managed and maintained. Responsibilities include: support the administration of test data and test beds (database).
Implementer	3	Implements and unit tests the test classes and test packages. Responsibilities include: creates the test components required to support testability requirements as defined by the designer

10.2 Staffing and Training Needs

n/a

11. Iteration Milestones

Milestone	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
Have Unit Tests	13.5.2020	13.5.2020	4.6.2020	28.5.2020

Milestone	Planned Start Date	Actual Start Date	Planned End Date	Actual End Date
Have Integration Tests	13.5.2020	13.5.2020	4.6.2020	28.5.2020
Have Field Tests	1.6.2020	1.6.2020	20.6.2020	X
20% coverage	13.5.2020	13.5.2020	4.6.2020	28.5.2020
Tests integrated in CI	13.5.2020	13.5.2020	13.5.2020	13.5.2020

12. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Untestable features in the framework	Cannot be avoided	Try to test it with integration tests
Testing scenario is not covered	Carefully design tests	Add scenario
Technical Difficulties for Testers	Try to help remote	Try it out on different browsers before and check the availability on multiple systems

13. Management Process and Procedures

n/a

BOOKLY - Software Requirements Specification

Use-Case Specification: Create Page

1. Use-Case: Create Page

1.1 Brief Description

The creation of a new page. The owner of the book has the opportunity to add new pages to his book. Later this option will lead to a link sharing.

2. Flow of Events

Create Book Entry

3. Preconditions

The User either received a link or uuid to create a page for a friend.

4. Function Points

Create Page FPs

BOOKLY - Software Requirements Specification

Use-Case Specification: Create Book

1. Use-Case: Create Book

1.1 Brief Description

After registering a user will have his own friendship book. It's correlated with his account.

Friendship Book Cover

2. Flow of Events

create friendship book cover

3. Special Requirements

3.1 Owning An Account

In order to browse through a friendship book the user has to have an account. After the registration one can add further styles. See Design Manage Cover.

4. Preconditions

4.1 The user has to be logged in

To ensure proper privacy of a friendship book the user has to be logged in when working with his book.

5. Function Points

Create Book FPs

BOOKLY - Software Requirements Specification

Use-Case Specification: Edit Page

1. Use-Case: Edit Page

1.1 Brief Description

A friend can enter a page by using a direct link or uuid. (e.g. he wants to change his favorite food from spaghetti to Mac'n'Cheese). For updating decorations see ManagePageDecorations

Manage Page Decorations

2. Flow of Events

Edit Book Entry

3. Special Requirements

3.1 Link or Account

In order to edit a page entry you either have a direct link or uuid.

4. Preconditions

4.1 Entry already created

The page entry to be edited has to be created before in order to edit it.

5. Function Points

Update Page FPs

BOOKLY - Software Requirements Specification

Use-Case Specification: Manage Book

1. Use-Case: Manage Book

ManageBook

1.1 Brief Description

This use case describes the creation, reading, updating and deleting of a book(CRUD). Since the creation and display of a book is more complex, they were defined as separate UseCases and stored in separate files. The activity diagram also does not show their functions in detail, but refers to their outsourced activities.

2. Flow of Events

For further details for reading, creating and updating pages look into their linked use case specification. ManageBookFlow

2.1 Basic flow

In general a user will create a friendship book cover and enter the data of this cover page. You will be able edit/update it later and you are able to delete the friendship book in its entirety.

2.2 Creation

After registering a user will have his own friendship book. It's correlated with his account.

Create Book

2.3 Read

A user can browse through his book. Starting with the cover, he can reach all entries by clicking on arrows to the right or left on the bottom of the page. The view of the cover or a page includes a photo or picture, labels and text.

Read Book

2.4 Edit

During editing the user can modify the title and subtitle of his friendship book. Later on the user can also select a background image, a theme and further decorations. Update Book

See also Design Manage Cover

2.5 Delete

A friendship book shall only be deleted by the owner. This is only possible by deleting his whole account. See Operate Account

3. Special Requirements

3.1 Owning An Account

In order to create a friendship book the user has to have an account. For editing the cover, the user has to own the account and view the page. After clicking on the pencil, the user will be able to make changes to his cover page.

4. Preconditions

4.1 The user has to be logged in

To ensure proper privacy of a friendship book the user has to be logged in when working with his book.

5. Postconditions

5.1 Create

After adding data to the text fields and/or photos, the user can save his friendship book cover.

5.3 Edit

The user can edit your book cover as often as he like. He only needs to be logged in. For Managing Pages see ManagePage

5.4 Delete

After confirming the deletion dialog of a book, the book pages will be no longer displayed in the list overview and all the book data will be permanently removed from the database. The book is correlated with the user account and only deletable by removing the account. See OperateAccount

6. Feature Files

Feature Manage Book

BOOKLY - Software Requirements Specification

Use-Case Specification: Manage Page

1. Use-Case: Manage Page

1.1 Brief Description

This use case describes the creation, reading, updating and deleting of a book page (CRUD). Since the creation, editing and display of pages are more complex, they were defined as separate UseCases and stored in separate files. The activity diagram also does not show their functions in detail, but refers to their outsourced activities.

2. Flow of Events

For further details for reading, creating and updating pages look into their linked use case specification. ManagePageFlow

2.1 Basic flow

In general a user will create a friendship book page/ entry and list all inserted data of this specific page. One will maybe edit/update it by sharing a specific link to a page or delete it.

2.2 Creation

The creation of a new page. The owner of the book has the opportunity to add new pages to his book. He will receive a uuid that he can share with anyone else for editing its data. See ShareLink

Create Book Entry

2.3 Read

A user can view an entry by browsing through his book. (Starting with the cover, he can reach all entries by arrows.) The functionality to the view of only one page includes a photo, labels, text and further decorations. See ManagePageDecorations

Read Book Entry

2.4 Edit

During editing the visitor/writer can modify the text entries but also change the pictures and decorations. Edit Book Entry

See also ManagePageDecorations

2.5 Delete

A page entry shall only be deleted by the user itself. The user can manage his pages within his profile. There he can scroll trough all of his pages and manually delete a page if he wants to. By deleting his whole account, all pages will be deleted too. Profile

See also OperateAccount

3. Special Requirements

3.1 Owning An Account

In order to create a friendship book the user has to have an account. For editing a page, the user has to create the page and share it with the friend.

4. Preconditions

4.1 The user has to be logged in

To ensure proper privacy of a friendship book the user has to be logged in when working with his book.

5. Postconditions

5.1 Create

After a new page is created, it must be saved in the database and displayed in the list overview.

5.2 Read

The user is able to read a page. No changes can be made.

5.3 Edit

After a page has been edited by a friend, the data must have been changed in the database and thus when reloaded shown in the list overview.

5.4 Delete

After confirming the deletion modal of a page, the page will be permanently removed and no longer displayed in the list overview and in the database too.

6. Feature Files

Feature Manage Page

BOOKLY - Software Requirements Specification

Use-Case Specification: UC Navigate Footer

1. Use-Case: Navigate Footer

navigationFooter_mockup

1.1 Brief Description

There shall always be a footer presented when scrolling to the bottom of a site. With this footer the user shall easily navigate to the following locations: - About us - Blog - ToS (Terms of Service) - GitLab - Help

2. Flow of Events

Navigate Footer

3. Preconditions

There are no preconditions, the footer shall be displayed to every visitor of the website

4. Function Points

Navigate Footer

BOOKLY - Software Requirements Specification

Use-Case Specification: Navigate Header

1 Brief Description

The header is intended to quickly access the main functionality of our webapp.

2. Flow of Events

HeaderFlow

2.1 Basic flow

The user can access its profile settings or logout, go to his own friendship book or invite some friends to look into his book. Header

See Account Operations, Read Book and sharelink_visibility_invite

3. Preconditions

The visitor has to be logged in to see the header.

4. Function Points

Navigate Header

BOOKLY - Software Requirements Specification

Use-Case Specification: Operate Account

1 Brief Description

This use case describes the (un)registration, login and logout of our web app. For every account flow, we decided to integrate the identity management solution *Keycloak*. Every protected side will redirect to the running Keycloak instance where it handles the account requests. This shall have the security advantage that our backend will never see any kind of sensitive credentials or has to handle and update authentication processes.

2. Flow of Events

Account

2.1 Basic flow

For accessing protected content, the visitor will be redirected to the login and registration form of keycloak. There one can login or register the account. After a successful authentication, one will be redirected back to the application. As one is logged in, one can also change its profile data. For this, a custom user interface shall be implemented and send the relevant data to Keycloak where it will be updated. If one does not intend to change its data or do anything else on our application, one can log out or go even further and delete its whole account and data.

2.2 Login

For accessing protected content, the visitor will be redirected to the login form of keycloak. A username and a password is needed, also some personal data. Is the login can be requested. Is the login successful, one will be redirected to his profile page. Otherwise a error message will show up and he can repeat his login or register.

Login

2.3 Logout

When browsing trough his pages or editing his cover, a user has can select an option in the header of the website. By clicking on logout, he will be logged out and redirected to the login mask.

Logout

2.4 Register

For registering a username and a password is needed, also some personal data. If the username is already taken an error message should show up. To protect against bots, a captcha query is built in. In addition, the general data protection guidelines must be accepted. After confirmation, a new account will be created and redirect the user directly to the startpage. Registration

Account Management

By clicking on the settings in the header, a user can click on profile and edit on its profile page his data. This is a custom page and no redirection to Keycloak. Every request will be handled in the backchannel and send therefore to Keycloak. Every personal data is allowed to be changed and updated (but not the username as it is used for the authentication). Header

A user can also delete his account including every data of his friendship book by going to his profile and selecting delete. Account Management

3. Preconditions

The visitor has to be logged in to manage its user data. If not he has to register.

BOOKLY - Software Requirements Specification

Use-Case Specification: Read Book

1. Use-Case: Read Book

1.1 Brief Description

A user can browse through his book. Starting with the cover, he can reach all entries by clicking on arrows to the right or left on the bottom of the page. The view of the cover or a page includes a photo or picture, labels and text.

Cover

2. Flow of Events

ReadBookFlow

3. Special Requirements

3.1 Owning An Account

In order to browse through a friendship book the user has to have an account. After the registration one can create his book including an own cover and start browsing through his book there. (BUT: he can make his friendship book visible to others if he wants to. But there one can only look inside the book, not manage it. See the privacy settings in the profile.)

4. Preconditions

4.1 The user has to be logged in

To ensure proper privacy of a friendship book the user has to be logged in when working with his book. (BUT: he can make his friendship book visible to others if he wants to. But there one can only look inside the book, not manage it. See the privacy settings in the profile.)

5. Function Points

Read Book FPs

BOOKLY - Software Requirements Specification

Use-Case Specification: Read Page

1. Use-Case: Read Page

1.1 Brief Description

A user can read page entries from his friends.

2. Flow of Events

Read Book Entry

3. Special Requirements

3.1 Owning An Account

In order to browse through the pages (his friendship book) the user has to have an account.

4. Preconditions

The user is logged in and has received at least one page entry

5. Function Points

Read Page FPs