Week 2 – Roles and Technologies

blog.bookly.online

admin October 13, 2019

For this weeks Developer Post we'd like to give a brief overview of our technologies and role allocation.

We decided to divide our roles the following way:

Alexandra

- **Use-Case Manager:** Describe and phrase the features / design of the website with the team
- **PO / Scrum Master:** Phrase acceptance criteria of the features + Coordinate sprints + Manage YouTrack + Assign Tags

Jeanne

- **Backend:** Create database structure, logic, and interfaces (API)
- **Testing Manager:** Test the interfaces (APIs) between Frontend, Backend and Database and the functionality of the Frontend

Nico

- **Deployment:** Host domain and integrate CI (and Docker)

- **Frontend:** Implement the design described in the Use-Cases + Webtests

These roles can now be transcribed into the **IBM Rational Unified Process (RUP)**

Role	Assigned	Previous Role
Requirements Specifier	Alexandra	Use-Case Manager
Designer	Nico	Frontend
Implementer	Jeanne	Backend
Tester	Jeanne	Testing Manager
Deployer	Nico	Deployment
Project Manager	Alexandra	PO / Scrum Master

We decided to rotate by writing blog posts and reviews. We will try to do periodic some code reviews before a new feature is released. Everyone is supposed to get an insight into the work of the team and discussions should be stimulated if and how to implement something better.

In terms of technologies, we decided to go with the following:

Project Management	YouTrack / GitLab / Discord (for communication)
Backend	Database: SQLite or Postgres (production)/ H2 (local) Framework: Spring MVC, Spring Data JPA
Frontend	Vue.js
Testing	JUnit
Security	Basic Auth optional: OAuth 2.0

Known Issues

Missing SSL Certificate on our website (we are working on that issue :P)

Update

Our current issues are available in our agile board on Youtrack:
https://nicoschinacher.myjetbrains.com/youtrack/agiles/

We divided our issues in two types: a story (the same terminology as defined by scrum) and a task. We are using a story for our features that we want to implement because it can have many subtasks for categorizing them. For example, we want to implement the login and create a story for this. Then we are defining two subtasks that belong to the story: Frontend and Backend. Those can move seperatly through each state and have different assignees.
We are using a task for any other issue or exercise we have to do that is not a feature for our endproduct or a subtasks as part of a story.

Both, a story and a task, are categorized by a RUP workflow and a RUP phase. By clicking on the issue you can see the selected types on the right side.

Week 3: SRS

blog.bookly.online

alex October 18, 2019

We finished the first draft of our Software Requirement Specification (SRS) and our Use Case Diagram this week. Keep in mind that this is only our very first draft so the SRS will grow in the following weeks and months.

You can find the whole documentation on our Gitlab site. We also use Gitlab as our version control:

> Software Requirements Specification

To complete the SRS we also created an Overall Use Case Diagram which you can find here:

> Overall Use Case Diagram

Week 4: Use Cases

blog.bookly.online

jeanne October 27, 2019

This week we revised the scope of our features and defined our system. The use case diagram has also been extended accordingly. We will gradually formulate and document the features that we want to implement by December.

Our system on which the application runs is a Linux server. For the time being, we have 15GB of storage space available. We set up a user for the CI deployment of Gitlab. He should only have the privileged right to update and restart our application. The idea behind it is as follows:

- Gitlab builds the application with all dependencies
- GitLab logs on to the Linux server via a securely stored SSH key.
- then the finished JAR file of our SpringBoot application is copied to the system and overwrites the old version
- an already configured system command restarts the application process with the exchanged JAR file

We want to deploy our application automatically because this is a process that is repeated permanently over the two semesters. The setup saves us a lot of time. We want to bring our changes live on a regular basis and always have access to an up-to-date application status.

And now to the actual use cases

Even though the account processing does not count as a use case, we still included it in the SDS and visualized it. See the following document: https://gitlab.com/project_bookly/bookly/tree/master/design/Account.md
(Don't worry, we didn't count it as a use case and did a bit more)

We have divided the functions of the friends book into page entries and the book per se. A page can be added, edited, deleted and viewed. A book cover can be edited and viewed. (Later several books will be added and can also be deleted). Since these are the most basic functionalities of our application, we don't want to classify them as just two CRUD-UCs, because they are more complex and take longer to implement. See the latest Manage Page and Manage Book.

Each UC is described narratively in words, as a mockup and as an activity diagram. The sub components are also linked in our latest SRS.

Week 5: Cucumber!

blog.bookly.online

admin November 3, 2019

Sup y'all, it's bookly,

and in this week we added Cucumber to our Project using Maven. You can already check it out on our Master-Branch on GitLab! We had some problems with merging our Maven and Master branch, but we finally made it work.

You can check out our first .feature file here: https://gitlab.com/project_bookly/bookly/blob/master/backend/src/test/resources/dhbw/online/bookly/manage_page.feature

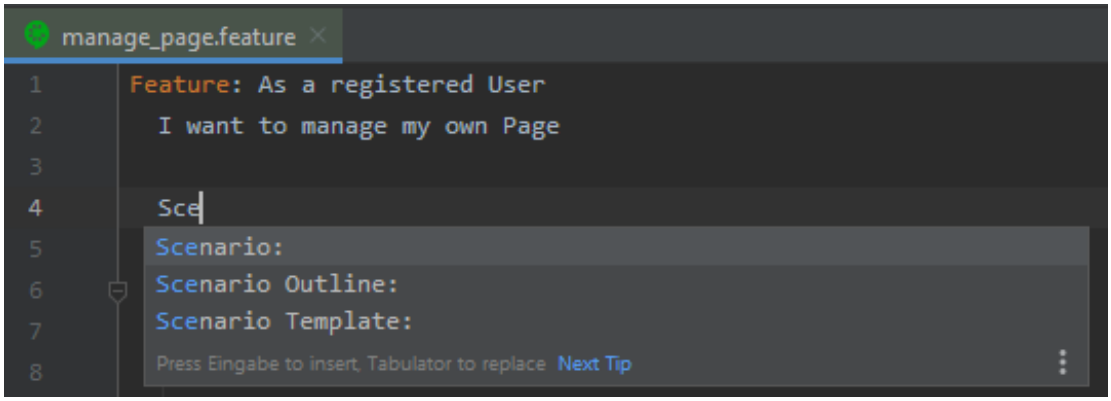
The second ManageBook.feature is here:

https://gitlab.com/project_bookly/bookly/blob/master/backend/src/test/resources/dhbw/online/bookly/ManageBook.feature

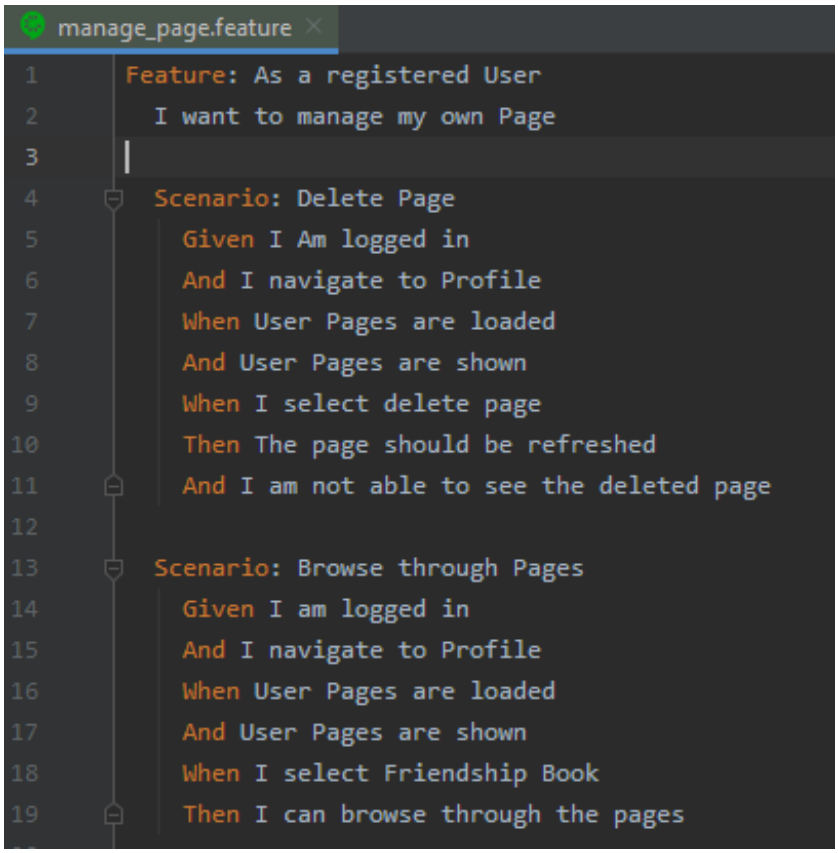
The corresponding step definitions can be found here:

https://gitlab.com/project_bookly/bookly/blob/master/backend/src/test/java/dhbw/online/bookly/Stepdefs.java

We also made sure that we have auto-completion and the correct formatting for our .feature files:



Proof that auto-complete works



Proof that IDE has correct formatting

We also added some User-Cases you can check out on our GitLab!
https://gitlab.com/project_bookly/bookly/blob/master/design/Account.md
https://gitlab.com/project_bookly/bookly/blob/master/design/ManageBook.md
https://gitlab.com/project_bookly/bookly/blob/master/design/ManagePage.md

We also setup our first Spring/Vue.js test application and finished our CI, which means we can deploy our project (Master-Branch on GitLab) to our Webserver with just one click.

Another issue was that HTTPS wasn't working, we finally fixed this problem using LetsEncrypt.

Best regards,

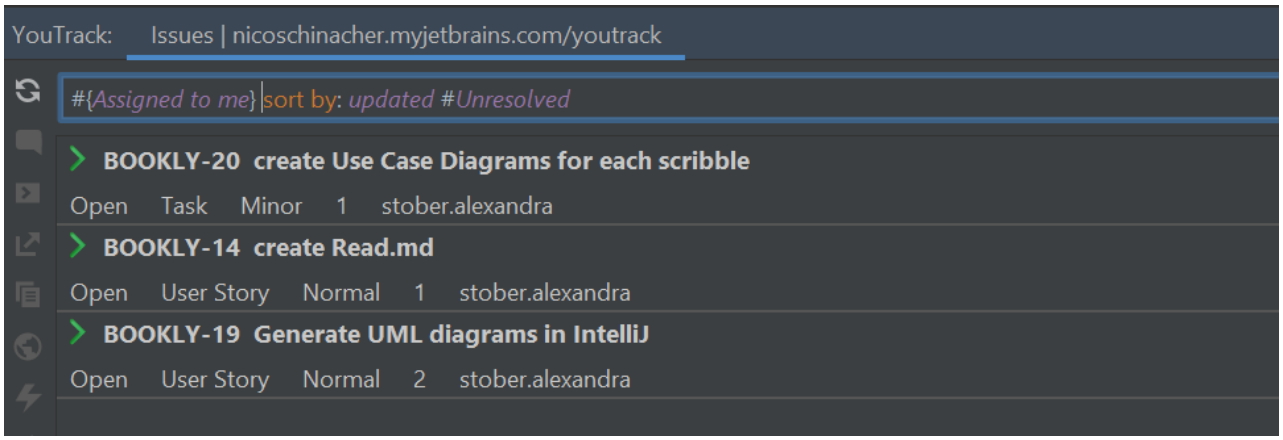
Team Bookly

Week 6: SCRUM

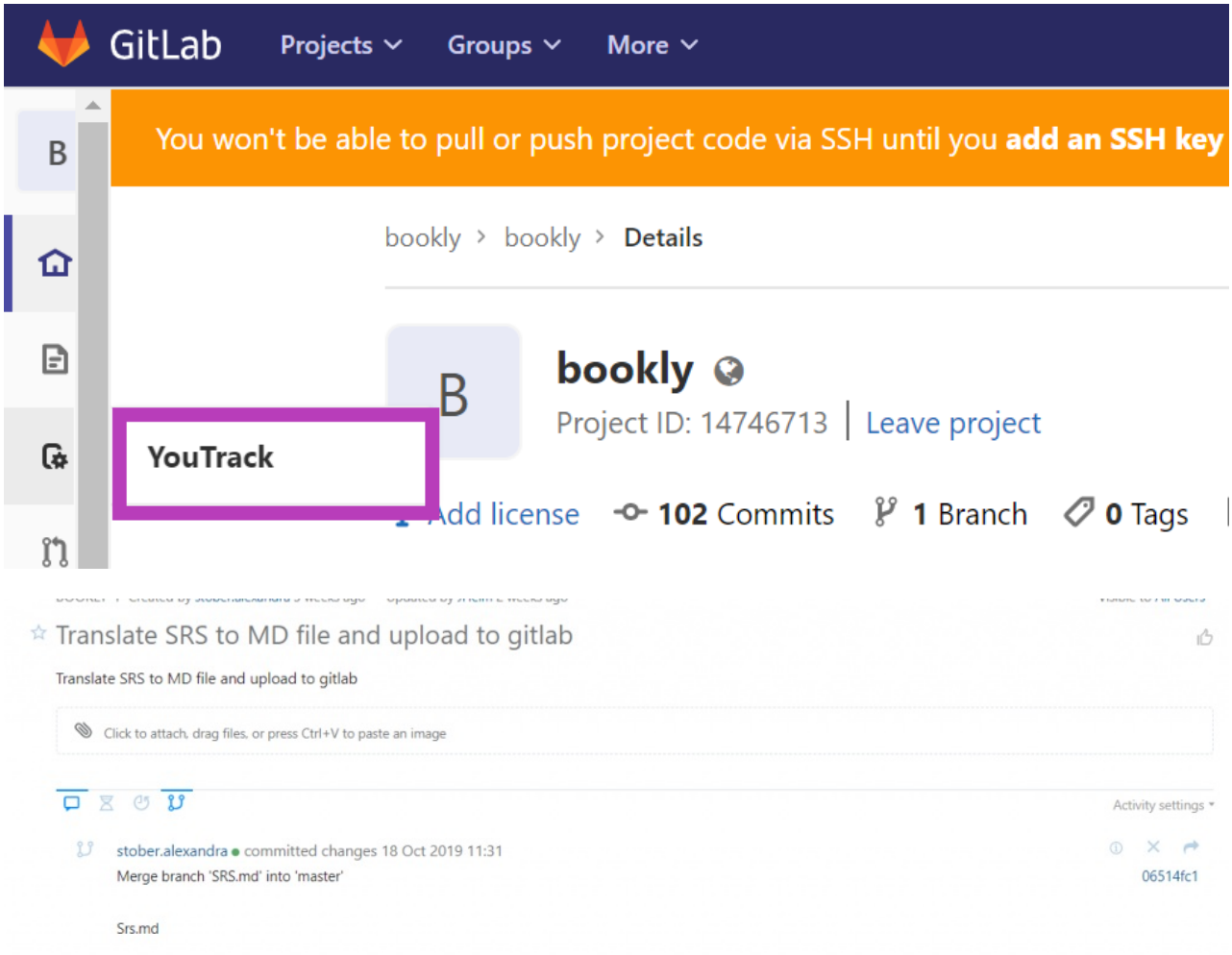
blog.bookly.online

alex November 10, 2019

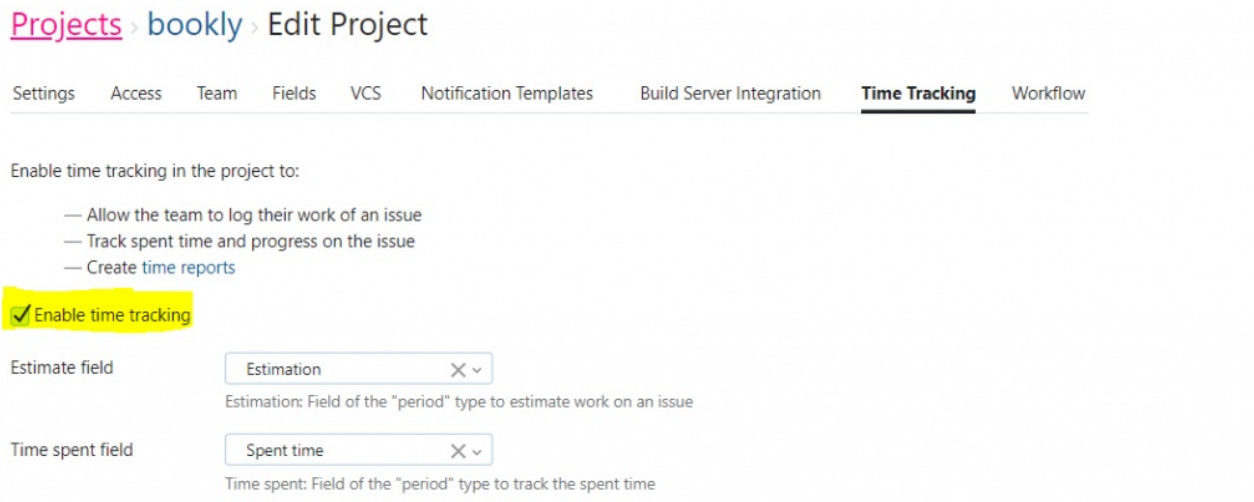
Integration with IDE for task list



Integration with GIT



Time tracking

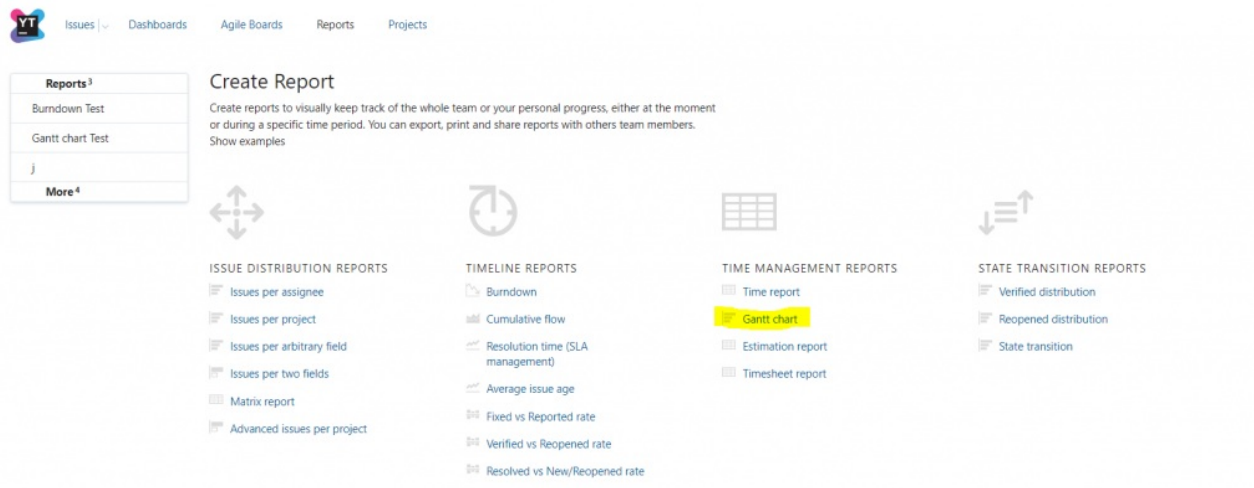


Here you can find our gitlab CI integration.

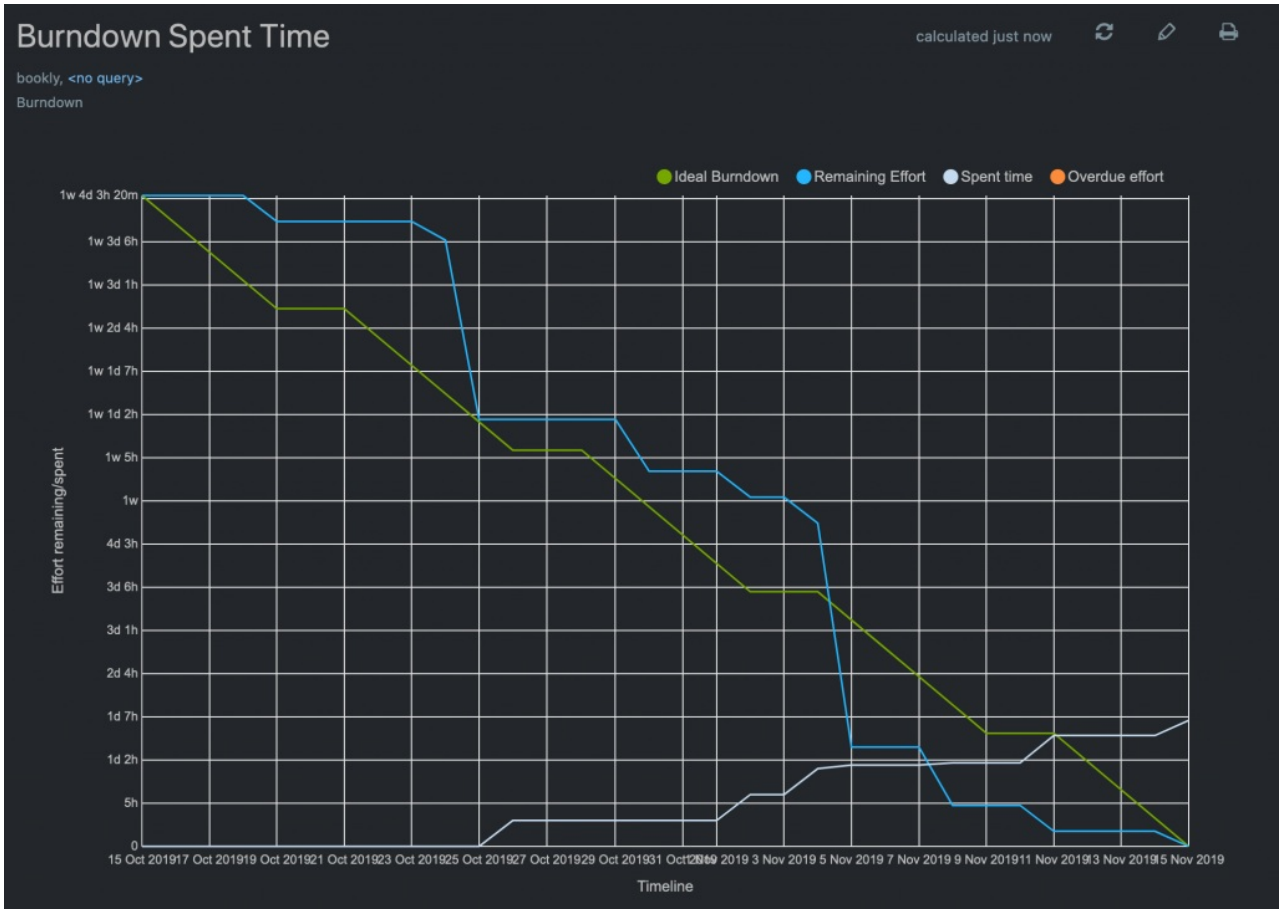
Due to the constraints from Youtrack we cannot provide a link to our public youtrack. Nevertheless, we will show you the correct tagging of issues with a screenshot. As you can see our tasks are assigned to someone, have an estimated time and are tagged with workflow and phase according to RUP.

~~BOOKLY-1~~	Translate SRS to MD file and upload to gitlab					× Inception	× Project Management	stober.alexandra	Nov 10
Normal	User Story	Done	stober.alexan...	Unscheduled	× 2	4h	3h		
~~BOOKLY-2~~	Insert Use Case Diagram into md					× Inception	× Project Management	stober.alexandra	Nov 10
Normal	User Story	Done	stober.alexan...	Unscheduled	× 1	15m	10m		
~~BOOKLY-3~~	Digitalize scribbles					× Design	× Inception	stober.alexandra	Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 2	4h	4h		
~~BOOKLY-4~~	setup system (VM)					× DevOps	× Inception	× Requirements	stober.alexandra Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 8	1d	1d		
~~BOOKLY-5~~	CI					× Configuration and Change Management	× DevOps	× Inception	stober.alexandra Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 3	2d	1d		
~~BOOKLY-7~~	setup SSL on our new VM/domain					× Configuration and Change Management	× DevOps	× Inception	JHelm Nov 10
Normal	User Story	Done	nico schinacher	Unscheduled	× 3	2h	30m		
~~BOOKLY-8~~	test Vue.js					× Environment	× Frontend	× Inception	JHelm Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 8	2d	1d		
~~BOOKLY-9~~	Testing and Embedding YouTrack Widgets					× Inception	× Project Management		JHelm Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 1	1h	2h		
~~BOOKLY-10~~	SSH Keys for Bookly-CI					× Configuration and Change Management	× DevOps	× Inception	JHelm Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 1	2h	30m		
~~BOOKLY-12~~	register our domain within the DNS for IPv6 (AAAA)					× DevOps	× Inception	× Project Management	JHelm Nov 10
Normal	User Story	Done	nico schinacher	Unscheduled	× 1	3h	1h		
~~BOOKLY-13~~	Get H2 up and running locally					× Backend	× DevOps	× Inception	× Requirements JHelm Nov 10
Normal	User Story	Done	JHelm	Unscheduled	× 3	1d	1d		
~~BOOKLY-15~~	Git Flow					× Inception	× Project Management		JHelm Nov 10
Normal	User Story	Done	nico schinacher	Unscheduled	× 3	15m	1h		

Youtrack can provide a GRANTT chart.



Update:
Our missing burndown chart with the spent time since 15/10/2019.



Update:
We were writing with the support of JetBrains and managed to get another free license and a guest account. So in short: Our board is finally public. You can find and access it here:
https://nicoschinacher.myjetbrains.com/youtrack/agiles

We were using tags for every RUP Workflow and RUP phase. But we managed to create fields and sets. They are visible by clicking on a task and looking on the right side.

Week 7: Retrospective

blog.bookly.online

jeanne November 14, 2019

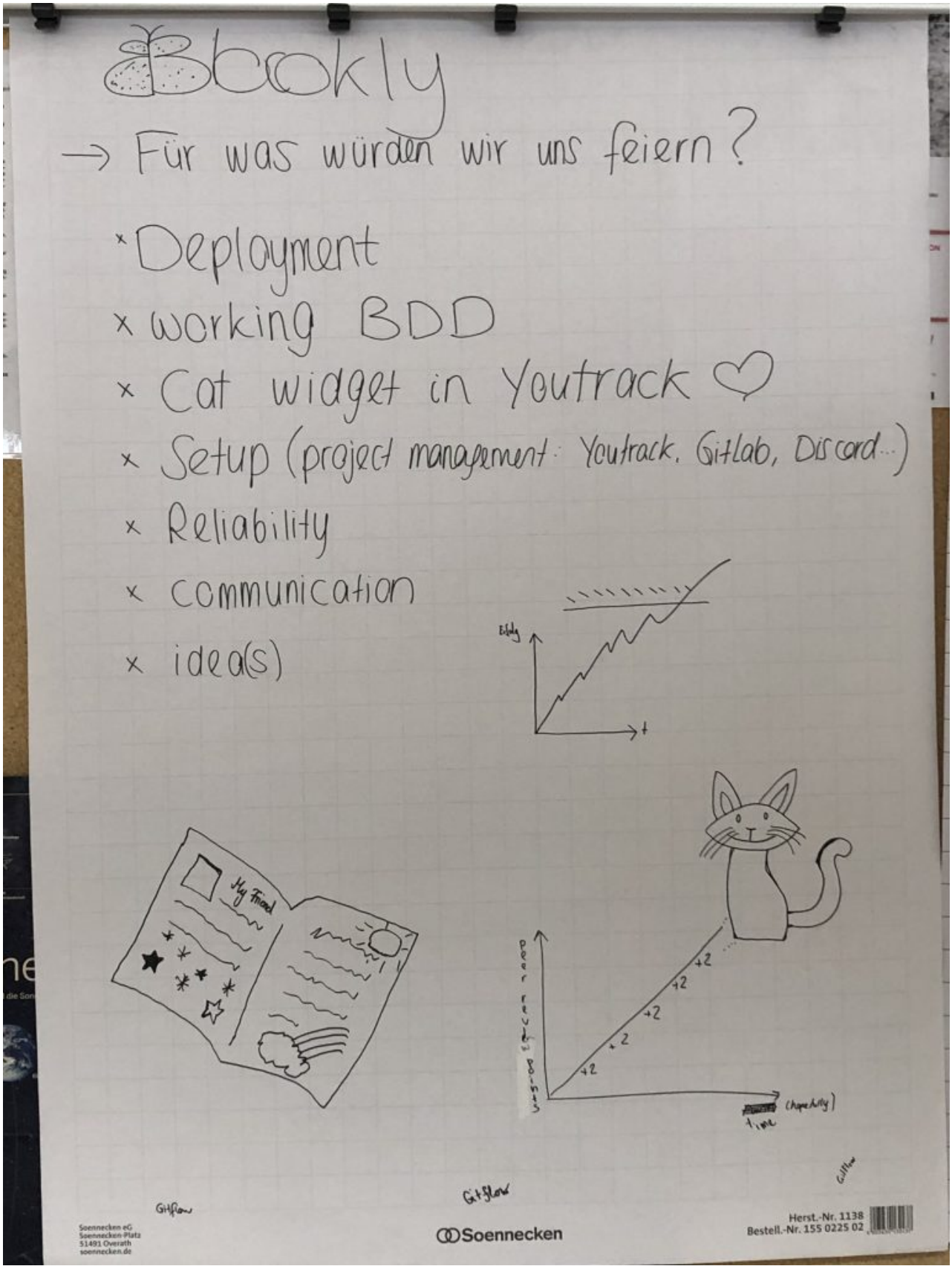
Hi everyone

Today the lecture was given by an external Scrummaster of DM. We did a retrospective together with our own teams. It turned out that we all agreed in what needs to be improved and finally want to start coding.

The retrospective started with how we felt about ourselves. Everyone should express this feeling individually with a means of transport of their choice. The feedback from the team members then began.

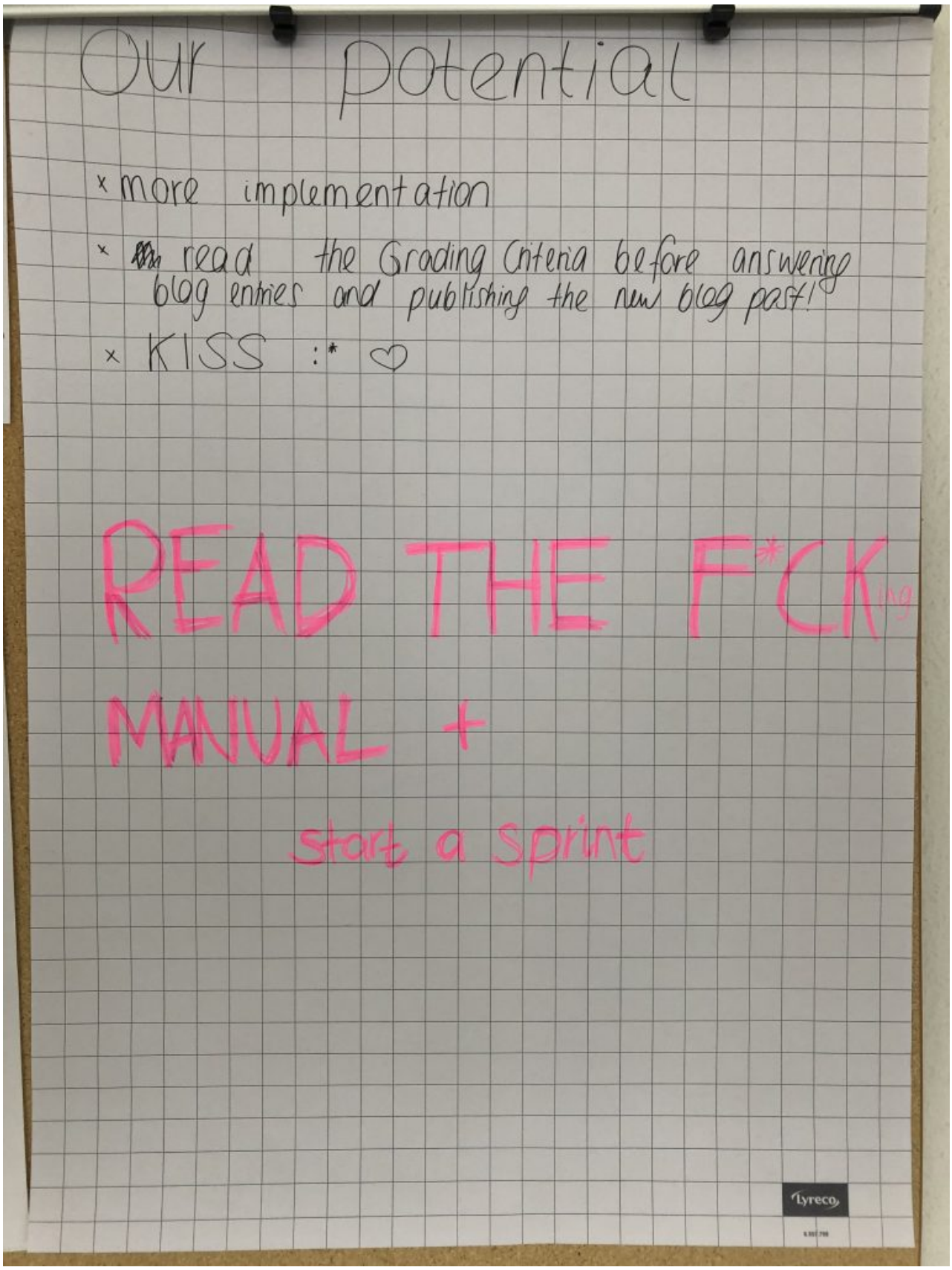
The following questions were thrown into the room one after the other and we each had 15 minutes to deal with them:

What would we celebrate for?



We are very proud of our automatic deployment, the already working BDD, the setup of Youtrack, GitLab and Discord and also the communication in our team. Almost every sunday we managed to publish a blog post and write two peer reviews while we worked on our project set up besides and wrote every expected documentation.

Where do we see our potentials? What do the measures look like?



On this picture we defined our potentials and our measures that we derive from them.

So we are still unsure how we can manage to implement the planned UCs until december as we haven't started with the real coding yet and in a few weeks their will be the first exams. We decided to start this week our first sprint and define fixed dates so we definitely start implementing something.

And for optimizing the writing of blog entries and peer reviews every team member should explicitly check the criteria again so that we do not have to duplicate work and that we can rely on each other completely.

Greetings,

bookly

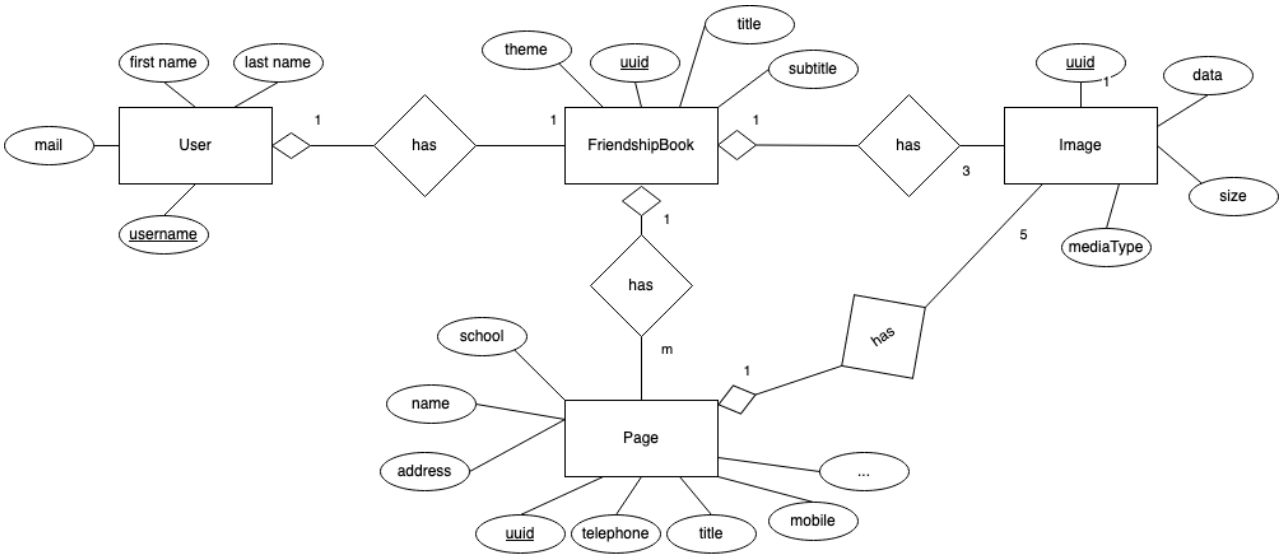
Week 8: Class Diagram

blog.bookly.online

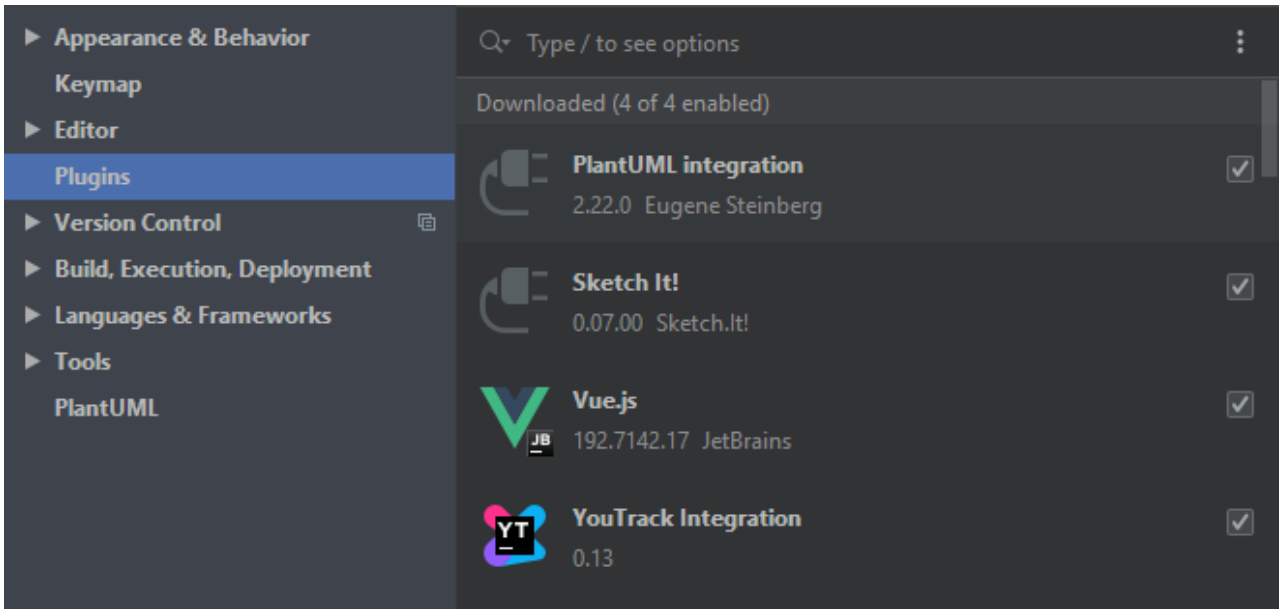
admin November 22, 2019

This weeks post is about UML, our project schema and some implementation we've already done!

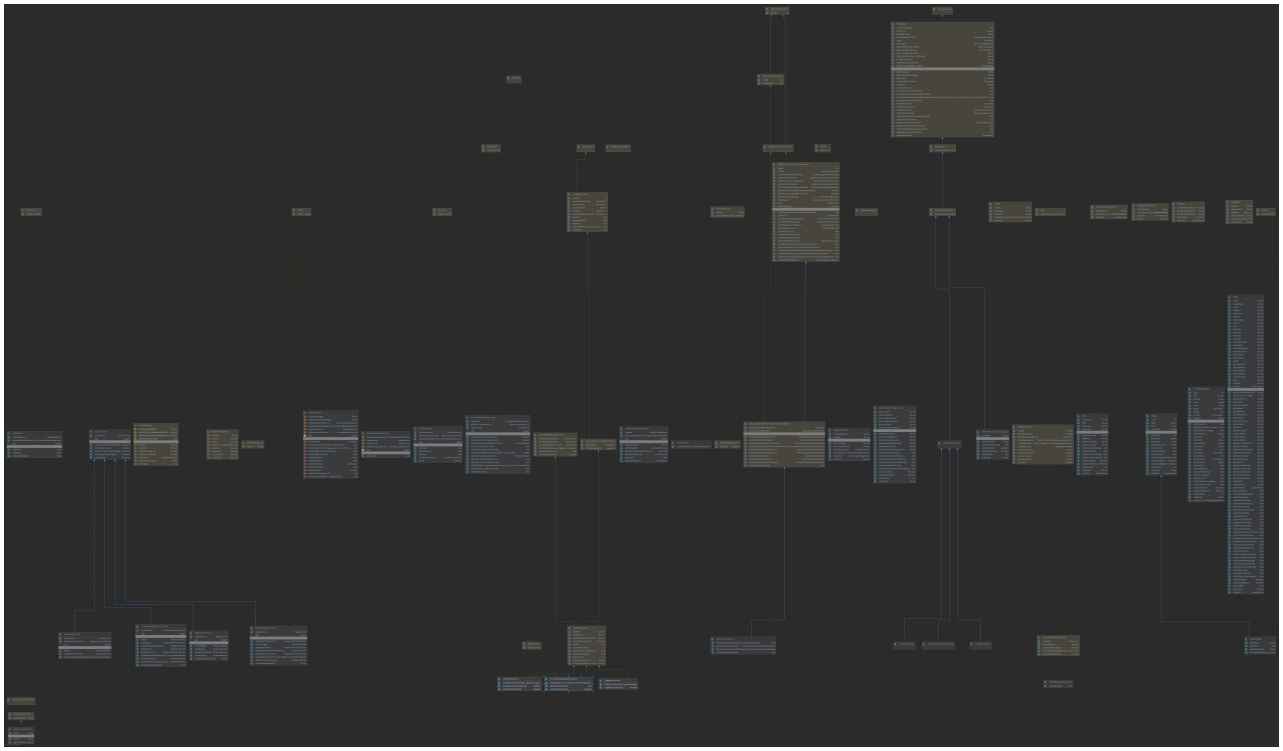
As we want to save user data and most importantly book entries, we decided to use a database. For developing local we are using H2, and in production SQLite. Here you can see our database schema as ER-model. The current version can be found here.



We use IntelliJ as our main IDE. This is why we have a wide range of plugins we can choose from. Here you can see a list of tools we installed. However because we use the Ultimate Edition Plugins like Git were already installed.

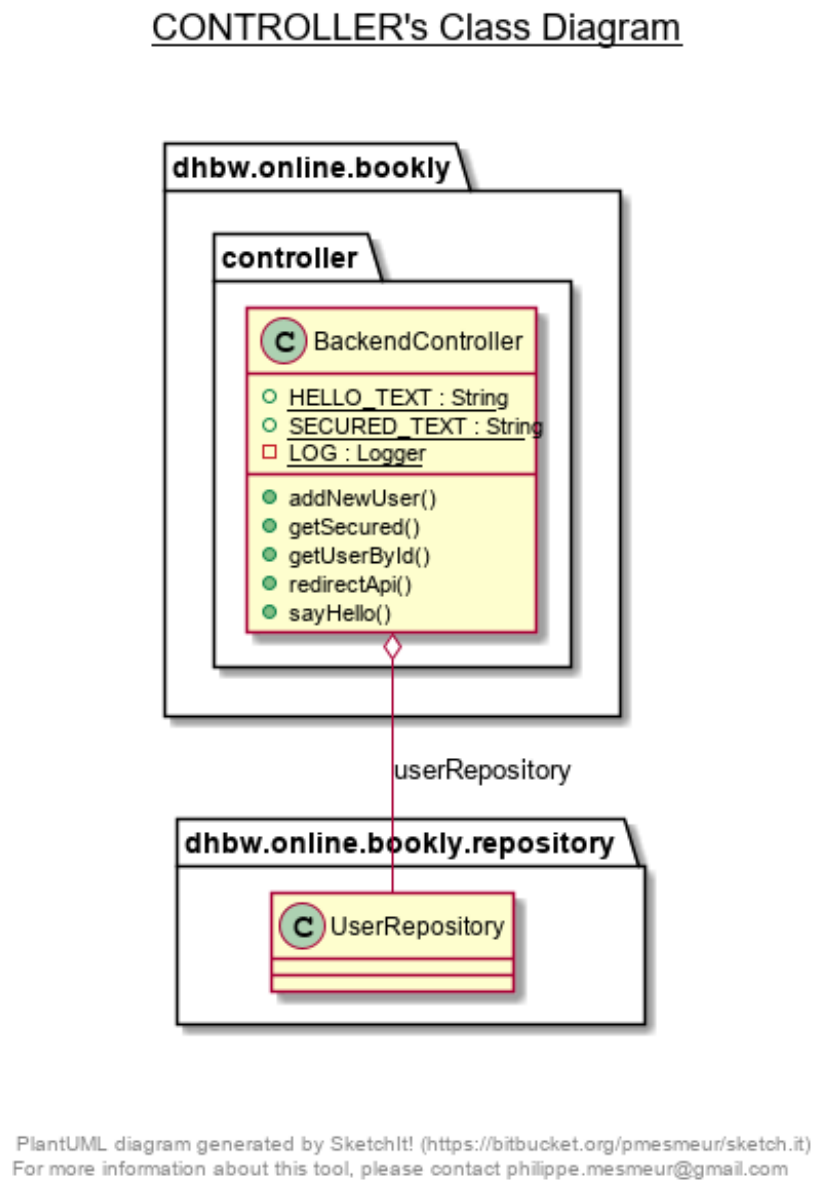


Using Sketch It! we can now generate fancy Class Diagrams! The most important class here is the BackendController because this class is the API which can be called by the frontend. You can see what methods this class has implemented (yet) in the following picture. As our Use Cases are mainly part of managing a book and a page, the most classes we need are REST controller, DTOs, Service-Classes and Repositories. We created most of them for generating the UML-Diagram but the most parts are not implemented yet. We are planing to realise them until december.
The current version can be found here.

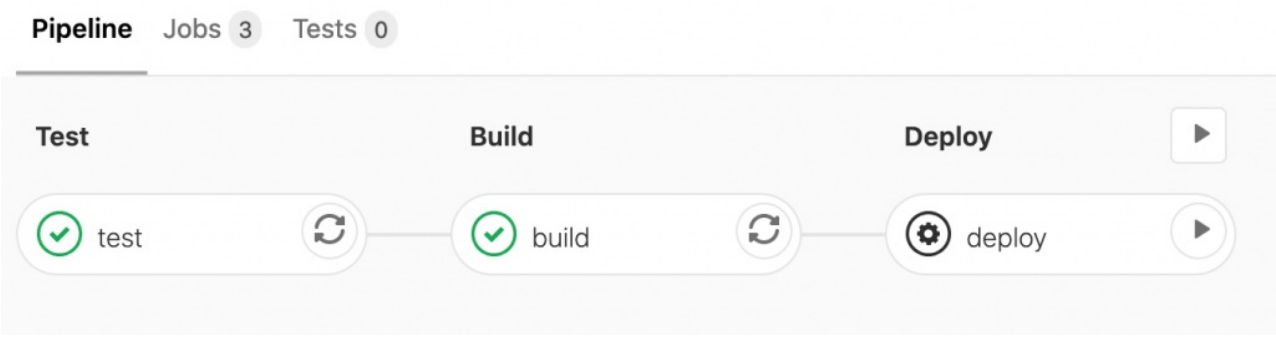


Here you can see the class diagram of the BackendController. It already has

functionalities like login and a simple implementation of a basic authentication and user api.



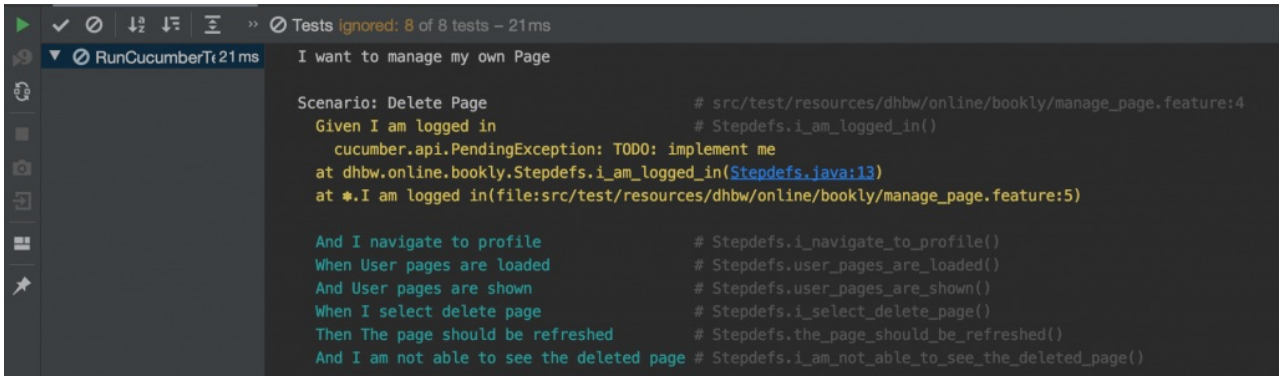
We proved already in one of our last blog entries that we are having automated testing and a working cucumber instance. The tests are running (local and automated by our CI after pushing to a feature branch) and will be implemented within the next sprints. In the following pictures you can see a pipeline and a test log output and also the local testing with IntelliJ.



Our current pipelines

```
304   [WARNING] Tests run: 8, Failures: 0, Errors: 0, Skipped: 8, Time elapsed: 0.547
    s - in dhbw.online.bookly.RunCucumberTest
305   [INFO]
306   [INFO] Results:
307   [INFO]
308   [WARNING] Tests run: 8, Failures: 0, Errors: 0, Skipped: 8
```

the log output of a test pipepline.



running a test in intelliJ

Implementation

In this week we also did a lot of implementation! We designed the landing page, added the login functionality by using basic authentication and a first simple API for a book and a user. Because our deployment works already so smoothly and automatic you can always check it out LIVE on bookly.online.

Week 9: Architecture

blog.bookly.online

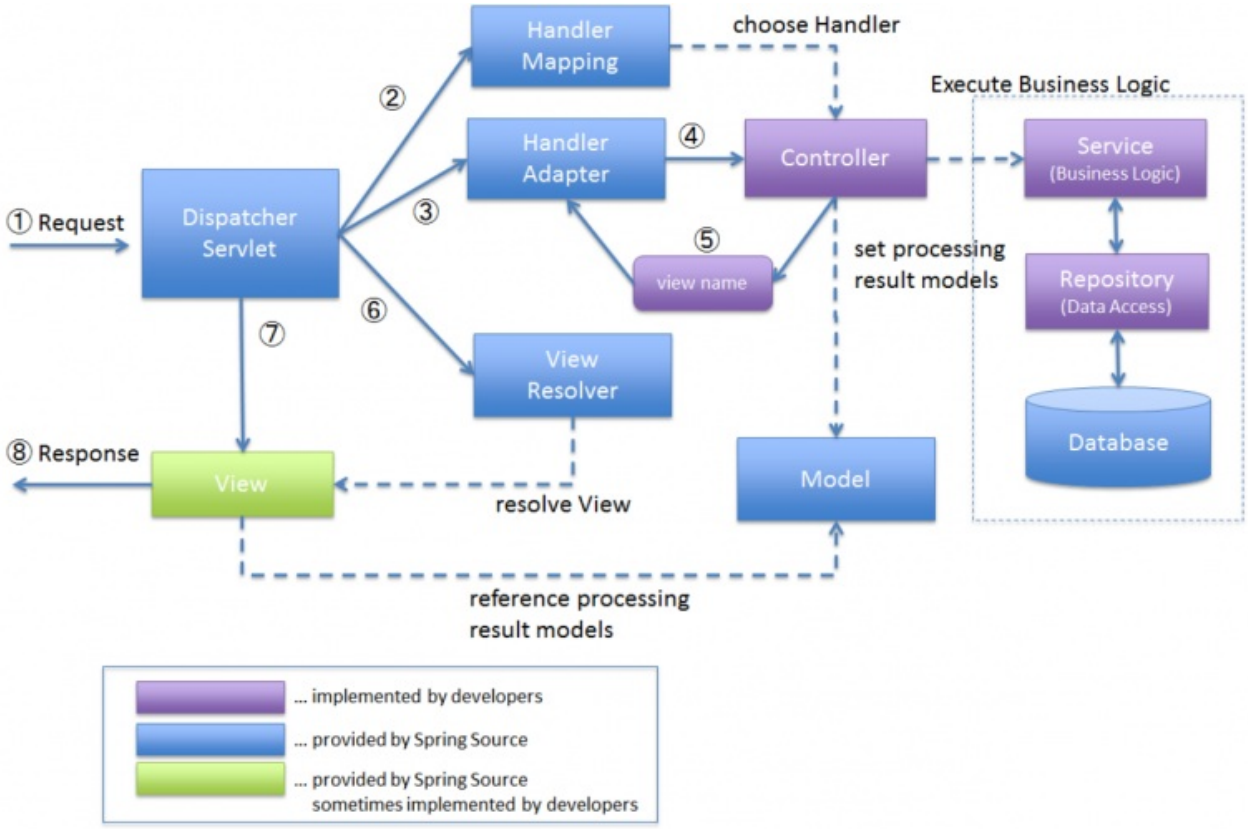
alex December 1, 2019

Hi there,

this week, we've been working on our Software Architecture Document which you can check out here. We are using Spring MVC as our main framework. This allows us to create an interactive and dynamic Web Application, that uses the Model-View-Controller logic and it offers the possibility of coding in Java.

A key difference between a traditional MVC controller and the RESTful web service controller in Spring is the way that the HTTP response body is created. Rather than relying on view technology to perform server-side rendering of the greeting data to HTML, this RESTful web service controller simply populates and returns an object. The object data will be written directly to the HTTP response as JSON.

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.



Within this document you can find:

- the class diagrams with marked MVC
- the ER diagram
- the deployment view

Week 10: Midterm

blog.bookly.online

jeanne December 5, 2019

Hey guys,

this week we want to show you, what we archieved this semester and sum up everything that we have done so far. If any link doesn't work please write to one of us!
The next blog post will be written next semester. Until then: Merry Christmas and guten Rutsch!

See you next year!

Team bookly

Our Git repository

Git repository

Our Use-Cases

As those two CRUDs are the main part of our functionality, we were allowed to split them into separate UCs and external documents where they are explained more well.

- UC Manage Book
    - Update -> not in scope and no individual UC
    - Delete -> in scope but no individual UC
- UC Manage Page

Unfortunately, this is one is not counted as UC, but we specified it anyway and invested just as much effort. The login and logout will be implemented this semester, the (un)registration within the next: UC Account

Our .features files

- Manage Book
- Manage Page

Software Specifications

- Software Requirement Specification
- Software Architecture Document
- Use-Case Diagram
- API Documentation with Swagger

Project Management

- Agile Board
- Reports (Burndown, Flow, Time spent per User etc.)
- Issue List

Midterm Presentation

Midterm presentation

All Blog Posts

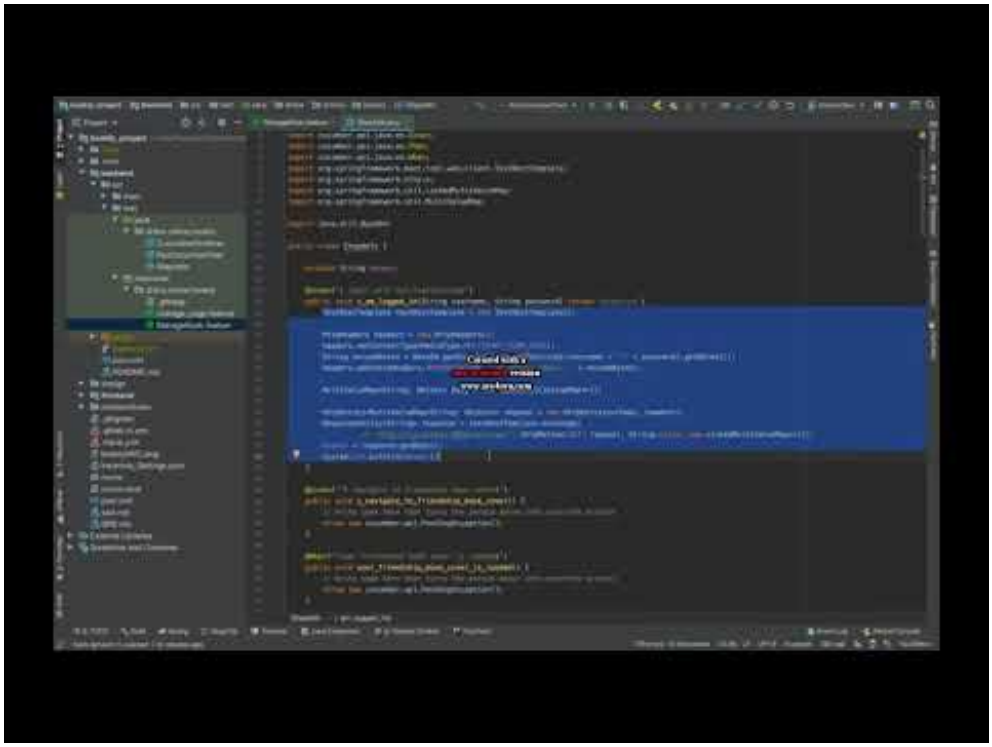
- HW 1: Vision
- HW 2: Roles and Technologies
- HW 3: SRS
- HW 4: Use-Cases
- HW 5: Testing
- HW 6: SCRUM
- HW 7: Retrospective
- HW 8: Class Diagramm
- HW 9: Architecture

Videos

Here you can see our deployment in action. Just watch the video bookly deployment.

Watch Video At: https://youtu.be/ZFZ7O4gUkOg

Our second video shows that our BDD is working. It is up and running with our first cucumber test.



Watch Video At: https://youtu.be/hBR4Yb2V8zU

That’s it for this year. Merry Christmas!

Week 12: We here. We back.

blog.bookly.online

admin April 27, 2020

Hello there,

After taking a break, we are back to rise to the top. We are back to conquer the world, to fulfill the dream of having a book we can look at 20 years later, just to look at our beloved friends and their dreams inside this book to rediscover our childhood.

We are **BACK** to finish what we've started.

Plans & Changes

There is still a lot of work ahead of us. This week we focused on getting back to the project, getting used to the workflow, finding out what we did the last time and we even tried to fix our sleep schedule.

Since last time, there have also been a lot of changes behind the curtain. We completely revamped our security system, thus making login and registration as safe as possible using *Keycloak*.

In addition, we moved our website to another server (don't worry, the name of the website still remains the same). On this server we also added more subdomains, most notably the **dev** subdomain on which we will deploy our current development status.

This decision has been made to provide a efficient and proactive workflow.

Keycloak – more security!

During the last weeks before the theory phase we decided to outsource the identity management to a separate server. For this we use Keycloak, which is available online at keycloak.bookly.online. It allows us to centrally manage user data, easy extensibility of login and registration forms, easy integration of social identity providers, and better security than if we had to implement it ourselves. For this purpose, a redirect to our keycloak server is performed when accessing protected pages. After authentication, our clients are issued user information and a token. The token can be used for further API requests. The integration was unfortunately a bit time consuming, but now we are excited about Keycloaks features. You can find more information here: keycloak.org or simply write a comment with questions. We are looking forward to any technological discussion.

Server changes!

As web server for all (sub-) domains we use an Apache2. Our CI was of course also changed. Currently, we start simple JARs, which are started by a system service. In the next week we will move to Docker Container and adjust the system services.

Here are the our updated important links to our website:

- dev.bookly.online
- keycloak.bookly.online
- bookly.online

Documentation

This week we also worked on documentation and design patterns. We finished our use cases! The picture shows our overall use case diagramm. You can find the documents listed below:

Overall Use Case Diagramm

Use Cases for the second semester

- UC 7: Share link, visibility & invite
- UC 8: Manage Page Decorations
- UC 9: Manage Cover Decorations
- UC 10: NavigateFooter

- UC 11: Navigate Header
- UC 12: Operate Account

Riskmanagement

To be prepared for any potential risks we've done some risk management to be aware of unfortunate incidents which might occur. You can see a screenshot below as well as find it on GoogleSpreadsheats.

		1 to 100	1 to 10			
Risk Name	Risk Description	Risk Probability of Occurrence	Risk Impact	Risk Factor	Risk Mitigation	Person in Charge of Tracking
Time	not enough time to fullfill every task	50%	9	4,5	finish work ahead of time for the next week; keep up to date with team members	Everyone
Technical difficulties	difficult bugs or laptop gets lost	65%	6	3,9	back up laptop/ code; get support from team members; look for alternative implementation	Everyone
Code quality	bad coding slows or breaks the application	30%	4	1,2	Code reviews; Ide code checks; Testing	Nico
Internet Provider	no internet connection due to unitymedia	25%	3	0,75	conntact Internet Provider to fix issue	Alexandra
Corona	2 Weeks of sick leave because of quarantine	10%	2	0,2	stay at home; team picks up tasks of sick person	Everyone
Hacked Server	server gets hacked by third party	9%	6	0,54	HTTPS; secure passwords	Jeanne
Database Corruption	Database gets accessed by third party	5%	4	0,2	Secured by strong password; setup new database if data gets corrupted	Alexandra

Time Spent on UCs

To summarize the work we've done so far and the time spent, we created a time table, which should give you a good overview. You can see a screenshot below as well as find it on GoogelSpreadsheats.

Week 13 Function Points and estimation of UCs

blog.bookly.online

alex May 1, 2020

Function Points

Our goal for this week was to estimate the hours we have to spent for the remaining use cases we are planning to implement this semester. This allows us to schedule the use cases in the remaining two and a half months.

To achieve an estimation as accurate as possible, we calculated function points for each use case (those which are already implemented and those which are planned for this semester). To do so, we used the website Tiny Tools.

At first, we filled the second table regarding our whole project. You can find our input here:

Complexity Adjustment Table

ITEM	COMPLEXITY ADJUSTMENT QUESTIONS	SCALE (No Influence 0 1 2 3 4 5 Essential)
1	Does the system require reliable backup and recovery?	1
2	Are data communications required?	5
3	Are there distributed processing functions?	1
4	Is performance critical?	0
5	Will the system run in an existing, heavily utilized operational environment?	0
6	Does the system require on-line data entry?	5
7	Does the on-line data entry require the input transaction to be built over multiple screens or operations?	5
8	Are the master files updated on-line?	5
9	Are the inputs, outputs, files or inquiries complex?	3
10	Is the internal processing complex?	2
11	Is the code to be designed reusable?	2
12	Are conversion and installation included in the design?	0
13	Is the system designed for multiple installations in different organizations?	0
14	Is the application designed to facilitate change and ease of use by the user?	0

Domain Characteristic Table | FP Calculation

The next step was the Domain Characteristic Table. This one, we filled for each use case differently. But first, we put some thought into our project and the information needed for the calculation:

UC1 Create Book	**RET**	**DET**	**FTR**	**Resulting Complexity**	**Count**
External Inputs		0	2	average	2
External Outputs		0	0	low	0
External Queries		0	0	low	0
Internal Logical Files	0	1		low	1
External Interface Files	0	0		low	0
Function Points	***14.1***				

UC2 Read Book	**RET**	**DET**	**FTR**	**Resulting Complexity**	**Count**
External Inputs		0	0	low	0
External Outputs		25	0	complex	25
External Queries		1	2	average	3

Internal Logical Files	0	1		average	1
External Interface Files	0	0		low	0
Function Points	***185.2***				

UC3 Update Book	**RET**	**DET**	**FTR**	**Resulting Complexity**	**Count**
External Inputs		2	1	average	3
External Outputs		0	0	low	0
External Queries		1	2	low	3
Internal Logical Files	1	0		low	1
External Interface Files	0	0		low	0
Function Points	***26.3***				

UC4 Create Page	**RET**	**DET**	**FTR**	**Resulting Complexity**	**Count**
External Inputs		0	0	low	0
External Outputs		0	0	low	0
External Queries		1	3	average	4
Internal Logical Files	1	0		low	1
External Interface Files	0	0		low	0
Function Points	***21.6***				

UC5 Read Page	**RET**	**DET**	**FTR**	**Resulting Complexity**	**Count**
External Inputs		0	0	low	0
External Outputs		25	3	complex	28
External Queries		2	3	complex	5
Internal Logical Files	6	0		average	6
External Interface Files	0	0		low	0
Function Points	***259.4***				

UC6 Update Page	RET	DET	FTR	Resulting Complexity	Count
External Inputs		20	3	complex	23
External Outputs		0	0	low	0
External Queries		1	1	low	2
Internal Logical Files	6	0		average	6
External Interface Files	0	0		low	0
Function Points	***191.8***				

UC7 Share link	RET	DET	FTR	Resulting Complexity	Count
External Inputs		3	3	average	6
External Outputs		1	2	low	3
External Queries		2	3	average	5
Internal Logical Files	0	0		low	0
External Interface Files	0	0		low	0
Function Points	***52.6***				

UC8 Manage Page Decorations	RET	DET	FTR	Resulting Complexity	Count
External Inputs		5	3	complex	8
External Outputs		5	3	complex	8
External Queries		3	3	average	6
Internal Logical Files	6	0		average	6
External Interface Files	0	0		low	0
Function Points	***176.7***				

UC9 Manage Cover Decorations	RET	DET	FTR	Resulting Complexity	Count
External Inputs		5	2	average	7

External Outputs		6	2	complex	8
External Queries		2	2	low	4
Internal Logical Files	4	0		average	4
External Interface Files	0	0		low	0
Function Points	***127.8***				

Find the original table with more details here

Our already implemented use cases are:

- UC 1: Create Book
- UC 2: Read Book
- UC 3: Update Book
- UC 4: Create Book Entry (Page)
- UC 5: Read Book Entry (Page)
- UC 6: Update Book Entry (Page)

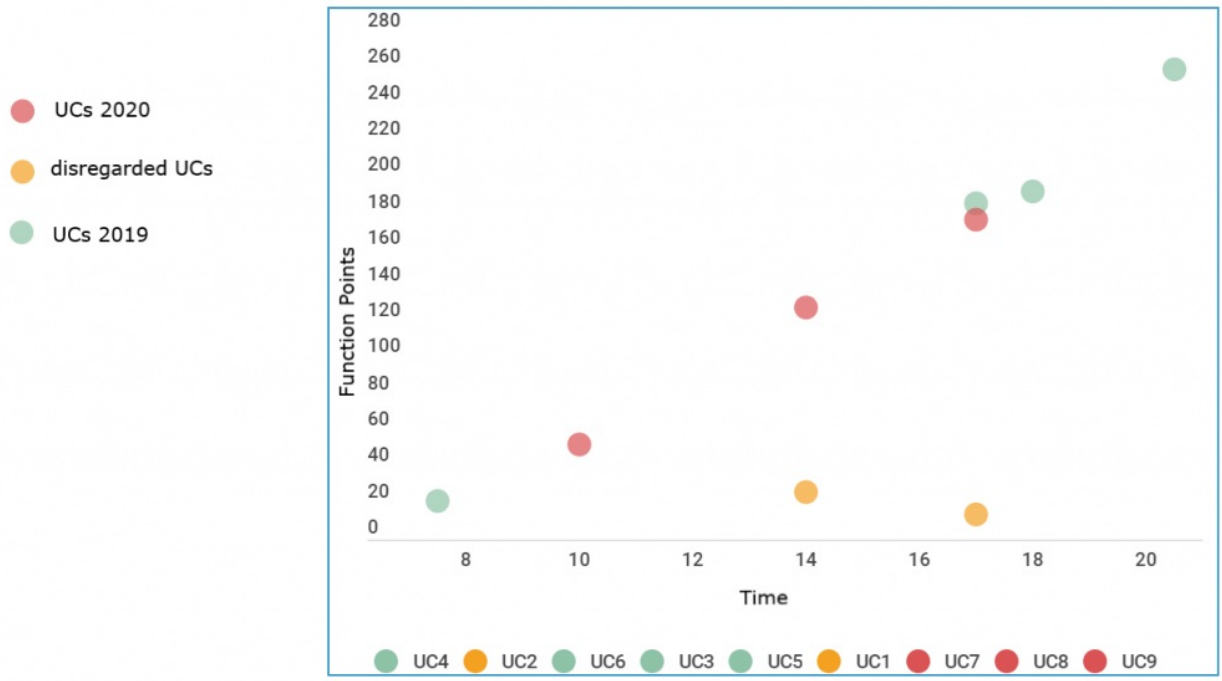
Our use cases that are planned for this semester:

- UC 7: Share link, visibility & invite
- UC 8: Manage Page Decorations
- UC 9: Manage Cover Decorations
- UC 10: NavigateFooter
- UC 11: Navigate Header
- UC 12: Operate Account

Furthermore, we used the function points and the spent time for the already implemented use cases to draw the following diagram (click to enlarge):

Use Cases over spent time and FPs

In the subsequent diagram you can see the time we spent on use cases in 2019 . The green and yellow dots represent these use cases. The yellow use cases are our first and second use case. We used them to get our feet wet. They are outliers and were both disregarded in the estimation of our new use cases for 2020. In order to determine the time we will need to implement the new use cases only the green colored use case were taken into account.



According to this graph the implementation of our use cases will need approximately 41 hours.

Estimation of Use Cases for 2020	**Estimated Time**
UC7: Share link, visibility & invite	10
UC8: Manage Page Decorations	17
UC9: Manage Cover Decorations	14
Total	***41***

This implementation estimate does not include tests. While we have already set up SonarQube to get an idea of our test coverage, only rudimentary tests have been implemented. Thus, we will still need more time to implement tests for our new and old use cases. In accordance with our cumulative flow diagram from the last semester we will need approximately another 150 hours.

Thanks for reading and hopefully you will continue with us on our journey!

Week 14: JUnit

blog.bookly.online

jeanne May 7, 2020

Hi everyone

We created our Testplan! You can find it here: Testplan.md
We decided to do 3 different kind of tests:

- Function Testing: Cucumber, JUnit
- Database Integrity Test: JUnit + Mockito
- Field Test

Our goal is to achieve 20% test coverage. Our start date for implementing tests is the 13.5.2020.

Inline-Update: We integrated the visibility of our tests in gitlab. See our live coverage badge here:

coverage 29.00%

test coverage on master-branch

Function Testing

We already implemented BDD tests for the use cases of the first semester. You can find our implemented step definition for Cucumber here. The corresponding feature files are here. For the Java Backend we want to implement JUnit Tests.

Database Integrity Test

To ensure a correct data processing for creating, reading, updating and deleting, we want to implement JUnit Tests in combination with the Framework Mockito for our backend API.

Field Test

We are especially excited to plan the field test. As we are implementing an app for kids, we want to do the test session with children. We can imagine that children with an age between 10 to 12 fit perfect into our user group. We are in contact with a teacher that would help us with her school class to test our app.

But first we want to realize a few more use cases and clean up our styling. Then we will plan, what kind of remote tool we will use for the test session, what questions we want to ask and how we prepare the system for them. We will update you as soon as we planned more

Gitlab-CI: Test Stage

Since last semester we have a working CI for testing, building and deploying our application. You can find the corresponding configuration file here.

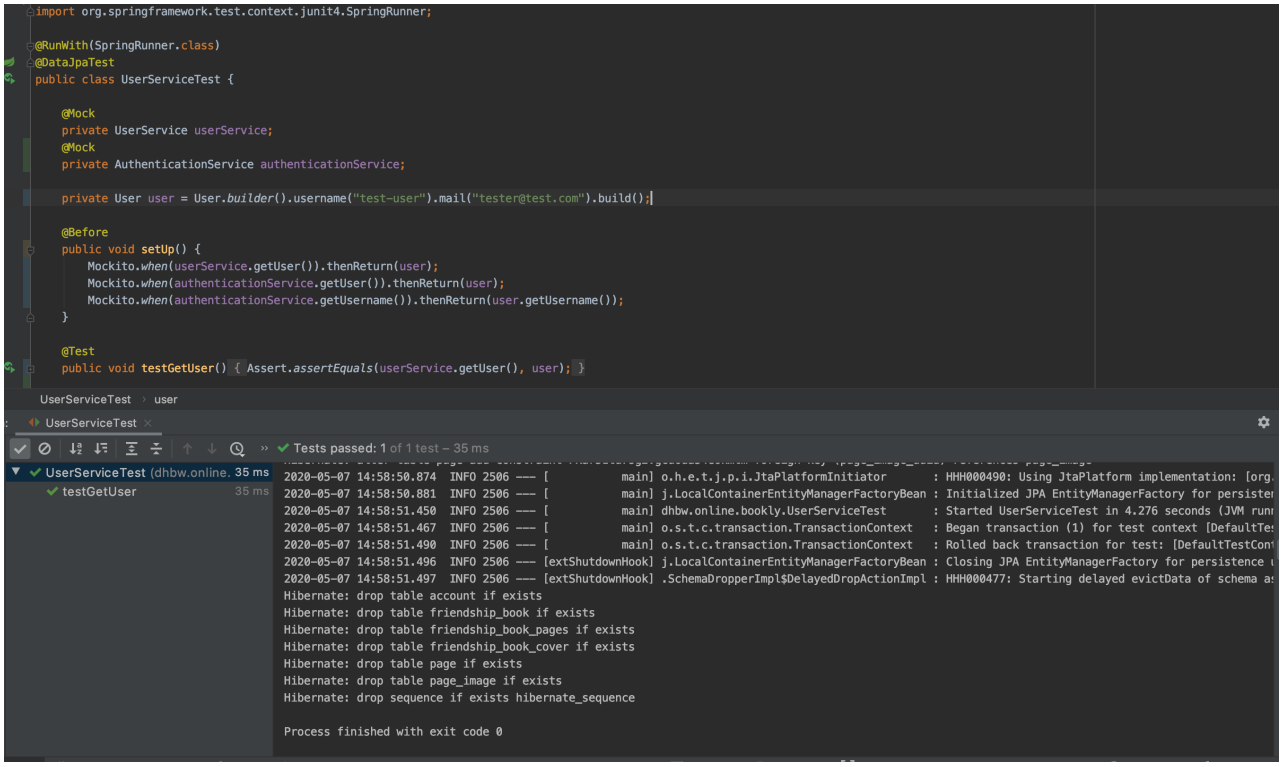
If you want to see the logging of any test, just go to our pipelines and click on a test stage you're interested in. Here's an example:
https://gitlab.com/project_bookly/bookly/-/jobs/542554917

Last semester we already filmed our first runnable cucumber tests. You can watch it here

https://youtu.be/hBR4Yb2V8zU

Proof of working JUnit Tests

As you can see we just implemented a mocked test for our UserService and our AuthenticationService. We will soon have more tests to run We used here the Junit4 SpringRunner of Spring's JUnit Integration too, to start the application context and autowire our services later on.



Proof of working JUnit Tests

We added the dependencies for JUnit and Mockito to the Maven-POM of our backend. You can find the complete document here: https://gitlab.com/project_bookly/bookly/-/blob/master/backend/pom.xml

```
<!-- Testing -->
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.13</version>
         <scope>test</scope>
      </dependency>
      <dependency>
         <groupId>org.mockito</groupId>
         <artifactId>mockito-all</artifactId>
         <version>1.10.19</version>
         <scope>test</scope>
      </dependency>
      .....
```

Automatic Test Data

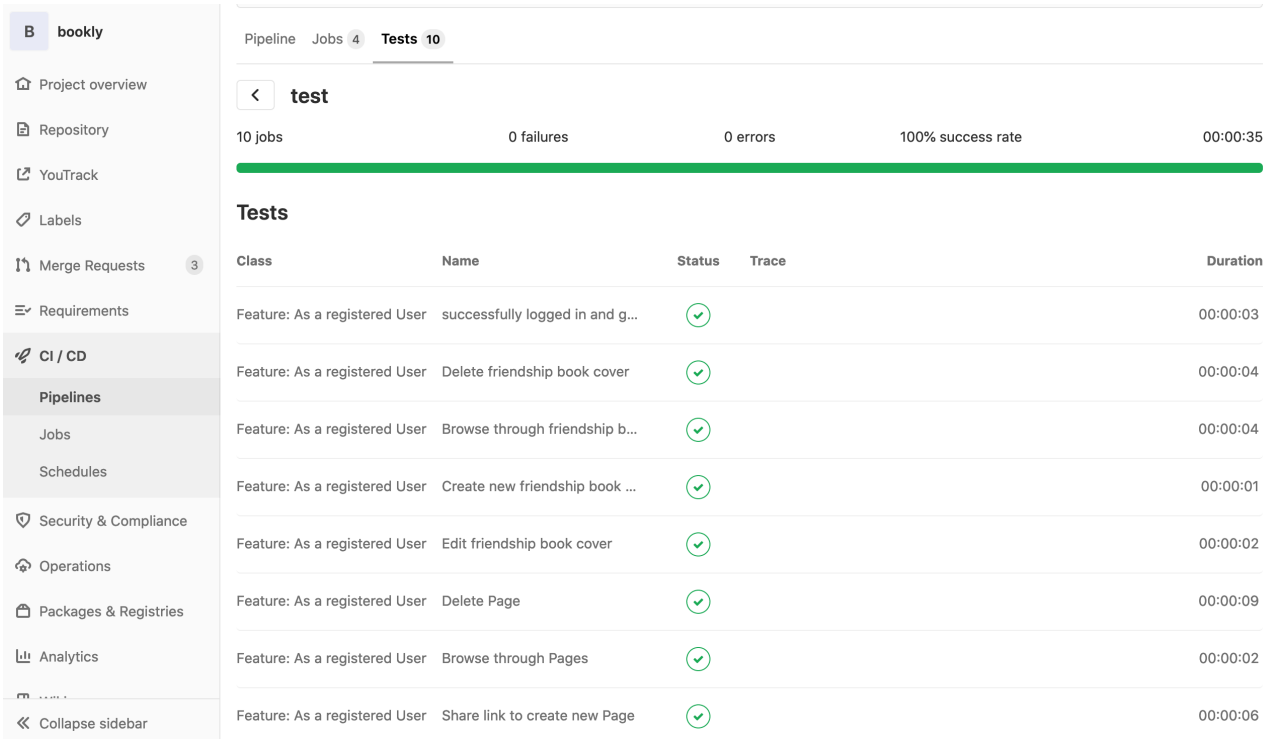
For our local deployment we already created some test data. It was necessary as our local H2 is not saving any data while running and is empty after a restart. So we implemented a Component that initialize the H2 at the beginning. We are using the same data for our test environment. You can look inside of the file here: https://gitlab.com/project_bookly/bookly/-/blob/master/backend/src/test/java/dhbw/online/bookly/TestData.java

As we are using the JUnit4 Spring Runner, the class is recognized as Component and automatically creates the data we need.

Gitlab: Test Coverage

We want to integrate the Maven Surefire Plugin for displaying the test coverage within Gitlab. After we implemented and integrated it into our project, we will update this blog post here

Update:
We integrated Surefire and JaCoCo into our project. We are now able to see a more detailed view of our passed/failed unit and cucumber tests by clicking to a specific pipeline.



Example of tests within a pipeline, generated by surefire

For the whole test coverage we created a badge for our master branch. The coverage is calculated by JaCoCo for our backend. We are extracting the coverage value from its result and are handing it over to Gitlab. The value is refreshed after a successful test stage and is visibile within the test pipeline or you can see our current test state here live:

test coverage on master-branch

Deployment

As you can see in our .gitlab-ci.yml we are deploying manually. Even though we have automated testing, we prefer to deploy our system by clicking a button because it gives us immense satisfaction and happiness.

Week 15 – Refactoring

blog.bookly.online

admin May 16, 2020

Hey guys,
this week we published some major updates to our main site. Moreover our task was to work through the first chapter of Refactoring: Improving the Design of Existing Code.

Updates

You can check the main updates on our site bookly.online. Most notably we added profile, book and page management, as well as fixed some bugs. We also added the main workflow of how a friend shall be able to access the page he/she is supposed to edit.
The main idea is to generate an invite code which can then be handed over to a friend. Then, this friend just goes to the main website and enters that code to directly edit the page. Simplicity is key because we don't want the user to run into any problems!

Fowler's Refactoring

Besides the development of our site we also worked through the first chapter of "Refactoring: Improving the Design of Existing Code".

Each of our group member used their own GitLab Project:

- Jeanne
- Alex
- Nico

You can check the commits over here:

- Jeanne
- Alex
- Nico

We appreciate any comments
Best regards
Team bookly

Week 16 – Design Patterns

blog.bookly.online

alex May 22, 2020

This week we looked at software patterns. We should integrate one of them into our code. Patterns like MVC, Observer and Singleton were not allowed because most frameworks offer them “for free”.

We decided to implement the ***SOFA*** principle for our FriendshipBookService. It expects short methods with few argument and one layer of abstraction. Each method is suppose to do one thing.

This principle caused great controversy in our group. On the one hand it doesn’t seem worth the effort because the project is too small to really call for a design pattern. In a small project the extraction of functionality does not improve the code readability a great deal. It might even hinder the reader of our code because the functionality of each method has to be determined and you have to jump from place to place in code. (Even if you can do so easily with the help of your IDE, it’s annoying.) We tried to make this the least painful possible by using “meaningful method names” that make their functionality clear. On the other hand, the pattern helps in a large project to get a quicker understanding of the code. It forces coders to make a habit of thinking about the various functionalities that they are using and structuring them in a coherent manner. Our consensus is that as long as you don’t overuse it. It is a useful tool. We will keep the pattern in our code and see if we will be able to use it more meaningfully in the future.

Update

It was pointed out that SOFA is a design principle and not a design pattern, so there was a slight misunderstanding on our side. 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, which was of interest because of the metrics topic.

Here you can see our code:

Before:

```
public class FriendshipBook {

  ...

    @JsonIgnore
    @OneToOne(cascade = {CascadeType.ALL})
    @ApiModelProperty(notes = "the cover image of the book",
            position = 3)
    private FriendshipBookCover cover;

    @OneToOne
    @ApiModelProperty(notes = "the user of which the book belongs to. is extracted from the
authentication",
            position = 4)
    private User user;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, targetEntity =
Page.class)
    @ApiModelProperty(notes = "list of all pages of the book",
            position = 5)
    private List<Page> pages = new ArrayList<>();

    @ApiModelProperty(notes = "ID of the Theme template",
            position = 6)
    private int theme;

    @JsonIgnore
    @OneToOne(cascade = {CascadeType.ALL})
    @ApiModelProperty(notes = "the first sticker of the book",
            position = 7)
    private FriendshipBookSticker sticker1;

    @JsonIgnore
    @OneToOne(cascade = {CascadeType.ALL})
    @ApiModelProperty(notes = "the second sticker of the book",
            position = 8)
    private FriendshipBookSticker sticker2;
```

After:

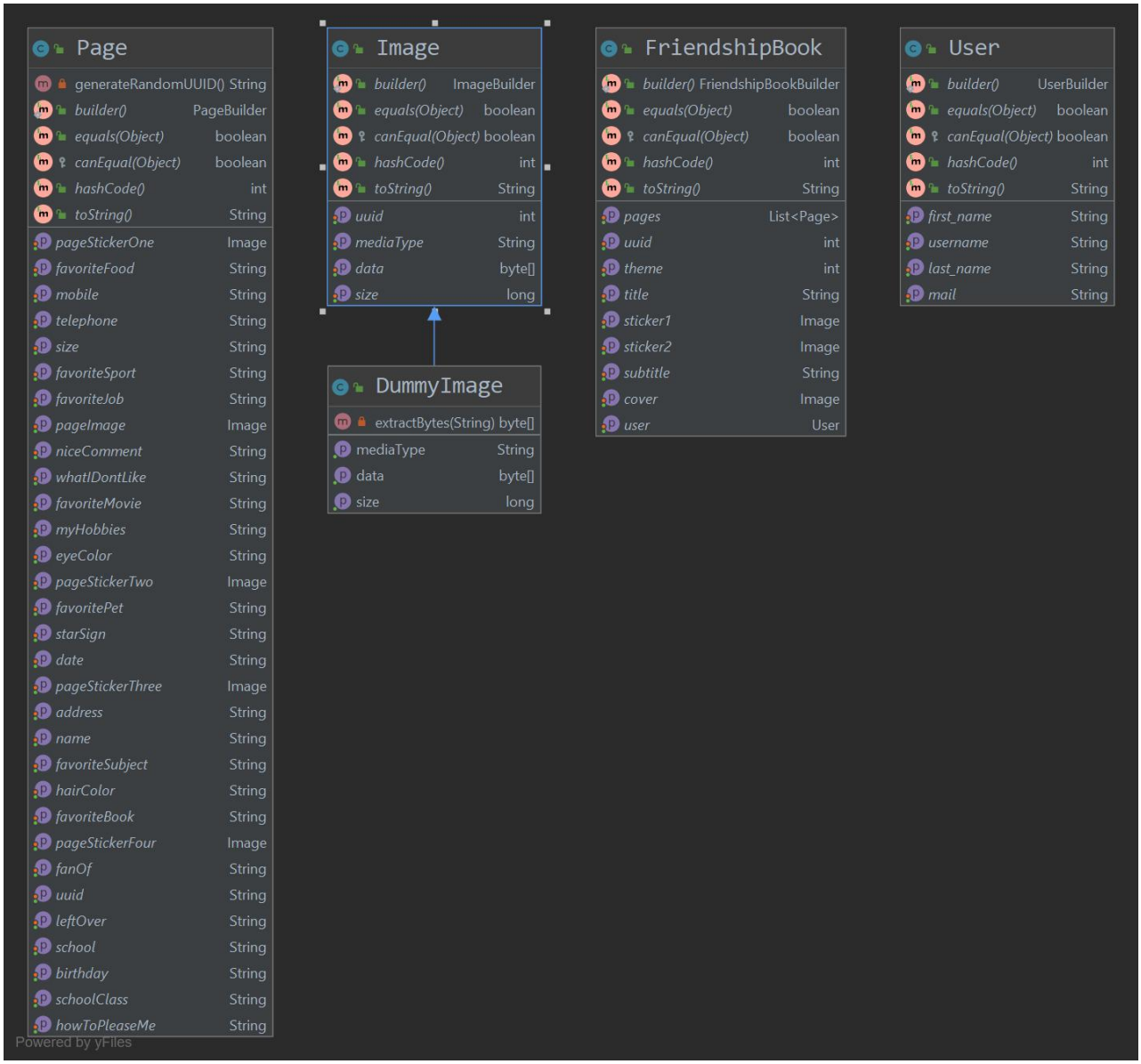
```
public class FriendshipBook {

...

    @JsonIgnore
    @OneToOne(cascade = { CascadeType.ALL })
    @ApiModelProperty(notes = "the cover image of the book", position = 3)
    private Image cover = new DummyImage();

    @OneToOne
    @ApiModelProperty(notes = "the user of which the book belongs to. is extracted from the
authentication", position = 4)
    private User user;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, targetEntity =
Page.class)
    @ApiModelProperty(notes = "list of all pages of the book", position = 5)
    private List<Page> pages = new ArrayList<>();

    @ApiModelProperty(notes = "ID of the Theme template", position = 6)
    private int theme;

    @JsonIgnore
    @OneToOne(cascade = { CascadeType.ALL })
    @ApiModelProperty(notes = "the first sticker of the book", position = 7)
    private Image sticker1 = new DummyImage();

    @JsonIgnore
    @OneToOne(cascade = { CascadeType.ALL })
    @ApiModelProperty(notes = "the second sticker of the book", position = 8)
    private Image sticker2 = new DummyImage();
```

Class diagram

Before


Page
uuid String
pageImage PageImage
name String
address String
telephone String
mobile String
schoolClass String
school String
size String
hairColor String
eyeColor String
birthday String
starSign String
favoriteSubject String
favoritePet String
howToPleaseMe String
whatIDontLike String
favoriteJob String
myHobbies String
fanOf String
favoriteMovie String
favoriteSport String
favoriteBook String
favoriteFood String
niceComment String
date String
leftOver String
pageStickerOne PageSticker
pageStickerTwo PageSticker
pageStickerThree PageSticker
pageStickerFour PageSticker
generateRandomUUID() String
builder() PageBuilder
getUuid() String
getPageImage() PageImage
getName() String
getAddress() String
getTelephone() String
getMobile() String
getSchoolClass() String
getSchool() String
getSize() String
getHairColor() String
getEyeColor() String
getBirthday() String
getStarSign() String
getFavoriteSubject() String
getFavoritePet() String
getHowToPleaseMe() String
getWhatIDontLike() String
getFavoriteJob() String
getMyHobbies() String
getFanOf() String
getFavoriteMovie() String
getFavoriteSport() String
getFavoriteBook() String
getFavoriteFood() String
getNiceComment() String
getDate() String
getLeftOver() String
getPageStickerOne() PageSticker
getPageStickerTwo() PageSticker
getPageStickerThree() PageSticker
getPageStickerFour() PageSticker
setUuid(String) void
setPageImage(PageImage) void
setName(String) void
setAddress(String) void
setTelephone(String) void
setMobile(String) void
setSchoolClass(String) void
setSchool(String) void
setSize(String) void
setHairColor(String) void
setEyeColor(String) void
setBirthday(String) void
setStarSign(String) void
setFavoriteSubject(String) void
setFavoritePet(String) void
setHowToPleaseMe(String) void
setWhatIDontLike(String) void
setFavoriteJob(String) void
setMyHobbies(String) void
setFanOf(String) void
setFavoriteMovie(String) void
setFavoriteSport(String) void
setFavoriteBook(String) void
setFavoriteFood(String) void
setNiceComment(String) void
setDate(String) void
setLeftOver(String) void
setPageStickerOne(PageSticker) void
setPageStickerTwo(PageSticker) void
setPageStickerThree(PageSticker) void
setPageStickerFour(PageSticker) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
FriendshipBook
uuid int
title String
subtitle String
cover FriendshipBookCover
user User
pages List<Page>
theme int
sticker1 FriendshipBookSticker
sticker2 FriendshipBookSticker
setTheme(int) void
setPages(ArrayList<Page>) void
builder() FriendshipBookBuilder
getUuid() int
getTitle() String
getSubtitle() String
getCover() FriendshipBookCover
getUser() User
getPages() List<Page>
getTheme() int
getSticker1() FriendshipBookSticker
getSticker2() FriendshipBookSticker
setUuid(int) void
setTitle(String) void
setSubtitle(String) void
setCover(FriendshipBookCover) void
setUser(User) void
setSticker1(FriendshipBookSticker) void
setSticker2(FriendshipBookSticker) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
FriendshipBookSticker
uuid int
data byte[]
size long
mediaType String
builder() FriendshipBookStickerBuilder
getUuid() int
getData() byte[]
getSize() long
getMediaType() String
setUuid(int) void
setData(byte[]) void
setSize(long) void
setMediaType(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
FriendshipBookCover
uuid int
data byte[]
size long
mediaType String
builder() FriendshipBookCoverBuilder
getUuid() int
getData() byte[]
getSize() long
getMediaType() String
setUuid(int) void
setData(byte[]) void
setSize(long) void
setMediaType(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
PageSticker
uuid int
data byte[]
size long
mediaType String
builder() PageStickerBuilder
getUuid() int
getData() byte[]
getSize() long
getMediaType() String
setUuid(int) void
setData(byte[]) void
setSize(long) void
setMediaType(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
PageImage
uuid int
data byte[]
size long
mediaType String
builder() PageImageBuilder
getUuid() int
getData() byte[]
getSize() long
getMediaType() String
setUuid(int) void
setData(byte[]) void
setSize(long) void
setMediaType(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String
User
username String
mail String
first_name String
last_name String
builder() UserBuilder
getUsername() String
getMail() String
getFirst_name() String
getLast_name() String
setUsername(String) void
setMail(String) void
setFirst_name(String) void
setLast_name(String) void
equals(Object) boolean
canEqual(Object) boolean
hashCode() int
toString() String

After



Overall class diagram

Overall Dependencies

In our overall class diagram, we highlighted where the pattern appears.

Week 17: Metrics

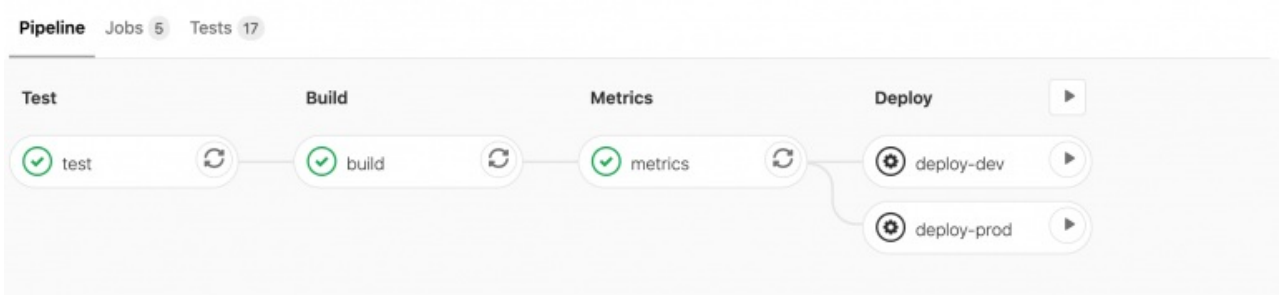
blog.bookly.online

jeanne May 28, 2020

Hey everyone,

For analyzing the metrics of our code we decided to use sonarqube. You can access the server here: sonarqube.bookly.online

We integrated the server into our deployment pipeline. After every build of any branch the metrics are updated. You can find the output of an example pipeline here.



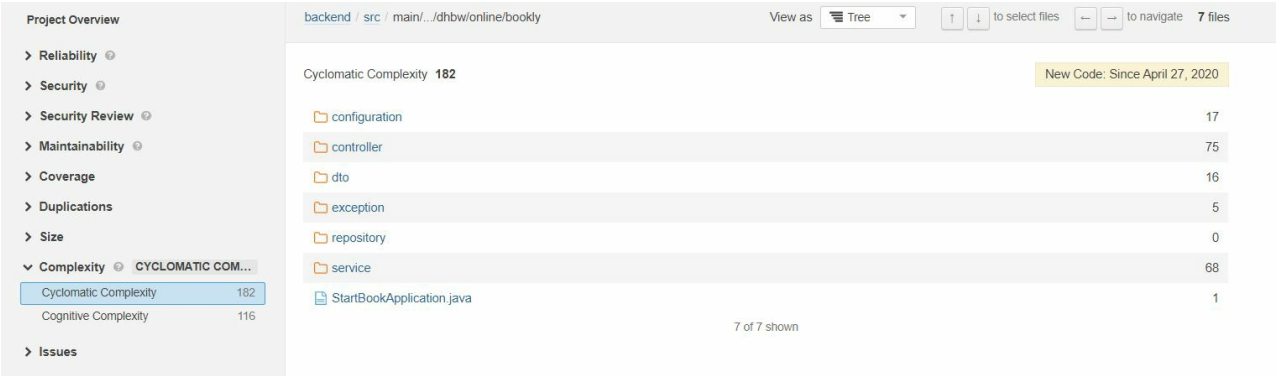
We decided to improve the metrics **Cyclometic Complexity** and reduce our **duplicated lines of code** (discussed and permitted by our Prof).

Cyclomatic Complexity (CYC)

In its simplest form, CYC is a count of the number of decisions in the source code. The higher the count, the more complex the code. So the less nested if-statements the better the rating for cyclometic complexity.

We tried to achieve this in the classes FriendshipBookController and EditPageController. We summed up some if-statements and deleted unnecessary requests. You can find the changes in this COMMIT and in you can see the improvements in the following pictures.

Before:

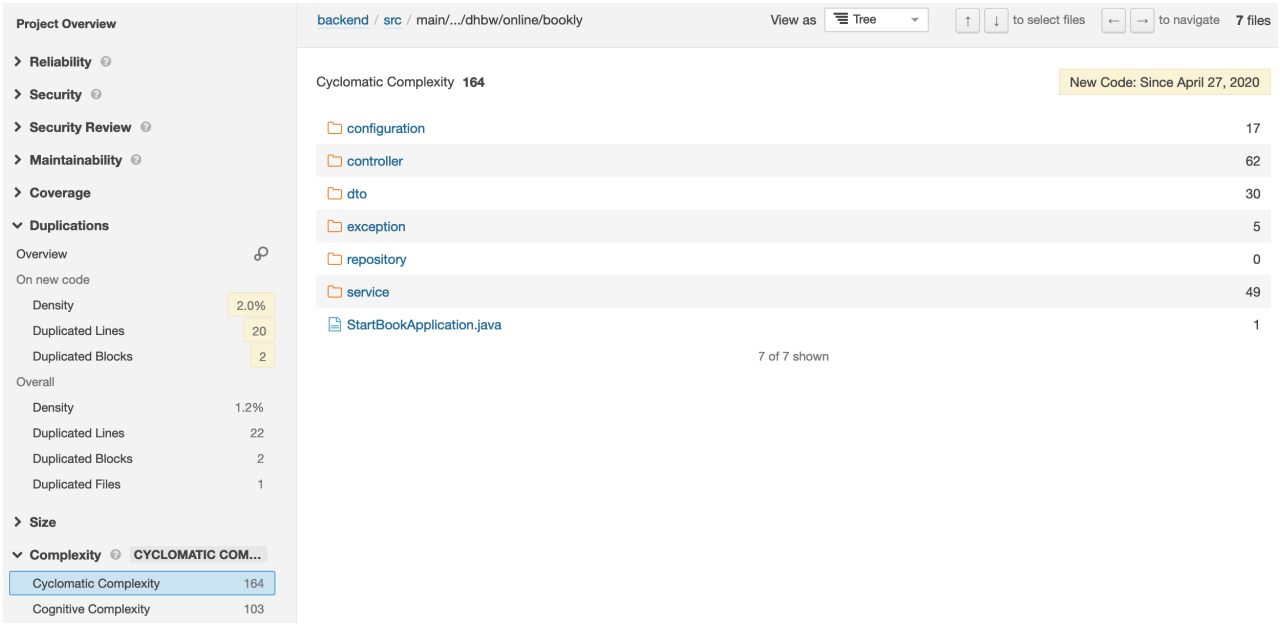


Before Refactoring: all Classes

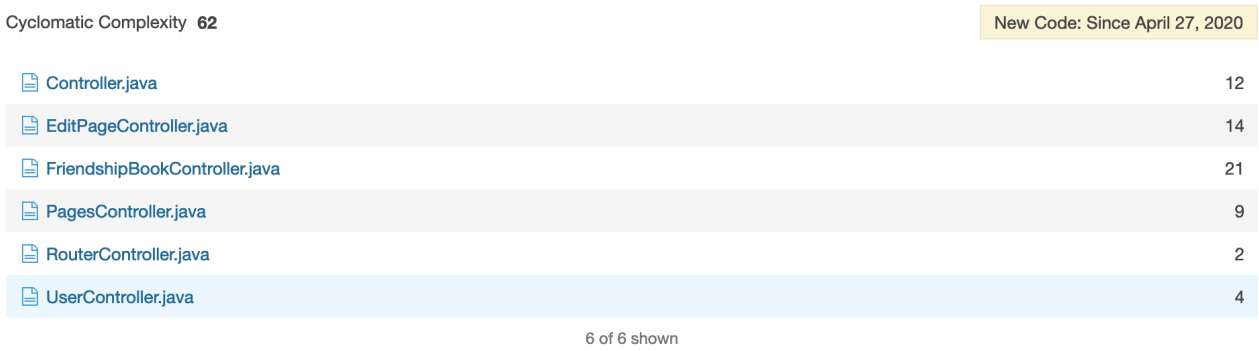
Before Refactoring: Package Controller

Before Refactoring: Package Service

After:



After Refactoring: All classes



After Refactoring: Package Controller

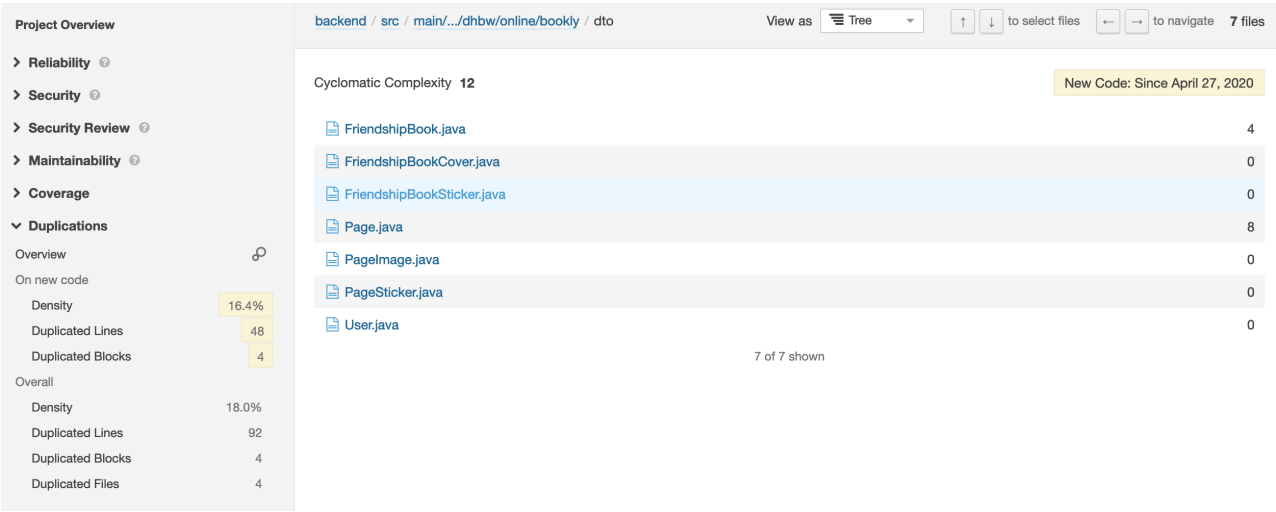
After Refactoring: Package Service

For us, our code seems to be less readable now. We saved more lines of code but we think that our code seems to be more complex.

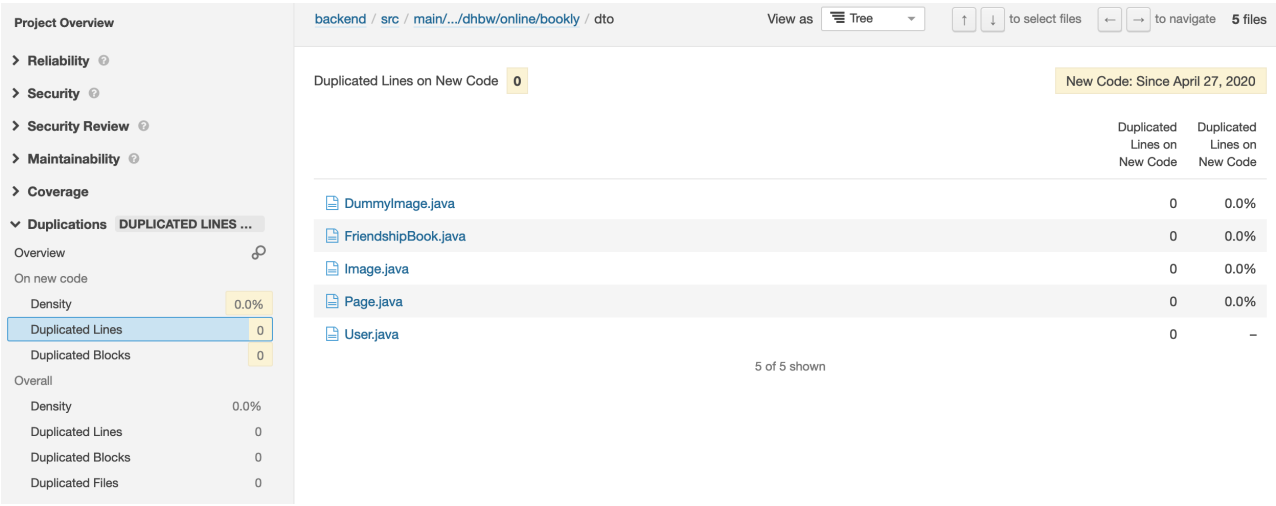
Duplicated Lines

As the name says, this metric is about the number of duplications in our project. At first we had a lot of duplication left. That's especially because of our multiple usage of image data in different objects but with the same attributes. After we refactored it and decided

to use only one class type, our duplication reduced a lot. You can see the difference in this COMMIT and the improvement in the following pictures.



Before Refactoring: Package DTO



After Refactoring: Package DTO

Functionality of Sonarqube

We've got another spot, where sonarqube detected duplicated code but we decided to keep it like this. In more detail we implemented two classes that generates test data for our application (See here). The first class generates data especially for our local environment as we are using the runtime database H2 that is deleting its data after a restart. The second class is especially for our test suite for the backend. Some code parts are overlapping. As those classes are used for different scenarios we are ignoring this note from sonarqube.

Even though sonarqube might be able to locate common bugs, we are not sure whether they are reliable for us or not. For example this one:

```
51                      @JsonIgnore
52  jean…               @OneToMany(cascade = CascadeType.ALL, targetEntity = Image.class)
53  jean…               @ApiModelProperty(notes = "the first sticker of the book", position = 7)
54  jean…               private List<Image> stickers = new ArrayList<Image>() {{
55                          add(new DummyImage());
56                          add(new DummyImage());
57                      }};
58  jean…
```

It's one out of two bugs that sonarqube is locating in our code. But for us those "bugs" are okay. We are grateful to have this warning but we decided to ignore it as we are well aware that we want this inline initialization It's shorter and much more readable than some extra initialization method. Maybe one can do it better..
Update: It's funny but later on we really had a bug doing the initialization like in the picture above So we fixed it by introducing an static initialization method for empty images. Thanks SonarQube <3

So.. It's nice

After we clicked us through the dashboard of sonarqube we became apparent that it helps us a lot to find some hotspots where we should reconsider coding decisions or to find vulnerabilities we didn't even think about. As last example sonarqube gave us a hint that we should probably use a POJO instead of our Database DTO's for our controller as it might be a difficult security issue. This warning also had some explanation text why it was marked. Pretty cool!

Test-Plan

We will soon update our testplan. You will find it here.

SAD

We also added the quality and metrics into our SAD here.

We are well aware that the badges for sonarqube aren't displayed correctly on GitLab. That's a strange behaviour that only occurs on GitLab and not in our local drafts or our local preview. At least the badges are linking to Sonarqube… We are working on that!
Update: we finally fixed it.

Best wishes,

bookly

Week 18: Retrospective

blog.bookly.online

admin June 15, 2020

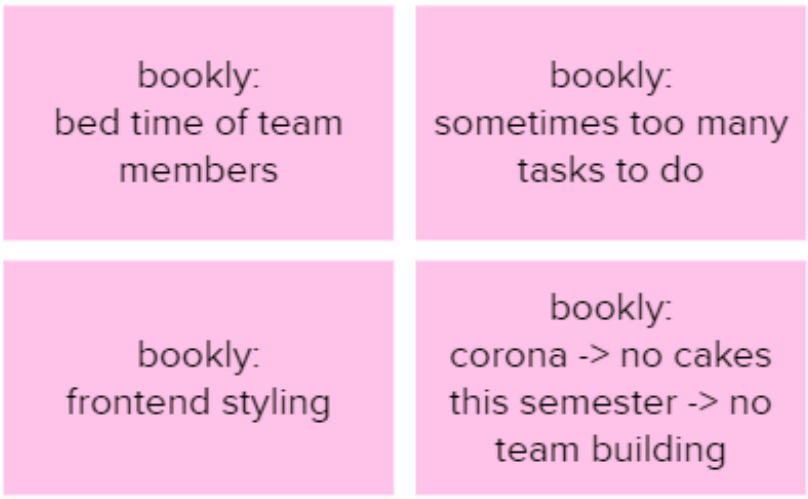
This time the retrospective was done using mural instead of post-its which was difficult because there are always people playing around, but it was also very fun

What went well?

Bookly: TimeManagement in Second Semester	bookly: Deployment, Stage/Dev Server, Reliability

- This semester we had a very good time management when it comes to finishing tasks on time
- We also got our deployment working (stage – and development server)
- We also had to trust in each other when working with deadlines, which worked out really well, thus realibility

What didn't go well



There a some things that didn't go well this semester.

- Because we had different sleep schedules we had difficulty managing meetings and working together at the same time
- For some weeks there are too many tasks when it comes to project management
- No one had experience and (mostly) talent to style our frontend :d
- Sadly we couldn't do any team building this semester due to corona

Actions

In order to fix some of the problems we came to the following conclusion:

- Each group member may bake a cake and we share them between each other
- We may need a fourth group member (this also leads to more cake ;D)
- Take actions in order to not miss any meetings (don't turn off alarm and vibrations on the phone)

Conclusion

Overwall it was a insightful retrospective because we combined and structured our thoughts on how things went and what and how we would do better next time.

Have a great day
Team bookly

Update

One of us baked a corona cake (“Russischer Zupfkuchen”) on the same day after the retro

Week 19: Installation Guide

blog.bookly.online

jeanne June 15, 2020

Hey everyone

So today we want to share our installation guide with you. Actually it should be pretty simple. See the current document here.

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.

So first one has to clone our repository. Within the backend folder of our project there is an *application.properties* file that has to be overwritten according to the own system requirements or own database connection. (The path to an application.properties can also be put explicit into the docker file.) Then the project can be build by excecuting **mvn install**.
Within the backend folder there is the *docker-compose* file located for our project. It starts an local PostgresDB. Also, the compose file builds our project as docker image and starts its container.

Go to localhost:8080 within your browser to access your own bookly instance. Use the attached PostgresDB with localhost:5432.
Have fun!

Best wishes,

Your bookly-Team!

Week 20 – Third test: Field testing

blog.bookly.online

jeanne June 15, 2020

Hey you all

So as our third test, we wanted to do a field test. We were talking with our professor about that as she is in contact with some teachers. We created some survey that you can find here.

So we wanted to clean up our code a bit more so that an field test would give us an more realistic feedback than finding our bugs. But the class with that we were supposed to work with is preparing for exams now. They are not able to test our app now within class. We hope that some students might test it in there freetime and fill out our survey. After consultation with our professor, the third test will not be scored for us if we do not receive any feedback. We will keep you up to date!

via GIPHY

Best wishes,

your bookly-team

Week 21 – Test coverage, Tech-Stack and CI

blog.bookly.online

jeanne June 15, 2020

Hey everyone,

Today we want to sum up our achievements and share them with you. We will link our achieved test coverage, list our used technologies, show our installations and finalize the post by referencing to our working Continous Integration (CI).

Testing and test coverage

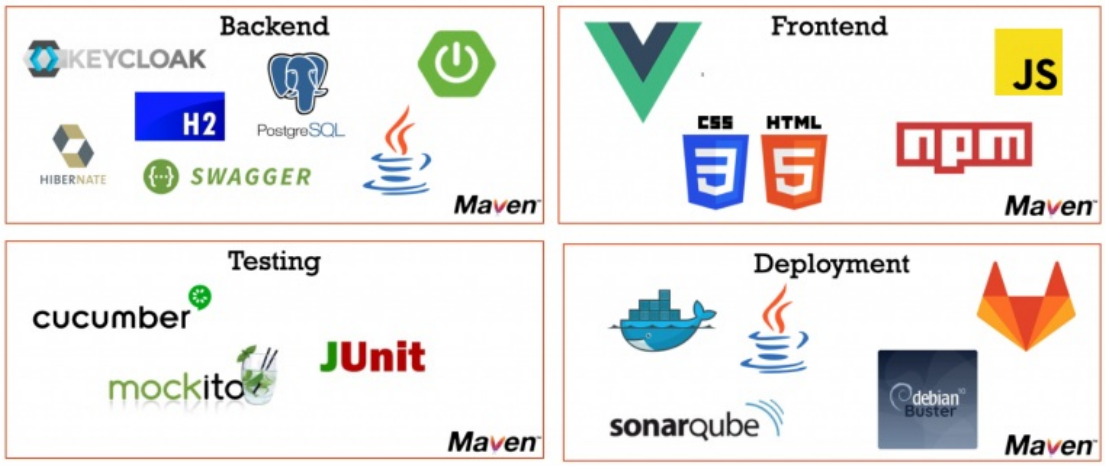
So we already did very detailed blog posts about our tests and metrics.
You can find them here:

via GIPHY

If we did miss something, please let us know!

Tech-Stack

We created some picture where you can find the main technologies we used for our project. It's more than we expected!



used technologies for bookly

Installation and deployment

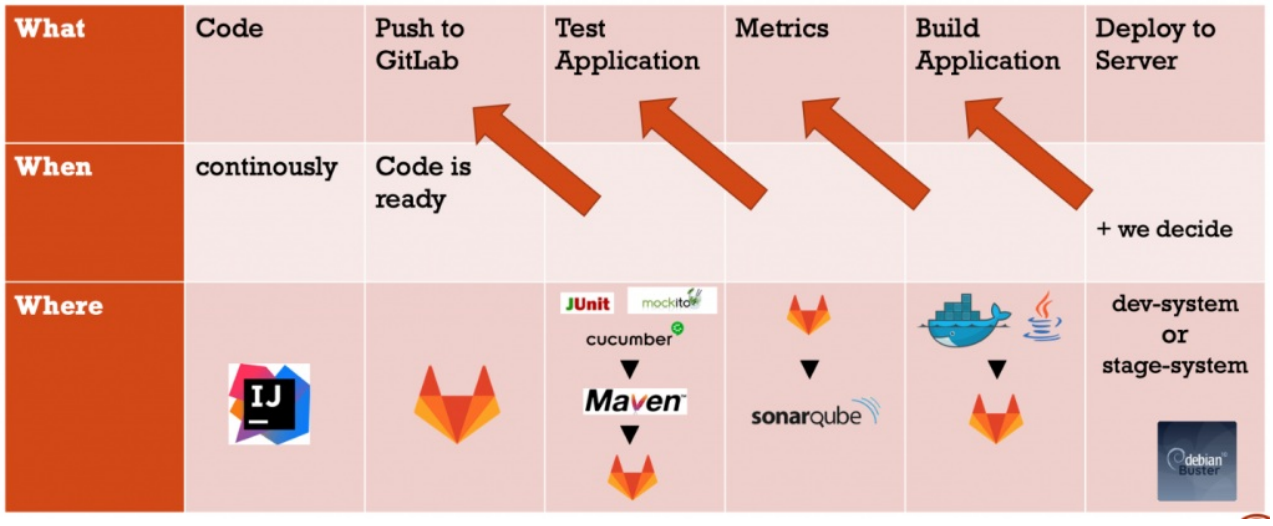
Our installation with another group worked fine, too! See our blog post here and from the group SaSeP here.

Continous Integration (CI) and deployment

We already wrote a blog about our latest deployment ci. You can find it in our metrics blog post.

So we do write code continously with our IDE IntelliJ. As we push changes to GitLab, our pipelines are triggered automatically for every branch. First tests will be runned followed by updating the metrics dashboard and building the application. The application can be build and runned as JAR or as docker container.

At least, we wanted to do the deployment manually (it's only one line of code that we can change any time). We can decide whether to deploy any branch to the dev-server or the master branch additionaly to the stage-system. They can be accessed at bookly.online or dev.bookly.online.



our pipelines – more detailled

Stay healthy,
Your bookly-Team