

Mapp: ING Customer Journey Mapping

T. M. Kluiters

L. P. Overweel

D. A. Vos

J. V. Vos



Mapp

ING Customer Journey Mapping

by

T. M. Kluiters

L. P. Overweel

D. A. Vos

J. V. Vos

in partial fulfillment of the requirements for the degree of

Bachelor of Science

in Computer Science and Engineering

at the Delft University of Technology,
to be presented on Tuesday July 4th, 2018 at 11:00 AM.

Project duration: April 25, 2018 - July 3, 2018
Supervisors: Dr. A. E. Zaidman, TU Delft
J. T. P. Markslag, ING
K. Purmer, ING
J. Brand, ING

An electronic version of this thesis is available at <https://repository.tudelft.nl/>.

Contents

1	Introduction	1
1.1	Report Structure	1
1.2	The Team	2
2	Research	3
2.1	Finding Our Project way of work	3
2.2	Background: Customer Journey Mapping way of work	4
2.3	Existing Customer Journey Mapping Tools way of work	5
2.4	Requirement Analysis implementation	7
2.5	Technical Evaluation implementation	7
2.6	Technical Choices	9
3	Version 0.1: Basic Structure	11
3.1	Front-End Architecture implementation	12
3.2	Back-End Architecture implementation	13
3.3	Sprint Evaluation.	14
4	Version 0.2: Real-Time Collaboration	15
4.1	Suggestion Boxes way of work	16
4.2	REST to Sockets implementation	16
4.3	Data Down, Events Up implementation	17
4.4	Element Inheritance implementation	18
4.5	Back-End Testing software quality	18
4.6	Sprint Evaluation.	19
5	Version 0.3: Minimum Lovable Product	21
5.1	Lane Types way of work visual design	22
5.2	Design Refinements visual design	23
5.3	Continuous Integration software quality	23
5.4	Sprint Evaluation.	24
6	Midterm User Evaluation	25
6.1	User Testing Round 1 usability testing	25
6.2	User Testing Conclusions way of work	28
6.3	Feature Prioritization implementation	29
7	Version 0.4: Visual Overhaul	31
7.1	Material Design Overhaul visual design	32
7.2	Journey Map Flexibility implementation	32
7.3	Lane Element Refactor implementation	33
7.4	Flexbox Refactor implementation	33
7.5	Front-End Testing software quality	34
7.6	Software Improvement Group (SIG) Feedback software quality	36
7.7	Sprint Evaluation.	37

8	Version 0.5: First Release	39
8.1	The Guide Components visual design implementation	40
8.2	Back-End Architecture Changes implementation	41
8.3	PDF Exports implementation	41
8.4	Client-server events API implementation	41
8.5	Sprint Evaluation.	42
9	Final User Evaluation	43
9.1	User Testing Round 2 usability testing	43
9.2	Summary usability testing	47
10	Release: Mapp	49
11	Conclusion	53

1

Introduction

Foreword

Over the past ten weeks, our group worked as a software development team at ING bank. Our objective was purposely left quite broad: to develop an internal tool for Customer Journey Experts (CJEs)¹ that improves some part of their workflow.

After a period of research to understand how CJEs work at ING, we decided to create *Mapp*, an online tool for collaboratively mapping out customer journeys. Over the course of five sprints, we implemented our tool—from setting up the basic structure to a first release. In this report, we describe this process, our results, and the technical and product lessons we learned along the way.

1.1. Report Structure

During this project, we focused on working in an agile, sprint-based fashion. One of the four founding principles of the agile way of work is *responding to change over following a plan* [1]. This is why we decided against following the typical report structure:² our project did not follow a waterfall approach, so neither should our report.

Instead, we decided to structure this report based on our sprints, to emphasize the process by which, and the context in which, we made our technical decisions. There is a chapter for the research we did before writing code (Chapter 2), followed by a chapter for each version of the software we have released and tested (Chapters 3 - 5; 7 - 8). Interspersed in these are two chapters in which we evaluate our midterm and final user acceptance testing sessions (Chapters 6 and 9). Finally, this report is capped off with a showcase of the final release of *Mapp* (Chapter 10) and a conclusion (Chapter 11).

We start off each development chapter with a showcase of *Mapp* at that point, along with some statistics about the sprint. Next, we highlight interesting product choices and technical decisions we made during that sprint. For example, in Section 3.1, we describe the architecture of our front end, while in Section 7.1, we explain our design overhaul to Material Design.

To make it easier for different readers to find content they are interested in, we have also marked each section with the following tags:

- **usability testing** This tag describes usability testing we have done in a controlled setting. Typically, this involved asking a Customer Journey Expert to attempt to map their tribe's most important customer journey in *Mapp* and observing their workflow to find ways to improve our tool.

¹Customer Journey Experts map out, step by step, how customers interact with ING to accomplish some goal, and work to improve these flows. For more about CJEs, see Section 2.2.

²A typical report structure being as follows: Problem Definition and Analysis; Design; Implementation; Conclusion; Discussion and Recommendations

- **way of work** This tag describes what we have learned about the ING way of work around customer journey mapping while developing our software, based on user interviews and usability testing.
- **visual design** This tag describes UX and visual design decisions we have made based on what we learned about the ING way of work.
- **implementation** This tag describes technical implementation decisions we have made.
- **software quality** This tag describes how we kept our software quality high while large parts of our technical implementations changed from sprint to sprint.

Within a chapter, sections generally follow the order in which the tags are explained above. You can find these tags in section headings and the table of contents (page [iv](#)).

1.2. The Team

We created *Mapp* at ING Nederland with a small team of four TU Delft Computer Science and Engineering students consisting of Thomas Kluiters, Leon Overweel, Daniël Vos, and Jelle Vos.

Our coach from TU Delft was:

- Dr. Andy Zaidman, associate professor in software engineering at the Software Engineering Group (SERG)³, department of Software Technology, faculty of Electrical Engineering, Mathematics, and Computer Science.

Our client at ING was:

- Han Markslag, IT Chapter Lead at ING Nederland.

At ING, we were also mentored and advised by the following people:

- Kyra Purmer, Customer Journey Expert at ING Nederland, advised us on the product side and was our day-to-day first point of contact.
- Jesse Brand, Senior Full Stack Engineer at ING Nederland, advised us on the technical side.
- Jorn Jokker, User Experience Designer at ING Nederland, advised us on the design side.
- Eva Vriezolk-Kraaijenbrink, Senior User Experience Designer / Service Designer at ING Nederland, advised us on the product side.
- Stephanie Pelsmakers, Manager Digital Customer Experience at ING Nederland, advised us on the product side.
- Bas Colenbrander, Digital Designer at Resoluut, advised us on the visual design side.
- Leendert Kalfsbeek, IT Area Lead at ING, helped us find a project at ING
- Sander Quik, IT Chapter Lead at ING Nederland, helped us find a project at ING

The following Customer Journey Experts at ING were also very helpful in providing feedback at several points during our development process:

- Marjolijn De Haas, Customer Journey Expert at ING Nederland
- Matthijs van Gaalen, Customer Journey Expert at ING Nederland
- Sibel Batman, Customer Journey Expert at ING Nederland
- Joeri van Raalte, Customer Journey Expert at ING Nederland

³<http://www.se.ewi.tudelft.nl/>

2

Research

Before we began developing *Mapp*, there was a two-week research period for the project. During this period, we set up our work environment at ING, including getting our laptops and software licenses. Further, we figured out what exactly our project was going to be: since the options were very open, we had to decide exactly what to build (Section 2.1). After we choose the project to make an internal tool for Customer Journey Experts, we researched how they work and how we could help improve their workflow (Section 2.2).

Based on this research, we decided to create a customer journey mapping tool, called *Mapp*. To get an understanding for the market, we researched what customer journey mapping tools currently exist (Section 2.3) and used the features of those tools to create some high-level requirements for our tool (2.4). Finally, we decided on the high-level technical architecture for *Mapp* (Section 2.5).

2.1. Finding Our Project way of work

Before the start of our project we had some setting up meetings to discuss what we would work on at ING. We talked to Leendert and Sander, who presented to us the following ideas:

- Gamify the site performance walls to stimulate teams to maintain their projects well.
- Speed up / automate the customer experience design process.
- Kill data inconsistencies between ING's deployments of CDaaS¹ and SNOW².

As a group we decided to work on option two: "Speed up / automate the customer experience design process" because we felt it was the most exciting option due to the possibility to work with employees of ING and solving a problem *people* have by means of Software. Having the opportunity to work directly with employees of ING, our clients, would help us understand the process of Software development in a Business setting.

Customer experience design process

The next step was to investigate the customer experience design process. We talked to Customer Journey Experts (CJEs), user interaction (UX) designers and software developers (devs), specifically we examined the way they communicated. Summarized, the process is as follows:

1. CJEs visualize customer journeys in the form of a customer journey map and pass this on to the UX designers.
2. UX designers draw the screens that customers see and model the interactions with the screens. Images of the screens are distributed to CJEs and devs.
3. After possibly multiple iterations between UX designers and CJEs, the devs receive the screens and specifications of APIs (service design blueprint) to use, which they use to develop the product.

¹Continuous Delivery as a Service

²ServiceNow, an agile development management system

Improve communication between teams

Most overhead in the customer experience design process is due to the communication between teams during multiple iterations. We learned that some CJEs do not specify their customer journey maps with all the information that other teams need and that the creation of customer journey maps can be blocking. To improve this, we decided it would help to have a single online environment where the teams could collaborate and see changes immediately. Lastly, we had to decide what we would show in the online environment: UX designs, service design blueprints, customer journey maps, or a selection of these.

UX designers have a preferred way to work using the applications Sketch³ and inVision⁴. This way works well so it is unfeasible to change this. Service design blueprints are not made for each project, and if they are, it is more to the end of the design process.

We decided the customer journey maps are the best to host in an online environment since they require many version changes and are accessed by multiple people.

2.2. Background: Customer Journey Mapping way of work

In order to make a good tool for Customer Journey Experts, we first had to understand how CJEs work. To do this, we asked ourselves the following questions:

- What is a customer journey?
- What is a customer journey map?
- How does ING do customer journey mapping?

In the following subsections, we summarize what we learned from asking these questions.

What is a Customer Journey?

When a customer interacts with a service from ING we refer to this interaction as being part of a customer journey. A customer journey is the chain of actions a customer performs to complete a goal they have. An example of a customer journey would be "having drinks with friends and splitting the money spent." Two steps in this journey would then be "creating a payment request" and "making a transaction."

What is a Customer Journey Map?

ING wants to enable customers to complete their journeys in the most pleasant way. To design pleasant customer journeys CJEs describe journeys using customer journey maps. A customer journey map presents the way a customer would walk through their journey. The map itself is in the form of a table where every column is a *step* and every row is a *lane*. Each lane describes a certain part of the step and could for example be: the action performed by the customer, a list of questions that the customer has or the customer's emotion. We discuss the types of lanes that ING commonly uses in Section 5.1.

Instead of using real customers in the customer journey maps, ING has defined a collection of personas to use. These personas are described in detail with data such as picture, name, age, living circumstances etc. While specifying a customer journey map, a CJE should always think from the perspective of that map's persona.

Customer Journey Mapping at ING

While many specialized tools exist, customer journey mapping at ING is usually done in more generic design-oriented programs.

ING allows CJEs to work using any tool they want. This means that many different programs are used by different people, from Sketch to Microsoft PowerPoint and Microsoft Excel.

³<https://www.sketchapp.com/>

⁴<https://www.invisionapp.com/>

The task of creating a customer journey map is usually that of the CJE, but sometimes a UX/UI designer is asked to create a screen design before a customer journey map actually exists. We talked to UX designer Jorn, who explained that he and his squad like using their design program Sketch to create a journey map.

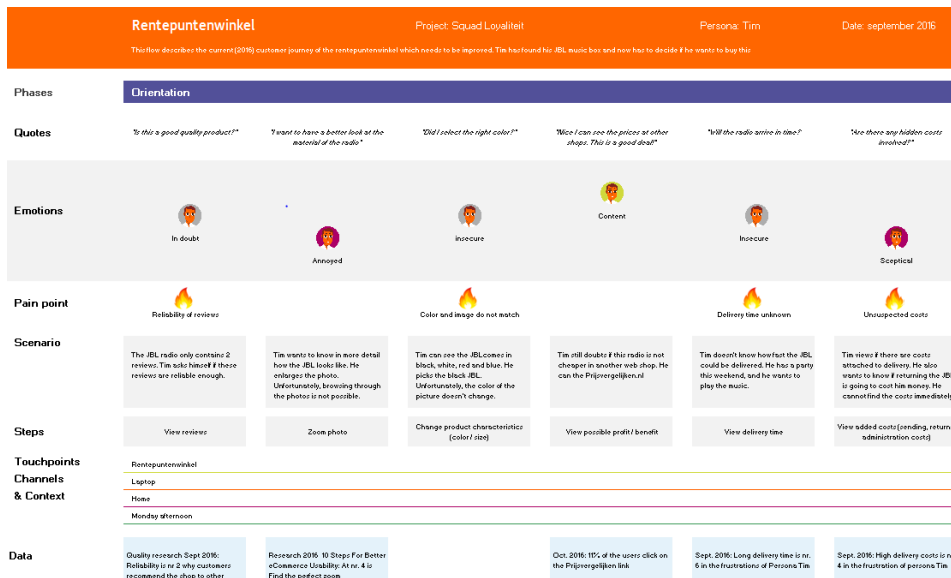


Figure 2.1: An example of a Customer Journey for the *Rentepuntenwinkel*, built by Eva Vriezolk-Kraaijenbrink in Microsoft Excel.

When we spoke to Eva, who teaches CJEs to build customer journey maps, she showed us an Excel template that she made. She recommends those CJEs to use the template and therefore to work in Microsoft Excel (Figure 2.1, for example, is made based on this template). Stephanie, who supervises many CJEs, is a big fan of using Microsoft PowerPoint. We also noticed that Youri, another CJE, used PowerPoint but in a different template and style.

A specialized customer journey mapping tool called Lush is already available on the intranet. In this tool, users write a customer journey map in a step-oriented custom syntax on top of Markdown⁵, which the tool then renders into an HTML page or a PDF. However, Lush is very cumbersome and is therefore very rarely used by any ING customer journey experts.

Lastly, some customer journey maps remain undigitalized. Sometimes, when such a map does need to be suddenly digitalized, a photo is taken and it is slightly edited in another design program.

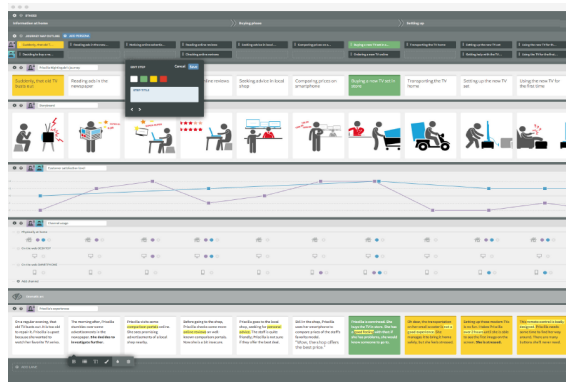
2.3. Existing Customer Journey Mapping Tools way of work

Because every company does journey mapping in their own way, there exist many different applications made for this purpose. Not only are there specialized customer journey mapping tools but also some standard visualizing and diagram-making tools can be used for this purpose. These standard tools include things like Microsoft Excel, Microsoft PowerPoint, and Sketch.

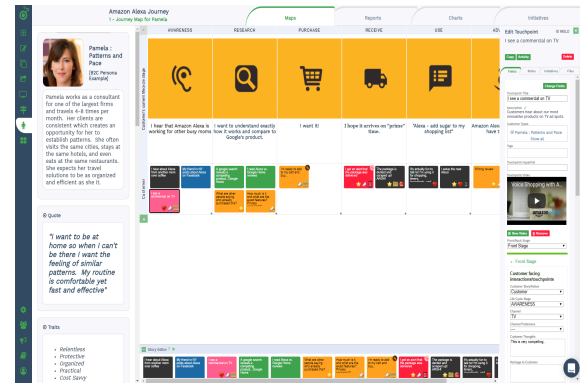
In this section, however, we focus only on tools that are made specifically for customer journey mapping. Several customer journey mapping tools, such as Smaply, Touchpoint Dashboard, CFN Insight, and UXPressia, currently exist on the market. These tools have the following functionality:

- **Smaply** (Figure 2.2a) is a visual customer experience management tool that puts the focus on lanes. It offers a persona editor (and thereby separate journeys for different personas) and a special *stakeholder map editor*, which is used to visualize the ecosystem of stakeholders involved in customer experience.

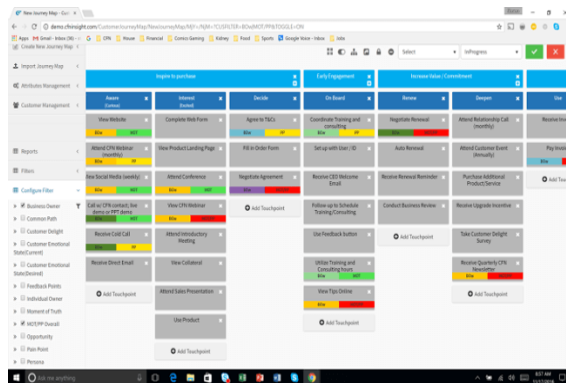
⁵<https://daringfireball.net/projects/markdown/>



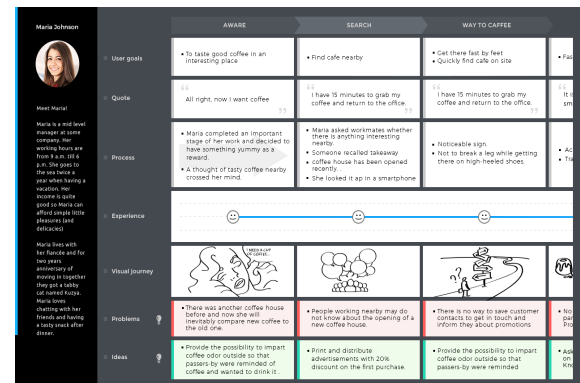
(a) Smaply



(b) Touchpoint Dashboard



(c) CustomersFirstNow (CFN) Insight



(d) UXPressia

Figure 2.2: Existing third-party customer journey mapping tools.

- **Touchpoint Dashboard** (Figure 2.2b) is a webapp for customer journey mapping that also allows for specifying the customer type. Its focus is on touchpoints, which the customer journey map is modeled around. Touchpoints can contain a great amount of data, including videos, images, etc.
- **CFN Insight** (Figure 2.2c) is a webapp for analyzing the customer experience but a major aspect is on assigning business owners who are responsible for the success of certain touchpoints. It is very step-oriented (as opposed to lane-oriented).
- **UXPressia CJM** (Figure 2.2d) is a simple lane-based webapp with built-in collaboration. It also contains a persona editor and a feature to set and manage business goals to improve the user experience based on the customer journey map.

We have summarized the features of these tools in Table 2.1.

	Collab	PDF	Images	Lanes	Templates
Smaply	•	•	•	•	•
Touchpoint Dashboard	•		•	•	•
CFN Insight	•			•	
UXPressia	•	•	•	•	

Table 2.1: Journey map tools comparison on the following features: *Collab* (live online collaboration), *PDF* (PDF export of maps), *Images* (adding images to cards), *Lanes* (having custom, user-defined lane types), and *Templates* (having several customizable templates for journeys).

2.4. Requirement Analysis implementation

Based on the features of customer journey mapping tools currently on the market in Table 2.1 and on our conversations with Eva, Kyra, and Stephanie, we decided that *Mapp* should have the following basic set of capabilities in order to be useful for Customer Journey Experts at ING.

- The ability to export the Customer Journey as a PDF
- The ability to work together in an online collaborative environment
- The ability to upload images
- The ability to create, delete, edit lanes
- The ability to create a template for a Customer Journey and create new Journeys based on this template
- The ability to delete and add steps to a Customer Journey
- The ability to add cards to a step in a Customer Journey
- The ability to delete and edit cards
- The ability to define basic information about the Customer Journey

These requirements are purposely quite high-level: our process is agile, and the requirements will be solidified during each sprint depending on user feedback and the development process.

2.5. Technical Evaluation implementation

During the research phase of this Project it became apparent that the product consists of two components: a front- and back-end component. The front-end must be able to communicate to the back-end and vice-versa. The front-end component would run in a browser whereas the back-end component would run in a server environment.

Architecture

For our architecture we chose Polymer for client-side development and Java 8 with the Spring MVC framework along with Hibernate for server-side development (additionally, we used Mockito, Spring Test and JUnit Jupiter for automated testing). Data is persisted on a PostgreSQL database. In the following sections we explain these choices in-depth.

Front-End

There are many client-side web development frameworks currently available. Some of the more popular ones include Angular [2], React [3], and Vue.js [4]. These include features for routing, server communication, templating, components, etc.

However, as these javascript frameworks are all quite massive and with their main advantage being easily to combine with other front-end libraries, they contain many features we do not need. Since ING runs an internal library "The Guide." with commonly used web-components, we can use the more lightweight framework Polymer [5] and use these web-components instead of combining other front-end libraries. The ING web components are pre-styled using the ING style guide and have also been functionally tested and security-screened.

We chose to follow this internal ING standard and use Polymer because it will speed up our development since we can go to ING engineers and internal forums with our issues, and it will make it easier for ING to maintain our tool in the future after the end of the Bachelor Project.

Also, the Polymer project has made it their mission [6] to get as close as possible to using native web components. Therefore when web components are a widely supported technology, polymer apps will be easily transferable to native code unlike other JavaScript frameworks.

Server

For (web) server-side development, which we will refer to as back-end development from now on, a lot of options were available to us. The following languages were considered; Python, JavaScript (Node) and Java. It was important for us that the language that was used would offer support for a wide variety of requirements; including real-time collaboration, database integration, tooling and experience among us. As we explored the different options more thoroughly, we came to the conclusion that a server written in Java would be the best option. We chose Java because it is a popular language among ING staff and some libraries exist internally to allow easy integration of ING tools such as authenticating ING employees in a unified manner. Also, Java is a language all of us are experienced with, along with its existing tooling such as IntelliJ, an Integrated Development Environment.

Java offers a large variety of options for back-end solutions. By far the most commonly used back-end solution Java offers is Java EE. Java Enterprise Edition offers a rich collection of features for scalable, secure, multi-tiered web applications. Java EE relies on a reference implementation of Java EE such as Jersey [7].

Another solution we took into account was Spring [8] (web). Spring is a Java library built around the Model View Control (MVC) pattern. Spring also offers a rich collection of APIs for Java web server development including a well-documented implementation of Web Sockets.

Initially we chose Jersey as a back-end web server solution as we preferred the ease of setting up a REST web server, serving the client the required data for a Minimal Viable Product. However, as the requirement for real-time collaboration was implemented on the Jersey server it became apparent that Jersey's support for WebSockets brought along a large amount of dependencies that were not part of the original Java EE reference specification. As we were still in the early stages of development the decision was made to migrate from Jersey to Spring. Along with Spring / Jersey we also used Hibernate and Jackson for our data modelling requirements.

Originally we did not plan on using Hibernate [9], as hibernate gives us less control over our Java Objects because of the large amount of annotations involved when using Hibernate. However, Hibernate allowed us to rapidly prototype different entity models because of the ability of Hibernate to generate the Database schema based on the annotations used.

For testing of the Spring Server JUnit 5 [10] (jupiter), Mockito [11] and Spring Test [12] were chosen for writing *unit tests* and *integration tests*.

Database

As our application would require collaborative editing our most important requirement was the ability to use transactions to ensure asynchronous access of data was consistent among clients. Specifically, row-level locking and transactions were important to make asynchronous changes to the same row-entry. We considered four databases: PostgreSQL [13], MongoDB [14], MySQL [15] and SQLite [16]. We compare the databases in table 2.2.

Table 2.2: Database comparison: Postgres, MongoDB, MySQL and SQLite.

	Transactions	Locking	Interface	Type	License
PostgreSQL	Yes	Row-level	API, GUI, SQL	Table, JSON	Postgres [17]
MongoDB	Document-level	DB-level	API	JSON	GNU AGPL [AGPL]
MySQL	Yes, except DDL [18]	Row-level	GUI, SQL	Table	GNU GPL [GPL]
SQLite	Yes	DB-level	API, SQL	Table	Public domain [19]

Based on the features of the selected databases, we chose PostgreSQL as our database because of its ability to lock on row-level, excellent transaction support, integration with Hibernate and most notably: experience among members of the team.

2.6. Technical Choices

To create a tool that people can use to collaboratively work on customer journey maps we decide to build it as a web-app with database, back-end and front-end.

We chose to develop a Spring back-end on a PostgreSQL database with a Polymer front-end.

For the database we require fine-grained locking and transactions to allow concurrent changes to linked data, PostgreSQL fulfills this requirement. Originally we chose to develop the back-end using Jersey, a java EE implementation. However, after the decision to switch from REST to WebSockets we chose to use Spring as it offers a rich collection of APIs for web server development and websockets. For the front-end we decided to work with Polymer, and to not combine it with other frameworks to keep the front-end lightweight. The gap in functionality between Polymer and a more heavyweight framework such as React.js would be filled by the *webcomponents* and *ING the guide* catalogues of ready-made HTML elements.

3

Version 0.1: Basic Structure

		Buttons (+) go here			
<p>Title Rentepuntenwinkel</p> <p>Date September 2016</p> <p>Description This flow describes the current</p> <p>Project Squad Loyaliteit</p> <p>Persona Tim</p>	Quotes	Is this a good quality product?	I want to have a better look at the material of the radio	Did I select the right color?	Nice I can see the prices at other shops. This is a good deal!
	Emotions	In doubt	Annoyed	Insecure	Content
	Moment of Truth	None	None	None	None
	Pain Point	Reliability of reviews	None	Color and image do not match	None
	Scenario	The JBL radio only contains 2 reviews. Tim asks himself if these reviews are reliable enough.	Tim wants to know in more detail how the JBL looks like. He enlarges the photo. Unfortunately, browsing through the photos is not possible.	Tim can see the JBL comes in black, white, red and blue. He picks the black JBL. Unfortunately, the color of the picture doesn't change.	Tim still doubts if this radio is not cheaper in another web shop. He can the Prijsvergelijken.nl
	Steps	View reviews	Zoom photo	Change product characteristics (color / size)	View possible profit / benefit
	Data	Quality research Sept 2016: Reliability is nr 2 why customers recommend the shop to other friends (nr. 1 is Good deals, nr.3 Good quality of products)	"Research 2016 10 Steps For Better eCommerce Usability: At nr. 4 is Find the perfect zoom Sept 2016: 23 % of the users click on the photo"	None	Oct. 2016: 11% of the users click on the Prijsvergelijken link 74% of the users leave the page and not continue order process
	Notes	None	None	None	None
		Buttons (+) go here			

Figure 3.1: Screenshot of version 0.1 of *Mapp*, taken at commit b707ad9.

128 commits (128 added) | **0** front-end tests | **21%** back-end coverage (58/274 lines)

Main changes introduced in version 0.1:

1. Set up a PostgreSQL database
2. Model entities in Java classes with Hibernate
3. Implement a basic API on the back-end
4. Create a basic layout on the front-end with editable fields and cards

During this first sprint, we started to solidify our ideas for the application into a front-end design and a back-end API architecture. Our goal was to get a working web app set up as quickly as possible so that we could validate our ideas of the high-level UX of *Mapp* with customer journey experts and other stakeholders.

We largely succeeded in this goal: by the end of the sprint, *Mapp* ran on our laptops with a full working stack including a database, Java back-end, REST API, and Polymer front-end. In the rest of this chapter, we highlight the front-end architecture in Section 3.1 and the back-end architecture in Section 3.2. We also summarize our end-of-sprint evaluation in Section 3.3.

3.1. Front-End Architecture implementation

The main structure of a Polymer application is defined by its custom elements and how they work together. For our app, we defined this structure during the first sprint, and, on a high level, it remained the same throughout the project¹.

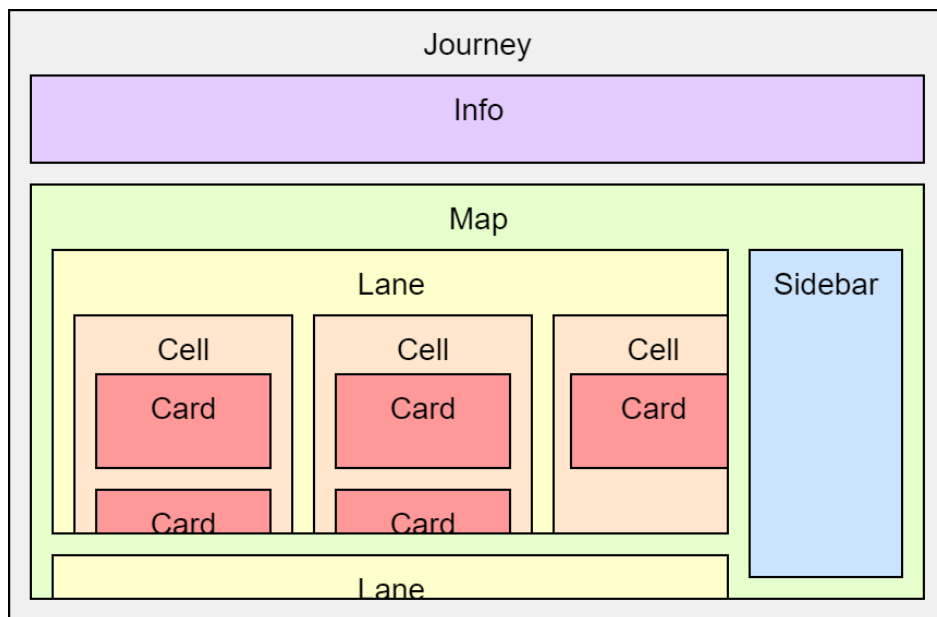


Figure 3.2: A high-level view of the architecture of the Polymer front-end for *Mapp*, showing the nesting of *Journey*, *Info*, *Map*, *Sidebar*, *Lane*, *Cell*, and *Card* elements.

The most important structural components in our app are the *Journey*, *Info*, *Map*, *Sidebar*, *Lane*, *Cell*, and *Card* elements. The nesting of these elements is illustrated in Figure 3.2; they function together as follows:

- The *Journey* element is responsible for maintaining the app's data and its socket connection to the server.
- The *Info* element shows the journey's metadata, including its title, persona, data, and description. It sticks to the top of the page and is always visible.
- The *Map* element shows the main content of the journey. It can be seen as a grid that has columns called *steps* and rows called *lanes*. Because it will quickly grow beyond the size of the browser viewport, the *Map* scrolls both vertically and horizontally.
- *Lane* elements are the main, horizontal axis of the *Map*. A *Lane* has a particular type of information about each *step* of the journey.

¹As you can see, the only large layout difference between Figure 3.2 and the screenshot of MLP 1 in Figure 3.1 is the location of the *Info* element, which moved from the left side to the top of the application.

- `Cell` elements are the secondary, vertical axis of the `Map`. For each *step* of the journey, each `Lane` has a `Cell`, forming a grid.
- Depending on the type of *lane*, a `Cell` contains up to one or several `Cards`. Most of the actual content in the journey is contained inside `Cards`, in the form of text, an image, and/or an emotion.
- The `Sidebar` shows relevant information and hints as to what to write in cards the lane that the user is currently editing. The sidebar can be opened and collapsed by the user.

3.2. Back-End Architecture implementation

During the development of the first version of the application there was no requirement for real-time collaboration, so we decided to develop a Jersey web server with HTTP (REST) endpoints as this was quicker and easier to implement to achieve the desired minimal back-end functionality. We carefully designed an OpenAPI (formerly known as Swagger) specification file [20] that reflected our requirements for the front-end.

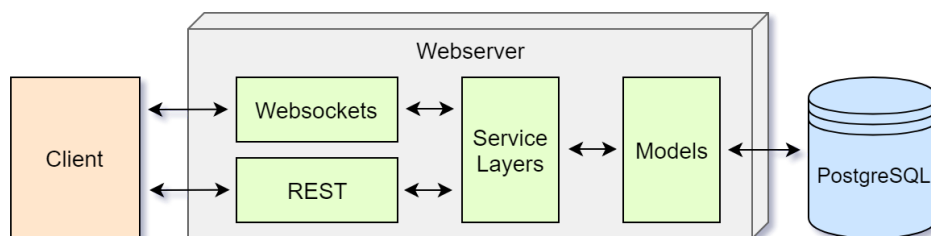


Figure 3.3: A high-level view of the architecture of the back-end and data flow for Mapp.

Since usage of HTTP REST endpoints would be temporary (as to why this is temporary, is explained in section 4.2), we used the *Service Layer* design pattern early in our application (see Figure 3.3). This allows us to decouple the logic required for the different entities described earlier from the Jersey HTTP REST endpoints such that they could later be re-used if different communication methods would be used. A downside of using this design pattern is the increase of boilerplate code required; for each *Endpoint* two classes were defined: *EndpointService* and *EndPointServiceImpl*.

Design Pattern: Service Layer

As much domain level logic code will depend on common components such as Data Repositories, Integration Gateways and User Interfaces, it can be desirable to create a *Service* that encapsulates these different common dependencies such that components using these services do not need knowledge of these dependencies.[21]

Because the individual entities would change considerably throughout sprints we used *Hibernate*. *Hibernate* would allow us to not worry about the Database design, and, instead, have *Hibernate* create the Database Schema based on how we defined our entities as Java classes. This allows us to be very flexible with our data model. However, as a drawback we will not have much control over how our database schema would be configured.

As *Hibernate* uses JDBC, we will not need to worry about database-specific features. As PostgreSQL is our database of choice we used the PostgreSQL JDBC driver.

As our data model (figure 3.2) has a lot of nested entities, it is possible for `Journey` entities to grow quite large. For simple queries such as adding a `Card` entity to a `Cell` entity it was necessary to retrieve the (parent) `Journey` object of that `Cell` to verify if the addition of this `Card` was valid by verifying if this `Cell` is part of the `Journey`. Obviously, it is undesired to fetch the entire `Journey` entity, so we used *lazy loading* to only load data relevant to the client-submitted query.

Lazy-Loading creates *proxies* of Java Objects, replacing methods by *proxy methods*. When a field is queried, like `getName()`, then, in the background a database session is used to fetch the required data. This way we can only fetch data we require.

Lazy loading had an interesting and unexpected side-effect; when a `Journey` entity is initially loaded into memory and returned to a client for initialization in JSON format (when a user connects to the server, the user queries the server for the entire `Journey` object it is connected to) much of the fields of the returned `Journey` would be emptied. We solved this by forcefully loading the entire `Journey` into memory whenever the entire `Journey` was expected.

Back-End Testing

Due to the rapid development of the product and its interaction with the front-end we did not rely on automated testing in the first sprint of the project. This can be explained by the fact that the code changed considerably throughout the sprint due to user-feedback. Instead of relying on automated testing we relied on user testing. This heavily cut down on the time it took to change and implement features as we did not have to change or create associated test classes with them. In later sprints, however, we will discuss the details of how we implemented automated tests on the Back-End.

We wrote some experimental automated tests to explore the way we should write tests for the back-end explaining why we did have (little) coverage.

3.3. Sprint Evaluation

In our evaluation of this sprint, we concluded the following:

- In a discussion with Andy, we decided that we should switch to an architecture that supports real-time collaboration as soon as possible, because many back-end things we built during this sprint will not work in that context.
- The front-end should be centered around lanes first, not steps first.
- There should be some extra added value for using our tool over just using post-its on the wall: maybe some extra information or automatic importing of data?

4

Version 0.2: Real-Time Collaboration

The screenshot shows a web application interface. On the left is an orange sidebar with the following text: Title: Rentepuntenwinkel, Date: September 2016, Description: This flow describes the current, Project: Squad Loyaliteit, Persona: Tim. The main area contains a table with columns separated by '+' signs. The table has rows for quotes, emotions, momentsOfTruth, painPoints, scenarios, steps, data, and notes. To the right of the table is an 'Example' card with a lion icon and text: 'This is an example description where it is explained what such a card/cell would usually contain.' Below the card is a 'Close' link.

	+	+	+	+	+
quotes	Is this a good quality product?	I want to have a better look at the material of the radio	Did I select the right color?	Nice I can see the prices at other shops. This is a good deal!	
emotions	In doubt	Annoyed	Insecure	Content	
momentsOfTruth	None	None	None	None	
painPoints	Reliability of reviews	None	Color and image do not match	None	
scenarios	The JBL radio only contains 2 reviews. Tim asks himself if these reviews are reliable enough.	Tim wants to know in more detail how the JBL looks like. He enlarges the photo. Unfortunately, browsing through the photos is not possible.	Tim can see the JBL comes in black, white, red and blue. He picks the black JBL. Unfortunately, the color of the picture doesn't change.	Tim still doubts if this radio is not cheaper in another web shop. He can the Prijsvergelijken.nl	
steps	View reviews	Zoom photo	Change product characteristics (color / size)	View possible profit / benefit	
data	Quality research Sept 2016: Reliability is nr 2 why customers recommend the shop to other friends (nr. 1 is Good deals, nr.3 Good quality of products)	"Research 2016: 10 Steps For Better eCommerce Usability; At nr. 4 is Find the perfect zoom Sept 2016: 23 % of the users click on the photo"	None	.2016: 11% of the users click on the Prijsvergelijken link 74% of the users leave the page and not continue order process	
notes	None	None	None	None	

Figure 4.1: Screenshot of version 0.2 of *Mapp*, taken at commit 1028c31.

253 commits (125 added) | **1** front-end test | **0%** back-end coverage.

Main changes introduced in version 0.2:

1. Add websockets for real-time collaboration
2. Refactor the front-end to be lane-oriented rather than step-oriented
3. Add a sidebar for suggestions
4. Set up back-end testing

During this sprint, we focused first on the points from our evaluation of our first sprint (Section 3.3). We made a context-aware sidebar that shows users relevant information and tips about the lane they're currently editing (Section 4.1).

Furthermore, we did a large architecture overhaul that affected the full stack of *Mapp*: to enable real-time collaboration, we switched from having the back-end and front-end communicate over a REST API to having them communicate over sockets (Section 4.2). Because of this, on the front-end, we also had to start to think more about Polymer-style persistence and data management (Section 4.3) and element inheritance (Section 4.4).

Finally, we started testing on the back-end (Section 4.5).

4.1. Suggestion Boxes way of work

In 2016 ING went through an agile transformation: ING implemented the Spotify management structure [22] and started using scrum inside the newly formed teams. For their agile transformation, ING also needed to fill many Customer Journey Expert positions and not everyone who entered this position did their job the same way. In the research phase we talked to Eva Vriezokolk-Kraaijenbrink. Eva trains new CJEs and told us about this problem of CJEs.

As a way to partially fix the problem of CJEs with different training backgrounds we decided we could add a suggestion bar to *Mapp* so that it could provide extra explanations. The sidebar changes content depending on which lane is selected and provides a detailed description of that lane. Specifically we heard people like to learn from examples and that we could include example cards in the sidebar. Aside from examples of cards we were suggested to provide CJEs with a few fully filled in customer journey maps as well.

4.2. REST to Sockets implementation

In order for real-time collaboration to work on the front-end, a duplex connection would have to be maintained between the server (back-end) and client (front-end), that is, the server should be able to send data to the client and the client should be able to send data to the server as opposed to the server only being able to send data when the user sends a request.

An important requirement of such connection is that users will only be notified of changes that are relevant to them, that is, when a User 1 is working on Journey *A* and another User 2 is working on Journey *B*, User 1 should not be notified of changes in Journey *B* and User 2 should not be notified of changes in Journey *A* respectively. To combat this, we decided to create the concept of `Rooms`. A `Room` will own one `Journey`. A user would be able to connect to a `Room` and would only be notified of changes relevant to that `Room`.

The most commonly used method for web servers is the usage of WebSockets [23]. Initially we relied on the WebSocket implementation of Jersey, however, we discovered that Jersey's support for WebSockets did not fit our requirements. We noticed the following shortcomings of Jersey's implementation of WebSockets:

- Integrated authentication of socket connections: in order to control the actions a user has over an entity authentication is used. Jersey would require us to write our own authentication API.
- Support for STOMP protocol: The Simple (or Streaming) Text Orientated Messaging Protocol would allow us to have less concerns about implementation of WebSockets on the front-end.
- Dynamic sockets endpoints: In order to differentiate from users being connected to different rooms we would need to uniquely identify rooms and configure unique endpoints users could connect to.

With these shortcomings the decision was made to switch over to a Spring-Based Server architecture. Spring has a rich set of features just for WebSockets, *Spring Messaging*, significantly reducing the development time of the Web Server as WebSockets are integrated seamlessly into Spring Messaging. As we used the *Service Layer* design pattern it was not difficult to switch over to Spring, as much code did not have knowledge of Jersey and thus not depended on it.

4.3. Data Down, Events Up implementation

During MLP 1, we persisted data in the front-end using a REST server. In our view, this matched very nicely with Polymer's concept of custom elements: each element could just take care of sending the server updates about its data when the user made a change. For example, the `Info` element could send a `PUT` request to the server when the user edited the title or description of a journey, and no other element would need to know about this change.

When we switched to sockets (see Section 4.2), however, this nice property broke down: it would not make sense for each element to maintain its own socket connection with the server, or for each element to make use of the same socket connection from different contexts. So, one element had to become responsible for communicating with the server; we quickly decided that this should be the highest-level element: the `Journey`.

This introduced a new challenge: if a user made a change to the text of a `Card` (nested inside a `Cell`, inside a `Lane`, inside the `Map`), how would the `Journey` know about it in order to send the updates to the server? After we struggled with Polymer's two-way data binding, observers, and path linking for a while, one of our technical supervisors at ING introduced us to a front-end design pattern called *Data Down, Events Up*.

Design Pattern: Data Down, Events Up.

This design pattern describes how nested elements communicate with each other when a variable or property changes. When element *A* is nested inside element *B*, *A* is the *child* and *B* is the *parent*; *B* is considered to be "above" *A*.

- *Data Down*: A parent element, using one-way binding, tells its children that one of their properties has been updated.
- *Events Up*: A child element dispatches an event to inform its parents that one of its properties has been updated. A parent listens for these events and takes care of any relevant persistence and propagation.

Events and Data Flow in *Mapp*

In *Mapp*, the *Data Down, Events Up* design pattern is implemented as follows. The `Journey` is in charge of the socket connection to the server is the front-end's source of truth for the state of all other elements. When, for example, a user edits a card, the following happens:

1. The `Card` observes its text field so that when it is edited by a user, it dispatches a `card.updateCard` event.
2. This event bubbles up through the `Cell`, `Lane`, and `Map` elements inside which the `Card` is nested.
3. The `Journey` listens for the `card.updateCard` event and forwards it to the server over the socket connection.
4. The server receives the update, persists it, and generates a new `card.updateCard` event that it sends to all clients listening to the socket connection.
5. The `Journey` receives the server's `card.updateCard` event and updates the relevant card in its JSON `journey` property.
6. Because the `Map`, `Lane`, `Cell`, and `Card` objects all have a one way bind directly with the `Journey`'s `journey` property, the updated `Card` text gets propagated down and rendered into the DOM automatically.

This approach greatly simplifies real-time collaboration because the front-end does not need to differentiate between its own changes and other clients' changes to display updates, since it receives

them all in the same way in step 5. Please refer to Section 8.4 for a complete overview of the events API that the client and server use to communicate.

4.4. Element Inheritance implementation

Standard inheritance in Polymer is in some ways not what we are used to from Object-Oriented Programming. While Polymer stays true to the principles of *code-reuse* and *overriding*, it does so unintuitively.

An element's main body is specified in the `<template>` tag, which contains the HTML that is put into a `#shadow-dom` when the element is loaded onto the page. This makes it so that every element's behaviour is contained to only work on the piece of HTML that it defines, making them modular *components*.

By default, Polymer considers an element's only field to be the `<template>` tag. This tag also contains the element's `<style>` tag. This means that when a sub-element wants to define its own styling it will *override* the entire `<template>` tag of its parent.

Apart from the template field, Polymer elements also contain a functional part. In this part, several functions define a component's behaviour: Next to its *properties*, it defines optional behaviour when the component has finished loading and allows for user-defined helper functions.

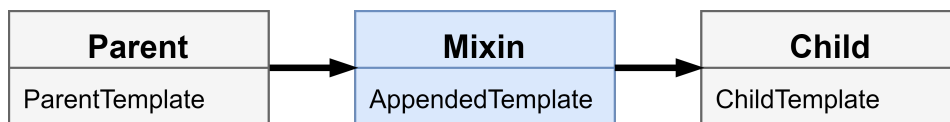


Figure 4.2: Design pattern: ComponentMixin

In order to be able to extend a base element with some HTML elements and custom styling we need to append to the `<template>` tag of the parent. We have devised a design pattern for this inspired by a `TemplateMixin`¹.

Design pattern: ComponentMixins

Rather than extending the base element, a sub-element should extend a `ComponentMixin`. The mixin imports the `<template>` from a child element (if it exists) and returns a class that extends the super class containing the parent's template with the child's template appended, see Figure 4.2.

The component's behaviour is automatically and correctly extended by Polymer. Because we have this virtual mixin class that we extend in our child class, we do not have to do any sort of appending ourselves. This also means that any JavaScript helper functions that are written in the parent class will be accessible and overridable in the child class. `ComponentMixins` are used in our project in every element that inherits, examples include child elements of: the `Lane`, `Cell`, and `Card` elements.

4.5. Back-End Testing software quality

In this version of the application the codebase grew more stable, as a result of the increase of stability we wrote unit tests and integration tests of the individual components and their interactions as code changes in the individual components would have little impact on the tests and thus would require less time to refactor.

Near the end of this sprint we wrote tests using JUnit 5 (Jupiter), Mockito and Spring Test. Each method of a service is tested by creating a nested class inside the service test class. These nested

¹<https://github.com/jaichandra/template-mixin>


```
class ServiceTest {
    private Service service;

    @BeforeEach
    void setUp() { /* create service and inject mocked dependencies */ }

    @Nested
    class ServiceMethodTest {

        @Nested
        class GoodWeatherTests { /* test correct arguments */ }

        @Nested
        class BadWeatherTests { /* test wrong arguments */ }
    }
}
```

Figure 4.3: Example of a back-end unit test.

classes contain two classes: `GoodWeatherTests` and `BadWeatherTests`, as illustrated in Figure 4.3.

It should be noted that these tests were not merged into the master branch before the end of the sprint, and therefore the code coverage of 0% in the header, does not reflect the original coverage of 68%. The tests could not be merged in time due to the change of Jersey to Spring, requiring a lot re-factoring work.

We also wrote integration tests that would run the server with an in-memory database (so that we have full control over database memory) and mock the front-end with a WebSocket client written in Java. As there was little support for WebSocket integration testing in the frameworks we used the tests required a lot of code to support basic features. We had to define our own code for tasks such as connecting-, subscribing to the server and verifying server interaction. To do this we made an abstract test class `WebSocketIntegrationTest`. This class has several helper methods: `subscribe(topic : String)`, `send(endpoint : String, data: Object)` and `getBacklog()`. Any integration test extends from this helper class.

- `subscribe(topic)` subscribes the underlying WebSocket client to the given topic.
- `send(endpoint, data)` will send the given data object in JSON format to the socket endpoint.
- `getBacklog()` returns a `BlockingDeque` containing the events that were published to the optics the WebSocket client subscribed to.

Because there is a significant (random) delay of 1 to 150 milliseconds between sending a request and receiving a response the `BlockingDeque`'s method `take()` is used to retrieve all data sent back to the client. This way we do not have to include `Thread.sleep` into our test code which can result in non-deterministic test-outcomes due to the possibility of requests taking longer than 150 milliseconds in rare cases. A downside of using `take()` is the fact that in the event that a server-side error occurred and no data is sent back to the client as a result; tests may never finish resulting in builds never finishing. As a compromise `poll()` can be used where a time can be specified such that the `BlockingDeque` will wait at most for that given time period for the data to become available - else an exception will be thrown.

4.6. Sprint Evaluation

In our evaluation of this sprint, we concluded the following:

- Lanes should not all be just text boxes; the emotions lane should be a vertical slider.

- We need to start focusing on software quality, especially on the front-end, and figure out whether we can set up some continuous integration.
- We should allow more flexibility in what lanes a journey can have, beyond the few we have hard-coded right now.

5

Version 0.3: Minimum Lovable Product








Persona	Title	Date	Project	Description			
Tim	Rentepuntenwinkel	September 2016	Squad Loyaliteit	This flow describes the current (2016) customer journey of the rente			
Steps	View reviews	Zoom photo	Change product characteristics (color / size)	View possible profit / benefit	View delivery time	View added costs (sending, returns, administration costs)	Search for saved rentepunte
Scenarios	The JBL radio only contains 2 reviews. Tim asks himself if these reviews are reliable enough.	Tim wants to know in more detail how the JBL looks like. He enlarges the photo. Unfortunately, browsing through the photos is not possible.	Tim can see the JBL comes in black, white, red and blue. He picks the black JBL. Unfortunately, the color of the picture doesn't change.	Tim still doubts if this radio is not cheaper in another web shop. He can the Prijsvergelijken.nl	Tim doesn't know how fast the JBL could be delivered. He has a party this weekend, and he wants to play the music.	Tim views if there are costs attached to delivery. He also wants to know if returning the JBL is going to cost him money. He cannot find the costs immediately.	Tim wants to know if he can buy the JBL with his amount of rentepunten. He cannot see the amount of rentepunten he owns. He is
Quotes	"Is this a good quality product?"	"I want to have a better look at the material of the radio"	"Did I select the right color?"	"Nice I can see the prices at other shops. This is a good deal!"	"Will the radio arrive in time?"	"Are there any hidden costs involved?"	"Can I buy this JBL with my points?"
Emotions							
Pain points	Reliability of reviews		Color and image do not match		Delivery time unknown	Unsuspected costs	Points not visible
Data	Quality research Sept 2016: Reliability is nr 2 why customers recommend the shop to other friends nr. 1 is Good deals, nr.3 Good quality of products)	Research 2016 10 Steps For Better eCommerce Usability: At nr. 4 is Find the perfect zoom Sept 2016: 23 % of the users click on the photo		74% of the users leave the page and not continue order process Oct. 2016: 11% of the users click on the Prijsvergelijken link	Sept. 2016: Long delivery time is nr. 6 in the frustrations of Persona Tim	Sept. 2016: High delivery costs is nr. 4 in the frustration of persona Tim Nov. 2017: No transparency about costs is nr.2 in most registered complaints	
Notes							

Figure 5.1: Screenshot of version 0.3 of *Mapp*, taken at commit 1e70257.

402 commits (149 added) | **1** front-end test | **68%** back-end coverage (113/166 lines)

Main changes introduced in version 0.3:

1. Change the emotion cards to contain an emoji-based slider
2. Allow for deletion of steps and cards
3. Set up a linter and testing on the front-end
4. Implement user-defined (custom) lanes
5. Show editor buttons only when hovering over elements

This was the sprint to build our Minimum Lovable Product (MLP): although we constantly went to Kyra and Jorn for feedback during the first two sprints, we planned to evaluate version 0.3 in a more formal, user acceptance testing setting (See Chapter 6). We also changed our sprints from ending on Fridays to ending on Tuesdays because this aligns better with the ING work week.

On the product side, getting ready for our MLP required quite some work. We defined which lane types we were going to implement in *Mapp* (Section 5.1) and implemented several of these lanes. Using Jorn's feedback, we refined the UX and visual design aspects of our app (Section 5.2). On the software quality side, we also came up with a solution for continuous integration, in the form of commit and push hooks for front-end tests and linting, and Sonar for back-end tests (Section 5.3).

5.1. Lane Types way of work visual design

Until this release, we only implemented one type of lane: a single text card lane that supported typing some text for each step. Then based on customer journey maps hanging around the ING office and on Eva's Excel customer journey map template (see Figure 2.1), we decided which lane types to implement in *Mapp*. For an overview of these lanes, see Table 5.1.

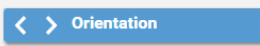

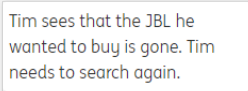
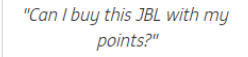


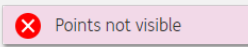

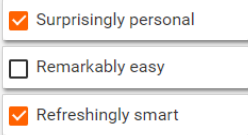
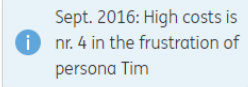
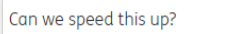
Lane	Description	Design
Timeline ¹	What phase of the journey the persona is in; e.g. <i>awareness, familiarity, consideration, purchase, and loyalty</i> .	
Step	A brief description of the action performed by the persona.	
Scenario	The story behind the step.	
Quote	What the persona is thinking, in the form of a quote.	
Emotion	How the persona feels, on a scale from a happy to an angry emoji.	
Emotion Description	A text description of how the persona feels, such as <i>anxious</i> or <i>excited</i> .	
Pain Points	Unpleasant experiences for the persona at this step.	
Context	The persona's location, channel, device, etc.	
CX Principles ²	ING's customer experience principles: <i>surprisingly personal, remarkably easy, and refreshingly smart</i> .	
Data	Research and data to support the claims in other lanes.	
Notes	Miscellaneous info that does not fit in any other lane.	

Table 5.1: The different lane types we implemented for *Mapp*. The Description column explains what content the lane contains, and the Design column explains how we designed the lane in *Mapp*.

¹Because phases in the timeline span multiple steps, implementing them required quite a large front-end refactor. See 7.4.

²We implemented this in version 0.5.

5.2. Design Refinements visual design

As we had a good start for the implementation of *Mapp* at this point, it was time to refine the way users interact with the product. We sat down with Jorn Jokker, a user interaction designer at ING. Jorn noticed that the info bar that we modeled after some customer journey maps we saw earlier, took up a lot of horizontal space. We learned that although on paper this might look good, people are used to a different format for digital products. To fix this we moved the info bar to the top.

Emotion cards had a dropdown menu at this point and Jorn suggested we should turn this into a slider as it demonstrates to people that there is a range of emotions. We implemented emotion cards as a vertical slider of which the knob is an emoji and the emoji changes depending on the height it is at.

Lastly, there were a lot of plus-buttons visible on the application that could be used to create a new card in a cell. All these plus-buttons might be confusing to users as it does not align with the *what you see is what you get* (WYSIWYG) principle. We fixed this, and similar problems, by only letting buttons appear while hovering the cursor close by.

5.3. Continuous Integration software quality

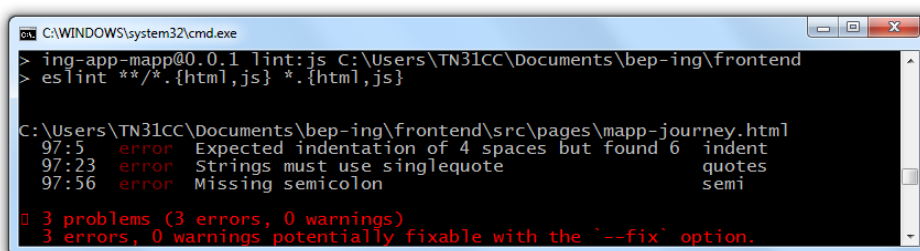
To ensure that we maintain well-styled code and to prevent us from merging failing builds, we wanted to use a continuous integration (CI) service such as Travis CI [24] or the open-source option Jenkins [25]. However, due to the fact that ING dependencies that our project depend on are hosted by the private ING network, external CI services were not an option as they would not be able to reach the closed-off network of ING. It also was not possible to use ING's CI servers as those only supported projects hosted on GitLab.

Front-End

As a workaround, we make use of git's pre-commit and pre-push hooks, implemented as bash scripts that are executed when a user issues the `git commit` and `git push` commands respectively. The NPM package Husky [26] conveniently creates the bash scripts and allows users to define a list of commands in a configuration file that is in the project directory. By having the configuration file in the project directory the commands executed inside the hooks can be easily updated in commits, as the hook files themselves are located in the ignored `.git/` directory.

Our pre-commit and pre-push hooks run the following commands:

pre-commit: Lint for CSS, JS and polymer
pre-push: Front-end tests + lint for CSS, JS and polymer



```
C:\WINDOWS\system32\cmd.exe
> ing-app-mapp@0.0.1 lint:js C:\Users\TN31CC\Documents\bep-ing\frontend
> eslint **/*.html,js *.html,js

C:\Users\TN31CC\Documents\bep-ing\frontend\src\pages\mapp-journey.html
 97:5   error  Expected indentation of 4 spaces but found 6  indent
 97:23  error  Strings must use singlequote                  quotes
 97:56  error  Missing semicolon                            semi

  3 problems (3 errors, 0 warnings)
  3 errors, 0 warnings potentially fixable with the `--fix` option.
```

Figure 5.2: Output of the JS linter for the code `console.log("test")`, 3 errors are triggered.

For CSS, JS and polymer linting we use ESLint [27], stylelint [28] and polymer lint [29] respectively. Front-end tests are ran using Selenium [30] (see Section 7.5). The configured hook scripts will only allow the user to complete the command (committing or pushing) if no errors are found. Note that warnings such as deviations from the ESLint best practices are allowed, but the user will be notified about them. An example of the output for one of the linters is shown in figure 5.2.

Since many errors found by the JavaScript and CSS linter are only stylistic such as: amount of tabs, missing whitespace and quote style, the linters offer a `--fix` command to fix these simple issues. We have configured the `lint:js-fix` and `lint:css-fix` NPM commands to run with the `fix` flag to help speed up the process of committing.

Back-End

As we did not have access to a CI server, maven's surefire plugin is used to run all tests locally. When a pull request is created the pull request is checked out locally and `maven test` is run. Depending on a successful local build, the pull request would be merged into the development branch by a reviewer. As the tests were not automatically run on a continuous integration server it was hard to know if these tests would succeed on an isolated machine. In numerous cases tests would not work on one machine but would work on another. This could have been prevented if we did have continuous integration. Continuous Integration would also not require Pull Requests to be locally checked out and verified as reviewers could look at the logs of the CI server. This made it significantly harder to have insight into code quality during review of pull requests as it would require much more time and effort.

In order to still have different quality measures for the back-end code, Sonar is used. A Sonar server would run in a Docker container on a local machine. On execution of `maven install` a code coverage report was generated by JaCoCo and pushed to the sonar server. Sonar will then process these results along with the produced code and create a quality report. The quality report informs us about cyclomatic complexity, tight coupling and other bad aspects (code smells) of code that require re-factoring.

5.4. Sprint Evaluation

We did not evaluate this sprint as we did the others because we evaluated version 0.3 of *Mapp* through our midterm user tests. Our conclusions from this evaluation can be found in Section 6.3.

6

Midterm User Evaluation

To make *Mapp* as useful to Customer Journey Experts as possible, it was important to get their feedback as early as possible. That's why we did this first round of user acceptance testing on our Minimum Lovable Product, version 0.3 of *Mapp*.

6.1. User Testing Round 1 usability testing

For the midterm evaluation we tried to schedule as many tests as possible. We ended up running the current version by four Customer Journey Experts. In this section we explain how these tests went and what we learned from them¹.

The exercise for Customer Journey Experts:

- **Phase 1:** We asked CJE's to recreate a small-scale journey they are accustomed to.
- **Phase 2:** When the CJE's had questions we would answer them and if they couldn't find a function they were actively looking for we would point it out. At the end we would guide them past the functionality they might have missed.
- **Phase 3:** We asked CJE's if they could use the tool as it currently is, and what they're still missing.

We conducted these user tests with Kyra, Marjolijn, Matthijs, and Sibel.

Kyra Purmer

Phase 1 Kyra, being more familiar with the UI of *Mapp*, started mapping out a fictional Customer Journey. As she added more cards she realised she was unfamiliar with the concept of 'Quotes:

" What do I have to fill in here? Can I delete this? "

It also took some time for Kyra to figure out that the Emotion lane requires the user to drag the emoticons.

Next, Kyra filled in more cards but mentioned that it would have been great if she could add images of UI screens, as she is used to working with images in her own Journeys:

" It would be really nice if I could add UI screens to steps. "

Then, much like the first comment Kyra made, it was unclear what a certain lane, scenarios, required:

" What are scenarios? "

¹The quotes in this section have been translated to English.

She also mentioned it would be great to drag cards into other cells. Next, Kyra added a step at the beginning of the Journey, but, could not delete it anymore due to a bug.

Kyra wanted to add a new lane which would keep track of API changes for certain steps. She was not sure if this would require a new Lane or could also be left in notes.

The Customer Journey Expert then noted that it would be desirable to be able to delete and move lanes:

“ It would be great if I could delete lanes or even move them. ”

Phase 2 Kyra managed to explore all features our version had at this point and gave us important feedback. She mentioned that the ability to add lanes was a nice addition. She also would like to have the ability to use a date picker for filling in the 'date' property of a Customer Journey. Lastly, she mentioned that it would be great to be able to customize the style of the lane.

Phase 3 Kyra could definitively see herself using Mapp if she could add images to cards, as this would make mapping her own Journeys much easier.

Marjolijn de Haas

Phase 1 Marjolijn first explored the view. She saw Thomas delete an accidental card that was left on the screen, which caused an immediate update on her screen:

“ Cool, this really is live! ”

Marjolijn had to ask us what the suggestion bar was for and was also interested if there would be some kind of start page. When she tried to type in a card she was struggling, trying to delete the placeholder text before she would enter anything:

“ Oh, the dots disappear when I start typing. ”

She also noticed a small bug where the text would fall out of the box in the 'quotes' lane. After playing around and pointing at the 'step' lane she asked:

“ Is there a way to add images to this? ”

Marjolijn figured out what multi-card lanes were because the plus button stayed visible even when she filled in one card. She did mention that the cards take up quite a lot of vertical space, which made it that she wanted to put less text in them. Marjolijn also figured out card deletion by herself:

“ If you click and don't fill in anything, does it stay there? Oh, this is how I delete it. ”

Marjolijn also mentioned that she designs customer journey maps with the specific purpose to target one part of it and improve on that. She suggested we add symbols or ways to highlight and emphasize both lanes and certain cards separately. This was the point of the experiment where she mentioned a couple of small things that were on her mind:

- If you add a card it should be selected by default.
- Should 'scenarios' be plural?
- The map should be big enough so you can scroll to type in the middle of the view.

Lastly, she thinks that there should be no distinction between single cards and multi-cards:

“ I - as a user - have no idea why this lane shows dots and this lane shows plus buttons. ”

She thinks that multi-cards are intuitive and therefore all cards should be multi-cards.

Phase 2 It was very hard for Marjolijn to find how to add custom lanes so we had to point it out. Marjolijn could see the benefit of the added flexibility but to be able to delete them is essential. She was also looking for a way to drag lanes (by clicking on many parts of the lane) so we had to explain this was not yet a feature.

Phase 3 When we asked if Marjolijn could use the tool as is she had a couple of comments. Firstly, she liked the idea of using a device to create a customer journey map instead of doing it on a brown paper, as is common. She was also fond of the low amount of effort she had to put in to specify emotions. She does consider the addition of images to the tool crucial if she wanted to use it:

“ I don’t feel like reading long textual journey maps, I’m a visual person. ”

Marjolijn specified that just a lane with images wouldn’t suffice. She would like to always be able to add text or images. She also thinks the lanes could be better separated and that the lane name pane could be frozen so it will still be seen when you’re scrolling far to the right. Lastly, she is unsure she would use the suggestion bar:

“ I rather work on the basis of an example than using an instruction. ”

So the addition of journey map templates would sound great to her.

Matthijs van Gaalen

Phase 1 Matthijs, unlike the others, started by filling in the top bar. For the persona he filled in a question mark. The other fields were also not entire clear to him:

“ I would propose you change this to something along the lines of ‘Which tribe or squad made this canvas?’ ”

Matthijs did have a good time exploring most of the rest of the tool. He liked that you could insert steps both right and left of another step, to add cards using the floating plus button and he even smiled when he dragged the emotion slider.

“ Actually, ‘quote’ should be part of these emotions, so maybe it should be below. Or above it. Or to the side. As long as it’s close by. ”

Phase 2 Matthijs had a hard time understanding the default lane names and he did not find the suggestion bar after he first collapsed it without thinking about it. He said that the suggestion bar is far outside of his field of vision and proposed adding this information left, close to the lane name, so it is better visible.

“ What could help is to go about it more visually and to transform the labels. For example: Instead of ‘scenario’ say ‘I do...’ ”

He also had suggestions for the other lane names.

Phase 3 Like Marjolijn, Matthijs said he was a visual person; he likes to first see an image, then a little description, followed by an emotion and only then a substantiation of that emotion using explanations and data. Matthijs is not sure everything should be flexible:

“ The majority of people do not configure their setup. ”

Matthijs is not yet sure he would use the tool. He explained that his way of working is to first note down everything he can think of to back the chosen emotion and only later to distill this into a simple overview. He thinks this might be hard to do when you are asked to categorize such notes (in all the different types of lanes).

Sibel Batman

Phase 1 Sibel was surprised she had to start up Google Chrome (as Mapp is only supported on this browser) and made the comment that:

“ 90% of Customer Journey Experts use Internet Explorer. ”

We recognize that not being able to support all browsers is not optimal, however, the amount of time it would require to have Mapp support all three major browsers is beyond the scope of this Bachelor Project.

Like many other Customer Journey Experts, Sibel was confused what certain lanes meant and commented:

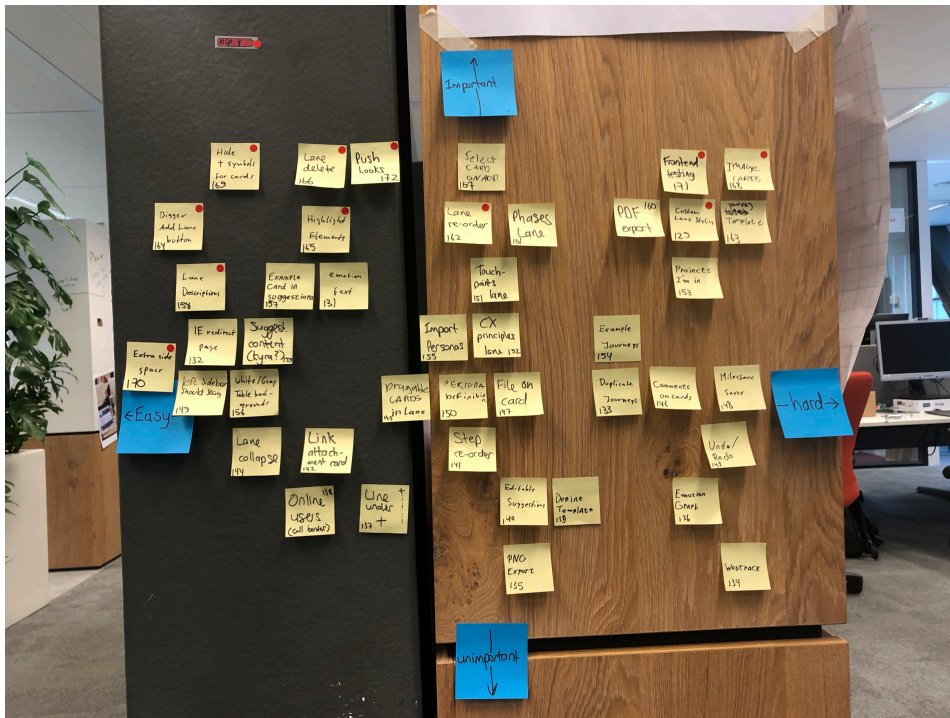


Figure 6.1: Top 50 issues to tackle, sorted by importance and difficulty. Post-its that are labeled with a red dot are in the backlog of sprint 4.

“ I don’t understand what scenario’s would mean. ”

It also wasn’t immediately clear to her that she could add more steps by hovering over a step, after helping Sibel find the button she started creating multiple steps with relative ease.

She also noted how it would be great to be able to see lines between emotions (connecting the emoticons with a curve):

“ It would be cool to have a line go through all emotions, as this allows us to visually see the improvement of making the user happier at a certain step. ”

Phase 2 Some features were not clear to Sibel initially and required some explaining. However, after we explained the features to her she had no trouble modeling the Customer Journey to her needs. She would like to have the ability to work with a Customer Journey template that is according to what she is used to working with, as most lanes she is used to were missing.

Phase 3 Despite the default collection of lanes not fulfilling the needs of Sibel, she can see herself working with Mapp:

“ I can absolutely see myself working with this, it’s so much easier, especially if you have the template I am used to. ”

6.2. User Testing Conclusions way of work

From the user tests we noticed that there were still some essential functions that were missing, but mainly that we needed to make the tool more flexible. While we based the tool on the template we received from Eva, it should also be usable by CJE that prefer working in their own, original format.

From all the evaluations we received massive amounts of feature requests. In the limited time of our project we decided to select only a handful of features to implement per sprint. In Section 6.3 we explain how we did this.

6.3. Feature Prioritization implementation

After the midterm user tests we had a group meeting in which we decided what were the 50 most important issues as encountered by the CJs that tested our product. We wrote these 50 issues on post-its and color coded them depending on in which sprint we wanted to finish them. To decide which issues would be tackled first we ordered them from important to unimportant and from easy to hard, so that every sprint we could find a balance in amount of issues and importance of issues to do. See figure 6.1 for the wall of issues we made.

7

Version 0.4: Visual Overhaul

	View reviews	Zoom photo	Change product characteristics (color / size)	View possible profit / benefit	View delivery time	View added costs (sending, returns, administration costs)	Search for saved rentepunten
Step Brief description of the action performed by the persona							
Scenario Description of the story behind the step	The JBL radio only contains 2 reviews. Tim asks himself if these reviews are reliable enough.	Tim wants to know in more detail how the JBL looks like. He enlarges the photo. Unfortunately, browsing through the photos is not possible.	Tim can see the JBL comes in black, white, red and blue. He picks the black JBL. Unfortunately, the color of the picture doesn't change.	Tim still doubts if this radio is not cheaper in another web shop. He can the Prijzvergelijken.nl	Tim doesn't know how fast the JBL could be delivered. He has a party this weekend, and he wants to play the music.	Tim views if there are costs attached to delivery. He also wants to know if returning the JBL is going to cost him money. He cannot find the costs immediately.	Tim wants to know if he can buy the JBL with his amount of rentepunten . He cannot see the amount of rentepunten he owns. He is not logged in.
Quote What is the persona thinking?	"Is this a good quality product?"	"I want to have a better look at the material of the radio"	"Did I select the right color?"	"Nice I can see the prices at other shops. This is a good deal!"	"Will the radio arrive in time?"	"Are there any hidden costs involved?"	"Can I buy this JBL with my points?"
Emotion The particular feeling of the persona at the step							
Pain Points Unpleasant experiences for the persona at this step	Reliability of reviews		Color and image do not match		Delivery time unknown	Unsuspected costs	Points not visible
Data Previous research results, current data to support the scenario / quote / pain	Quality research Sept 2016: Reliability is nr 2 why customers recommend the...	Research 2016 10 Steps For Better eCommerce Usability: At nr. 4 is Find the perfect zoom		Oct. 2016: 11% of the users click on the Prijzvergelijken link	Sept. 2016: Long delivery time is nr. 6 in the frustrations of Persona Tim	Sept. 2016: High delivery costs is nr. 4 in the frustration of persona Tim	

Figure 7.1: Screenshot of version 0.4 of *Mapp*, taken at commit 38e1585.

564 commits (162 added) | **9** front-end tests | **81%** back-end coverage (591/729 lines)

Main changes introduced in version 0.4:

1. Add different types to custom lanes
2. Implement lane deletion and drag-and-drop reorder
3. Change the styling to be in line with Material and ING design
4. Adding images to cards through drag-and-drop

In this sprint, we had very clear focus: based on our feedback from Customer Journey Experts during the midterm evaluation (Section 6), we knew exactly what features to prioritize. This is why we made journey mapping more flexible by making lanes deletable and reorderable (Section 7.2). We also did two big refactors to enable important future features: we refactored our front-end Polymer elements to be more general (Section 7.3) and our map to be built around flexboxes instead of HTML tables (Section 7.4). This allowed us to properly set up front-end testing as well (Section 7.5). We also had some time to work with a visual designer to apply Material Design principles to *Mapp* (Section 7.1) and to look at the feedback we got from the Software Improvement Group (Section 7.6).

7.1. Material Design Overhaul visual design

ING adopts the same standard for tools made for internal use as they do for external use with the idea that “if it is good enough for our customers, it is definitely good enough for our employees.” This means that *Mapp* should follow the same style guidelines as other ING applications. To make sure we follow the ING style, we sat down with designer Bas.

The main issue that Bas noticed was that we think about the journey map as a table of cards, but that visually there are no cards. To fix this we flipped the colors: the background is now grey and the cards are white. Also, we added small shadows to the cards, info bar and emoji to conform more to the material design principles. We increased spacing between cards and drew row lines on the background to further improve the visual hierarchy. We implemented some simple changes immediately, such as: ING font, ING colors and ING icons.

7.2. Journey Map Flexibility implementation

During our midterm user evaluation we found out CJE make very different kinds of customer journey maps. Particularly the order and types of lanes they use is very different (for more details see chapter 6). To allow each CJE to make the kind of map that they prefer, we added the following features:

- Lanes have a customizable name and description.
- Lanes can be added using a button that spawns a context menu with the options to add a Single-Card Lane, a Multi-Card Lane, an Emotions Lane, a Timeline Lane, or a Checkmark Lane.
- All lanes can now be deleted.
- Lanes can be reordered using a draggable handle.
- Lanes can be styled to fit what the CJE wants to convey using the properties: background color, icon, text alignment and text decoration.

Reordering lanes by dragging with shadow DOMs

During the implementation of above features we stumbled upon the difficulty of implementing lane reordering by dragging in Polymer. Polymer adds the contents of a custom element to a shadow DOM. While this is normally very useful as it encapsulates the element’s behavior it made dragging and sorting lanes very hard to implement as there was no Polymer solution that makes dragging elements possible and thus relied on *jQuery UI* [31] to achieve the desired effect, which is known to not work well with a shadow DOM.

In order for *jQuery UI* to still work with our polymer shadow DOMs we used some workarounds:

- First, as we wanted a button to be used as a handle for users to drag a lane, we needed to make this button visible to *jQuery UI*, even if it was hidden a few levels deep into the shadow DOM. We achieved this by defining the handle button element on the level that *jQuery UI* has access to and then passing it as a child to the lane elements which would, in turn, lane the button visible with the `<slot>` tag.
- Second, *jQuery UI* requires an implementation of a method that clones the element the user is dragging to create a *Phantom* element in order to give the user the feeling of dragging the original element. To combat this, we wrote a custom clone function that copies lane elements

and assigns some temporary data the original lane had, making it look like the original lane while not affecting the original lane.

In practice, dragging an element involves three steps:

- The user drags the handle of the element they want to drag, *jQuery UI* hides the element out of view.
- *jQuery UI* creates a new, *Phantom* element that looks exactly like the element the user originally wanted to drag and places it at the users' cursor.
- The user releases the mouse and dragging stops. *jQuery UI* deletes the *Phantom* element and makes the original, hidden, element re-appear at the position the user stopped dragging.

7.3. Lane Element Refactor implementation

When we decided which lane types to implement in version 0.3 of *Mapp*¹, we did so by creating separate Polymer elements for each lane type: there was a `lane-data`, a `lane-step`, a `lane-pain-points`, etc. Each lane type also had its own nested `cell` and `card`—a `lane-data` required a `cell-data` and a `card-data`, for example—which meant that we were creating a lot of elements on the front-end.

We soon realized, however, that many of these lanes were very similar and differed only in styling: a Data lane is really the same as a Pain Points lane with a different logo and background color, and a Step lane is really the same as a Scenario lane with different text formatting. This was an opportunity for abstraction and simplification.

For the Step, Scenario, Quote, Emotion Description, Pain Points, Context, Data, and Notes lanes, we created a single card class: the `CardRich`. Some of these lanes require several of such cards while others require at most one, so we created `CellMulti` and `LaneMulti` elements, and `CellSingle` and `LaneSingle` elements, respectively. Using CSS variables bound to Polymer properties loaded from the server, these lanes can still all be styled individually after the refactor.

This leaves us with the following lane types on the front-end:

- **Single-Card Lane:** Covers the Step, Scenario, Quote, Emotion Description lanes, as well as any user-added single-card lanes.
- **Multi-Card Lane:** Covers the Pain Points, Context, Data, and Notes lanes, as well as any user-added multi-card lanes.
- **Emotion Lane:** Covers the Emotion lane.
- **Phase Lane:** Covers the Timeline lane.
- **Checkmark Lane:** Covers the CX principles lane.

This refactor cleared out a lot of duplicate and boilerplate code from our front-end and also made our later steps of templating and user-defined lane styling much easier to implement. To see what these lanes look like when they are styled, see Table 5.1.

7.4. Flexbox Refactor implementation

Before this version, our map was implemented as an HTML `<table>` element. To stay component-based, however, we had to apply some quirks to this table. According to the W3C standards, a `<table>` has first-order children called `<tr>`s. These table rows, in turn, contain `<td>`s: Table data/cells. Now, because we wanted to split the responsibilities of `<mapp-lane>` and `<mapp-cell>` elements, this meant we did not have these first-order children. Instead we structured the DOM as in Figure 7.2a.

¹See Section 5.1 for a list of which different lane types *Mapp* supports.

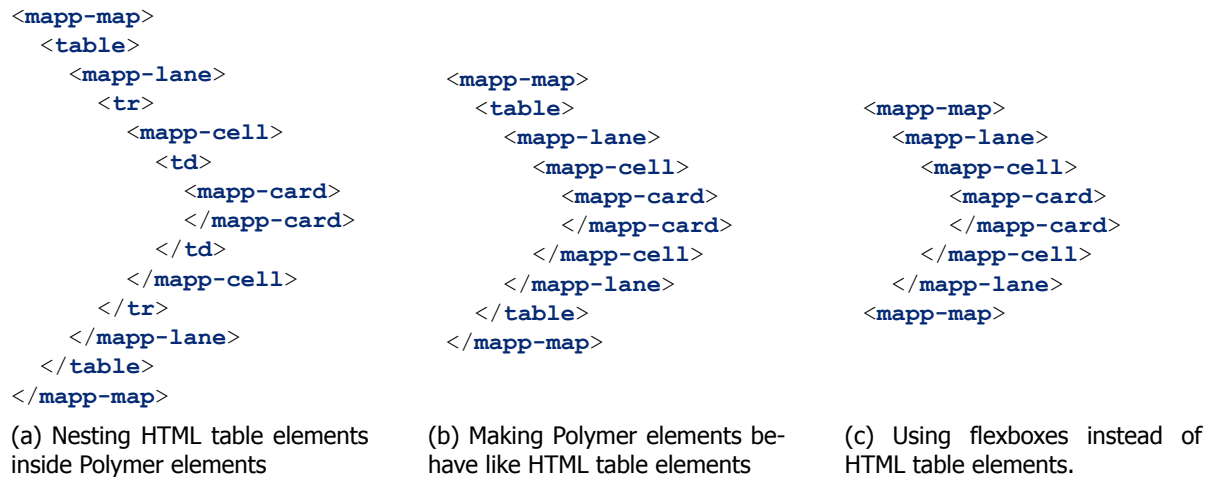


Figure 7.2: The evolution of the high-level DOM of our Map element.

To overcome this excessive nesting, we removed the `<tr>` and `<td>` elements and used the CSS `display` property to make the `<mapp-lane>` and `<mapp-cell>` elements behave like HTML `<tr>` and `<td>` elements, respectively. This made the DOM a lot more manageable, as in Figure 7.2b, but it proved to have one big disadvantage due of the way Polymer's `dom-repeat` element works.

The Polymer `dom-repeat` Element

The `dom-repeat` element binds to a JavaScript array that for every element in the array, Polymer dynamically inserts a child element into the DOM. When an HTML element like an ordered list or a table has a mix of both dynamically inserted child elements and hard-coded child elements, however, the order in which these are rendered is unpredictable.

This meant that at some point, it became impossible for us to nest the toolbar and lanes in the same HTML table element, which made it very difficult to keep them horizontally aligned². It was because of this that we decided to stop using the table element altogether and instead started using CSS Flexbox to layout the entire map. This gave us a fully trimmed DOM, as in Figure 7.2c.

Using `display: flex`, the `<mapp-map>` element would now vertically stack `<mapp-lane>` elements, which would in turn horizontally stack `<mapp-cell>` elements, which would vertically stack `<mapp-card>` elements without the use of Flexboxes, because we did not want the cards to have the same height (a handy property that comes along with the use of Flexboxes).

7.5. Front-End Testing software quality

While we can easily test the front-end code manually using `polymer serve` and chrome's devTools it is very time consuming and therefore only a small subset of features can be tested manually after a change. To enable automatic testing the polymer team have built the Web Component Tester (WCT) for end-to-end tests. It builds on top of some existing tools such as Selenium [30] (webdriver for different browsers) and Chai [32] (javascript assertion library).

We use the WCT to test for features of the front-end that might be broken after code changes.

We configure a `/test/` path that loads the test suite for each of the tested components and can then run them in a headless chrome window. As we decided chrome was the only browser that had to

²This was because, inside the HTML table, the toolbar was a hard-coded table row, while the Lane table rows were inserted dynamically.


```

C:\WINDOWS\system32\cmd.exe
> ing-app-mapp@0.0.1 test C:\Users\TN31CC\Documents\bep-ing\frontend
> polymer test --skip-selenium-install

Starting Selenium server for local browsers
Selenium server running on port 56050
chrome 67 Beginning tests via http://localhost:8000/components/in
g-app-mapp/generated-index.html?cli_browser_id=0
chrome failed to maximize
chrome 67 Tests passed
Test run ended with great success
chrome 67 (0/0/0)

```

Figure 7.3: Output of the selenium front-end tests (`npm test`), 9 tests are run in a headless chrome window and all 9 succeed.

be supported, testing is only done in a chrome window but the configuration can easily be changed to run on the browsers: Internet Explorer, Edge, Firefox and Opera as well.

A test suite is defined in a `.html` file that exists of a header that loads all imports, the element that is to be tested and a script tag that contains testing functions.

Figure 7.3 shows an example of the output of `npm test`. The test suites that are defined under `/test/index.html` are executed resulting in a total of 9 tests that all pass. To inspect test failures one could run the tests in a browser for the mocha interface as shown in figure 7.4. The mocha [33] interface can show for each test: pass/fail, execution time and the tests code.

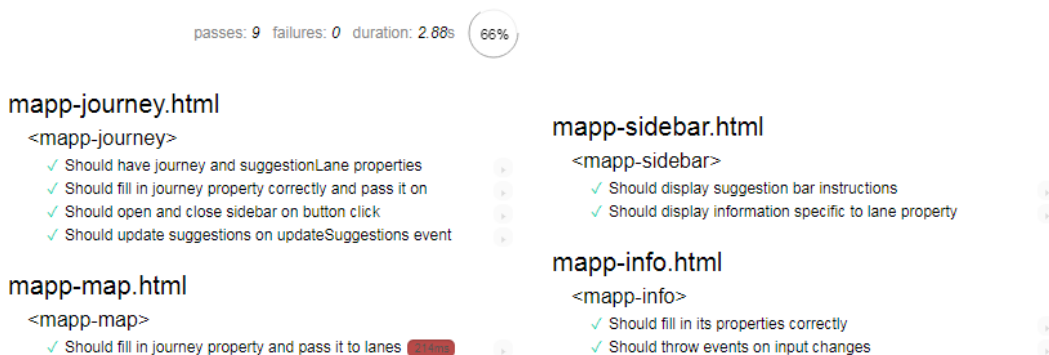


Figure 7.4: Output of the front-end tests run using the mocha interface, 9 tests are ran in the chrome window and all 9 succeed but one test takes exceptionally long.

Test Functions

Many of the tests follow a similar pattern to the one presented in Figure 7.5: first some testing data is defined, then some properties of the test element are set and finally changes are observed. Changes in the Document Object Model (DOM) are propagated asynchronously and many elements change dynamically so to make sure we only observe changes once the whole DOM is updated we use the `flush` function (line 12). The `flush` function is executed only after all observers have triggered. To prevent tests from influencing each other while they are waiting for their asynchronous changes to happen, the `done` function is used (line 1 and 16). Using `done` all tests run one after another and the next test only starts when the previous test has called the `done` function.

Tested Features

Metrics such as line or branch coverage are less interesting for polymer front-end code than for the java back-end. This is because bugs usually emerge due to mistakes in the interaction between elements than due to mistakes in an element itself. To still keep track of how complete our front-end tests are and what tests are more important than others we maintain a list of features that the front-end is

```

1 test("Should update suggestions on updateSuggestions event", (done) => {
2   testValue = {
3     displayName: "Test Name",
4     suggestionText: "This is a test suggestion text"
5   };
6
7   journey.sidebarRight = true;
8   journey.dispatchEvent(
9     new CustomEvent('updateSuggestions', { detail: { lane: testValue } })
10  );
11
12  flush(() => {
13    assert.equal(testValue, journey.suggestionLane);
14    sidebarLane = journey.shadowRoot.querySelector("mapp-sidebar").lane;
15    assert.equal(testValue, sidebarLane);
16    done();
17  });

```

Figure 7.5: A sample front-end test case.

supposed to have. Every test is supposed to test a single defined feature.

Below is a list of features to test on the front-end. Features that are already tested have been marked in bold.

- **Open and close the sidebar after clicking the arrow button**
- **Update the suggestion bar after a suggestion event**
- **Dispatch update events after changing info text fields**
- **Create the correct lanes after receiving journey**
- **Display correct suggestion bar contents**
- **Redirect to the overview page on logo click**
- **Card add button fires create event after click**
- **Card remove button fires delete event after click**
- **Lane add button opens menu and fires create event on click**
- **Lane remove button fires delete event after click and confirmation**
- **Card text box blur fires update card event**
- Step add button fires create event after click
- Step remove button fires delete event after click and confirmation
- Receive card create event adds card and selects it
- Receive step create event inserts a step
- Receive lane create event appends a lane
- Receive reorder lane event reorders lane
- Emotion slider release fires update card event
- Card menu button opens menu
- Emphasizing card updates its appearance and fires event
- Lane style button opens dialog
- Buttons in lane style dialog should fire lane style events

We have decided not to automatically test for real-time collaboration aspects as well as client-server interactions as it would require the coordination of multiple services simultaneously.

7.6. Software Improvement Group (SIG) Feedback software quality

We also received our feedback from the Software Improvement Group (SIG) during this sprint. We scored a 4.1 for maintainability and were provided some feedback for our code. SIG mentioned that

we did not score higher than a 4.1 due to the code scoring lower on the 'unit size' metric. This means that some methods are too long and thus harder to test and understand.

SIG mentioned the following:

- Some methods are too long in `StepServiceImpl`
- In `ResourceServiceImpl` the `storeBase64Image` should be split up into two methods: one method for validating the data and one method for storing the data.

We did not receive any other feedback other than the feedback mentioned in the last two paragraphs. However, with the feedback we did receive we took immediate action and resolved the problems SIG mentioned. That is, we re-factored some methods in `StepServiceImpl` and `storeBase64Image` now consists of two methods.

We did not receive our second round of feedback from SIG in time to incorporate it into this report.

7.7. Sprint Evaluation

In our evaluation of this sprint, we concluded the following:

- *Mapp* itself is already quite usable, but we need to build surrounding infrastructure before anyone can use it in production (e.g. an overview of journeys, creating new journeys, and exporting journeys).
- We're still missing the CX Principles and Timeline lanes.

8

Version 0.5: First Release

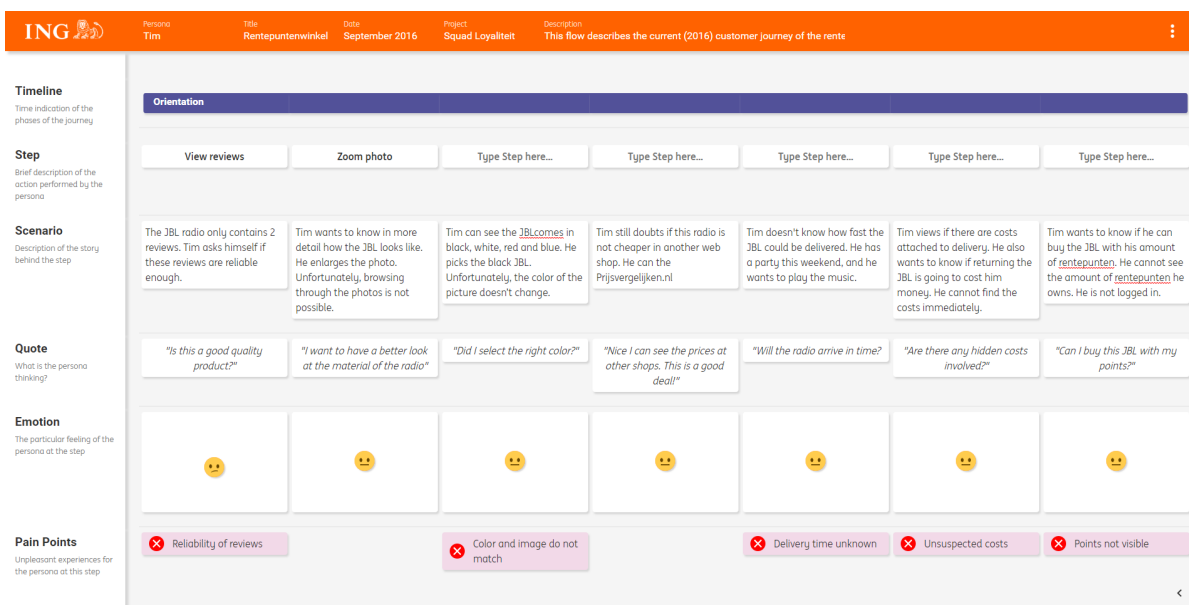


Figure 8.1: Screenshot of version 0.5 of *Mapp*, taken at commit `f1b18423`.

768 commits (206 added) | **16** front-end tests | **82%** back-end coverage (769/927 lines)

Main changes introduced in version 0.5:

1. An overview page to view existing journeys
2. Creating new journeys through templating
3. User-defined custom lane styling
4. Timeline and CX principles lanes
5. Exporting journeys to PDF and JSON, and importing them from JSON

In this sprint, we focused on adding functionality to round out *Mapp* and make it usable in production by adding things like a journey overview page, new journey creation from templates, and exporting a journey as JSON and PDF (Section 8.3). In this chapter, we also expand on some technical implementations such as our use of ING's Polymer elements (Section 8.1, the way the back-end persists data to the database (8.2), and the events API between our front-end and back-end (Section 8.4)

8.1. The Guide Components visual design implementation

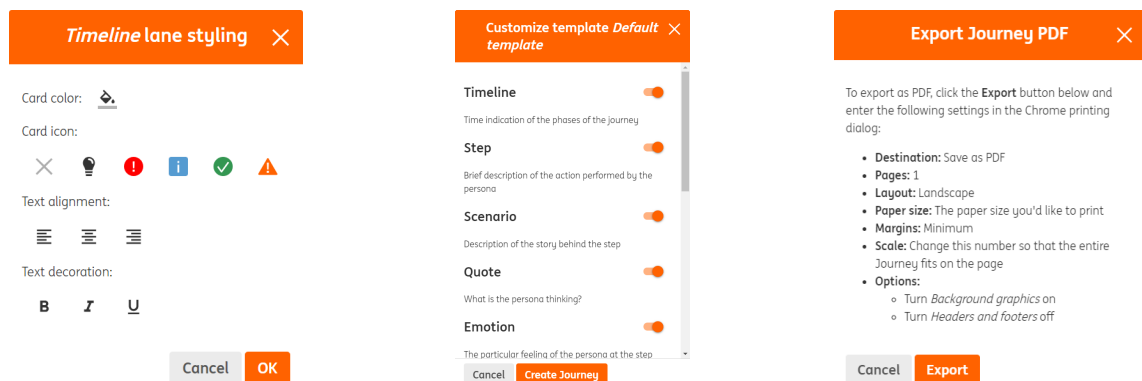
ING puts accessibility and security constraints on all apps. To still make it easy for developers to adhere to these constraints ING's *the guide* runs a catalog of polymer 2 components that are styled and programmed properly. For our release we tried to reuse as many components from the guide as possible.

ING Styling

A lot of styling has already been done by *The Guide*, this includes colours, fonts and other assets. This makes it much easier to give Mapp the look-and-feel of an ING application. The styling is imported into a Polymer components and automatically applied to most components such as buttons and text.

ING Dialog

We use the ING Dialog multiple times through Mapp as in some cases we require to prompt the user with multiple options inside a dialog.



(a) A dialog prompting the user which styles to apply to a lane

(b) A dialog prompting the user with journey creation options

(c) A dialog prompting the user with PDF export instructions

Figure 8.2: Examples of using ING's dialog component

ING Input

The ING Input components is a crucial component for our application as it allows users to edit text directly by clicking on the text.

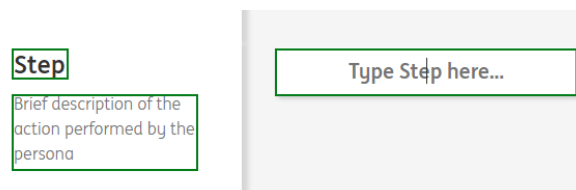


Figure 8.3: Three ING Input components, outlined in green

8.2. Back-End Architecture Changes implementation

Templates proved to be a harder to implement feature than initially thought. Conceptually a template is a collection of `Lane` entities. These lanes can then be used to define a new `Journey` object. A naive approach to accomplishing this is retrieving the collection of `Lane` entities that belong to a `Template` entity and then assigning them to a new `Journey` object. This has the undesired effect that, now, the `Lane` entities are owned by a `Journey` but also by a `Template`. As a result, changes made to the `Template` will be reflected to the `Lane` entities inside the `Journey` entity and vice versa. To solve this, every subclass of `Lane` must either override a `copy` method or fallback on the default `copy` method inside of the `Lane` class (this is due to the fact that some subclasses of `Lane` do not contain any fields and thus creating a new copy is the same as instantiating a new instance). The `copy` method must create a *deep copy* of the declaring `Lane`. Then, when a new journey will be created based on a template, the collection of `Lane` entities belonging to that template will be copied, persisted into the database and assigned to the journey.

Due to the fact that an overview page was a requirement for this sprint, we needed to configure some REST endpoints that would accomplish the following:

- **Fetching all rooms:** to give users an overview of their rooms
- **Fetching all templates:** to give users an overview of templates they can use
- **Creating a new room with a template:** to give users the ability to create a new room based on a selected template

8.3. PDF Exports implementation

Being able to export a PDF was a much desired feature as it allows Customer Journey Experts to easily share their Customer Journey. For PDF exporting there were two options to consider: let the client generate a PDF or let the server generate the PDF.

Server generating PDF

PDF generation on the server is complex. It would require the server to start up a web-browser in the background, connect to the correct Journey room, take screenshots of the web-page, moving horizontally over the web page to capture all elements, then, merge all screenshots and convert the screenshot to a PDF. We did not consider this option to be feasible as it was far too complex to implement in a single sprint.

Client generating PDF

Some libraries exist to generate a PDF based on HTML elements, however, as Polymer's shadow DOM elements do not work well with most libraries it was not possible to have a functioning PDF export feature.

However, we found that Chrome's print feature, when configured correctly, can produce great results with little development effort on our end. So, instead of letting the client or server do all the work we created a button that would prompt the user with some configuration suggestions and then display Chrome's printing dialog if the user selected the *Export* option (see Figure 8.2c).

8.4. Client-server events API implementation

When we started our project using a REST API we documented the API using a swagger specification [20]. Swagger can use the openAPI 2.0 and 3.0 standards to neatly specify API endpoints for REST services. However, since websockets are not a supported technology by these standards we define the API in our own way. For clarity we make a distinction between events that are pushed by the server (server-side) and events that are pushed by clients (client-side). Most events are pushed by the front-end after they are triggered by a user action, then processed and broadcasted by the server to all clients connected to the room. Events that behave like this are displayed in table 8.1. Some events

such as the initialize event are never broadcasted but sent to the server using HTTP and the response is sent to a single user with websockets, see table 8.2 for an overview of these events.

Event type	Description
cards.updateCard	Updated card data, cell ID and lane ID
cards.updateSelectionState	Selected (true / false), selector ID, card ID, cell ID and lane ID
cards.addCard	New card, lane ID, cell ID, preceding card ID and creator ID
cards.deleteCard	Card ID, cell ID and lane ID of deleted card
info.updateInfo	Updated contents of a field in the info bar
steps.addStep	Newly created step and ID of preceding step
steps.deleteStep	Step ID of deleted step
lanes.addLane	Newly created lane and its cells
lanes.updateLane	Updated lane data
lanes.deleteLane	Deleted lane data
lanes.orderLane	Reordered lane ID and preceding lane ID

Table 8.1: The different event types we implemented for the client and server side of *Mapp*. The Description column explains what content the event contains.

Events in Table 8.1 are grouped by which resource they concern and are prefixed by the name of that group e.g. `cards` or `lanes`. This naming convention makes it easy to see which client and server events perform a similar task. However, in hindsight we would slightly change the convention as it is hard to distinguish which events are created by the server and which by the client. A solution would be to name server events in the past tense as they are usually responses to client events. The server's `cards.updateCard` event would for example become: `cards.updatedCard`.

Event type	Description
<code>{roomId}/initialize</code>	Initialize a connection to the given room ID
initializeEvent	Room ID, user ID and journey for the session (only sent to one specific user)
POST <code>/api/images</code>	Image to be uploaded as base64 string
imageUpload	Link to the uploaded image

Table 8.2: The different event types we implemented for single client-server interactions of *Mapp*. The Description column explains what content the event or HTTP request contains.

8.5. Sprint Evaluation

We did not evaluate this sprint as we did the others because we evaluated version 0.5 of *Mapp* through our final user tests. Our conclusions from this evaluation can be found in Chapter 9.

9

Final User Evaluation

9.1. User Testing Round 2 usability testing

For the final user evaluation we used the same strategy we used for the Midterm Evaluation. We managed to schedule meetings with five users: three CJs and two UX.¹

The exercise for Customer Journey Experts:

- **Phase 1:** We asked CJs to recreate a small-scale journey they are accustomed to.
- **Phase 2:** When the CJs had questions we would answer them and if they couldn't find a function they were actively looking for we would point it out. At the end we would guide them past the functionality they might have missed.
- **Phase 3:** We asked CJs if they could use the tool as it currently is, and what they're still missing.

We conducted these user tests with Kyra Purmer, Stephanie Pelsmakers, Joeri van Raalte, Jorn Jokker and Eva Vriezokolk-Kraaijenbrink.

9.1.1. Kyra Purmer

Phase 1 As Kyra was very familiar with the tool due to weekly testing, she did not have much trouble creating a new journey from a template. She de-selected a few default lanes from the dialog prompt. She then created some cards in a single step, describing a step from one of her Journeys. She tried to copy a card to see if this feature was supported yet.

She noticed the CX Principles lane and mentioned that one of the names of the checkmarks was not correct. Also, after adding some cards to the CX Principles lanes she noticed a bug that caused old CX principles to not be completely deleted from the front-end memory.

After adding some more cards, Kyra tried to upload an image. As we only supported a selection of image formats and her image was not supported, no image was uploaded to the server and no image was displayed.

Phase 2 It wasn't clear to Kyra you can export the Customer Journey as PDF. After explaining how this could be achieved Kyra mentioned that it's great to be able to export as PDF and that the settings for Chrome should not be difficult for other Customer Journey Experts to figure out.

Phase 3 Kyra mentioned that, apart from the bugs, she could absolutely see herself using Mapp. When asked the question "What features are you missing that would be essential for you to work with Mapp", Kyra responded with:

¹The quotes in this section have been translated to English.

“ Nothing. ”

9.1.2. Stephanie Pelsmakers

Phase 1 Stephanie initially created a new Journey from a template, looking through the lanes she thought they were all sufficient. After creating the Journey she entered some basic information about her Journey: the persona and the title. After filling in this information she started looking around and added a phase card. She then entered a name for the phase card and wanted to expand the phase to the right:

“ I expect another step to be added automatically when I do this, it would be a nice to have feature. ”

She then inspected the options inside the left bar of lanes and clicked on “stylize lane” for a ‘step’ lane:

“ I like the fact that I can stylize my lanes. ”

She then tried to stylize the phase lane which did not work:

“ I am confused why I can style a phase lane ”

After adding multiple cards to cells and mapping her journey she mentioned how the application worked really well:

“ The application is very simple to use, which is nice! ”

Another nice to have feature Stephanie mentioned was the ability to add comments to cards:

“ It would be great if I could add comments to cards, as opposed to having a ‘notes’ lane ”

Stephanie mentioned that the CX Principles are undergoing a change: no longer are the values true and false used, but a score is assigned to how well journey adheres to the CX principles:

“ CX Principles are now indicated with numbers instead of true and false. ”

Phase 2 It wasn’t quite clear to Stephanie that you can add custom lanes, but she was satisfied with the set of lanes that were currently being used and thus she thought the feature was nice to have.

Another feature Stephanie did not notice was the ability to drag lanes. When mentioned she thought it was very useful to have.

Phase 3 Stephanie was very positive about Mapp, she can absolutely see herself use the application:

“ I can see myself and my colleagues using this application. ”

Moreover, when we initially spoke with Stephanie in the second sprint, Stephanie wasn’t sure if Mapp could ease the way Customer Journeys could be made at ING. After using the tool, however, she was convinced that the tool could help her and her colleagues:

“ Initially I was not convinced an application would be able to work well for customer journeys, but this has convinced me. ”

9.1.3. Joeri van Raalte

Phase 1 When Joeri wanted to create a new Journey he asked whether he had to press ‘default journey’, but when he did he knew how to go on. Once he created the journey, he filled in the top bar and asked whether it was customary to make one map per persona. He commented that a persona picker would be nice, and that the persona field could also be moved to the right. When he added a phase and deleted it, things were clear, but adding a step to actually extend the phase was not:

“ Adding those steps was not entirely intuitive for me, but in the end I found the way. ”

Joeri also soon found the `..` icon on the cards and played with options. He added and removed a highlight and tested the images upload by creating a screenshot in paint. He added it by dragging it from a folder, as the filepicker was slow:

“ Ahh cool! This is exciting. But now I have put it in it has shrunk like 20 times. ”

He tried to do the same but instead of an image he dragged a pdf, which did not work. Joeri scrolled on further but was unsure what to put into the data lane. He did get excited seeing the checkmark lane:

“ Hey CX principles, cool! ”

Joeri was similarly content with the emotions and phase lane. All in all he liked the ease of use of the special lanes, but there was still room for improvement here and there.

Phase 2 We showed Joeri how to properly extend a phase. He explained that because the lane was called `timeline` he concluded it must be about single timestamps. After that he also immediately spotted how to change the color of a phase:

“ Ah yes, to me this wasn't clear at all. ”

He thinks we could have chosen more descriptive names and proposed we might add a little info icon that you can hover over. We explained that this information is already available in the sidebar and Joeri was genuinely surprised to spot it:

“ That sidebar doesn't stand out at all. ”

When we showed Joeri that he could give the lanes a custom styling he wondered if he would ever change the background color of a card. Next, we showed him the `export to PDF` function, which he found very cumbersome. He also noticed that the sidebar was still visible in the PDF. Joeri also had two other suggestions for us:

- Adding an actual zoom button instead of solely relying on the browser zoom.
- A `coach mark tour` that guides you past the main functionality of the editor (as seen in digital 20, ING's new online environment).

Phase 3 When we asked Joeri if he could use the tool as is he replied with a clear `yes!`. Yes, some things are not entirely obvious yet but according to him, many issues are also a matter of getting familiar with the tool. If he were to use it he would first instruct his colleague and then see how far they can take it.

“ But in this shape I think it already works quite well ”

Joeri explains that he thinks the main benefits are that you can easily attach screenshots and he likes the feel of it, especially if you were to build a journey quickly and alone; as you would not have to do it on a giant brown paper. He did specify that he is not sure which method is better, though. Finally, he had two more possible contributions to make:

- In many lanes it does not make sense to have a symbol on every card in the lane. Perhaps it would be useful to add a symbol to a single card.
- For CX principles people have started working with `+++`, `+-` and `-` instead of a simple yes and no. Perhaps this could be worked into the CX principles lane.

9.1.4. Jorn Jokker

Phase 1 As Jorn is a UX designer we asked him not to construct a customer journey he was familiar with but rather to try to find his way around all the functionality of the tool. We had a shorter evaluation with Jorn than with the other users.

Jorn went through the tool from top to bottom, starting with the phases. After creating a phase card he clicked many times on the left and right arrow, to find out it did not do anything.

“ I think these arrows should not be visible if they will not complete an action. ”

Changing the text in the title bar also made Jorn point our attention to the text boxes. He showed that text can now overlap. Another issue arose when Jorn tried to change the text in a step card:

“ This placeholder text is so dark that it almost looks like regular text. ”

Jorn was happy to see how the emotion lane worked and found that the card interaction (adding, deleting, etc.) was intuitive. When he came to adding a new lane he noticed that there was again a small issue with the placeholder text:

“ The text that says ‘custom lane’ is not placeholder text, it is regular text. ”

Jorn liked how flexible the lanes were but steps appeared to be a little bit more fixed. The toolbar at the top sometimes disappeared and it is cumbersome to scroll up to add a step. Instead Jorn proposed to have a large button hover to the side of a step when the mouse is close to the side of one, to be able to add steps wherever you are on the map, vertically.

Phase 2 The main feature Jorn had missed was that of exporting to PDF. While it is a cumbersome function to use it is okay, as long as it gives the desired result.

Phase 3 Jorn thinks that:

“ Apart from some small possible improvements in placeholder text and button placement I think the tool is already very usable. ”

He was happy to see the UX improvements that we made and to see the interactions that the special lanes served.

9.1.5. Eva Vriezekolk-Kraaijenbrink

Phase 1 Eva had never seen *Mapp* before, and only discussed the idea for the tool with us conceptually when we met her at the beginning of our project.

She quickly found her way around the tool, and was generally very excited about the possibility of teaching Customer Journey Experts to use *Mapp* instead of the existing tooling in Microsoft Excel and Microsoft PowerPoint. Eva ran into a few pain points using *Mapp*, however:

- It was unclear to Eva how she could add steps beyond the first step; she expected to be able to click next to existing cards to add new cards instead of having to first insert a new step at the top.
- Before there were multiple steps, it was unclear how phases could expand.
- When adding a new lane, Eva had a hard time figuring out the difference between a single-card lane and a multi-card lane

Eva noted, however, that *Mapp* can be considered an *expert tool*, which means that most of these small UX issues are not a problem: once you see someone do it, you know what to do. In our discussion, we concluded that a short demo or training session should be enough to teach users these work flows in *Mapp*.

Phase 2 During this phase, we showed Eva some features she really liked, such as the ability to reorder lanes by dragging them. She was initially disappointed that our default customer journey map did not have lanes in the order that her Excel template has them; this resolved that issue.

Eva also noticed that we had no Moments of Truth lane; on this queue, we showed her that she could add her own lane for this and style it with a background color and triangle icon.

Phase 3 Eva definitely wants to use the tool we have built:

“ If I could, I would use it tomorrow—and show the CJEs at the training I’m giving. ”

She also noted that:

“ While there are some small improvements to be made this tool is so much better than what we have lying around right now. ”

In our discussion, we also agreed to make a template in *Mapp* that is exactly like her Excel sheet in terms of what lanes there are and what order they are in.

9.2. Summary usability testing

Overall, this final user evaluation shows that both Customer Journey Experts at ING, as well as their teachers and UX designers, are very excited about *Mapp*. They actually want to use the tool we built!

Most of the people we interviewed had some small improvements that they’d like to see, but they characterized them all as nice-to-haves: there were no show stoppers. These improvements include UX tweaks, new templates, and more fine-grained controls for the UX principles lane.

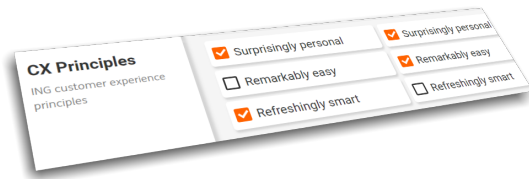
10

Release: *Mapp*

The screenshot shows the Mapp interface for a customer journey map. At the top, there is an orange header with the ING logo and metadata: Name: Tim, Title: Rentepuntenwinkel, Date: September 2016, Project: Squad Loyaliteit, and Description: This flow describes the current (2016) customer journey of the rents. Below the header is a navigation bar with a blue 'Orientation' tab. The main content area is a grid with seven columns representing steps in the journey. The left sidebar contains labels for 'Timeline', 'Step', 'Scenario', 'Quote', 'Emotion', and 'Pain Points'. The 'Step' row contains labels: 'View reviews', 'Zoom photo', and five 'Type Step here...' placeholders. The 'Scenario' row contains text boxes with descriptions of actions. The 'Quote' row contains text boxes with customer quotes. The 'Emotion' row contains seven yellow smiley face icons. The 'Pain Points' row contains five pink boxes with red 'X' icons and labels: 'Reliability of reviews', 'Color and image do not match', 'Delivery time unknown', 'Unsuspected costs', and 'Points not visible'.

Mapp offers Customer Journey Experts a way to collaborate in real-time on the customer experience design process. They can easily create and export customer journey maps that contain emotions, images, checkboxes, phases and text. Using templates and customization options customer journey maps can be fitted to the needs of any ING department.

Features

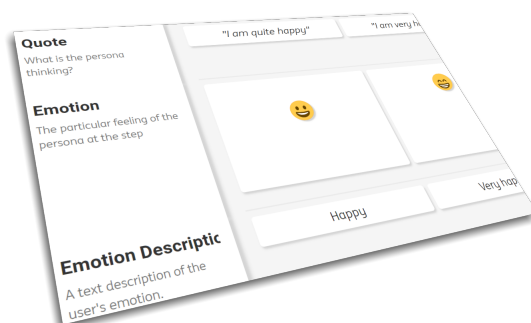
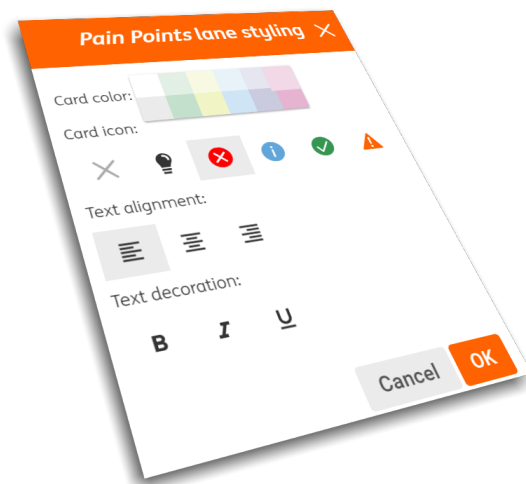


Custom lane styling

Use the lane styling dialog to take full control of your lane's styling. Add icons, change colors or emphasize a lane using formatted text.

Customer Experience principles

Easily define how your journey's steps conform to the *Customer Experience Principles* as defined by ING using the CX Principles lane.

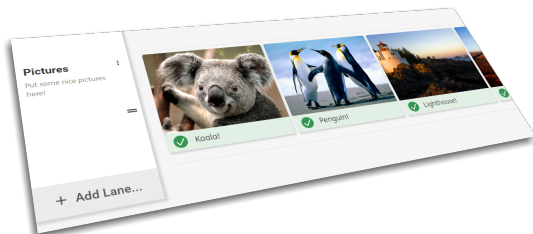
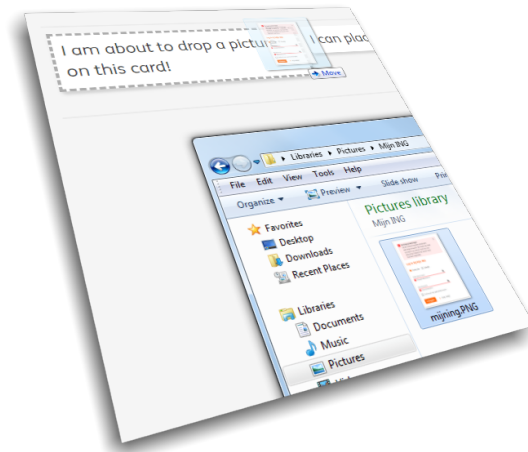


Emotion cards

Emotion cards allow you to express your persona's feelings as accurately as possible. Using the emoji as a vertical slider you can express emotions all the way from very angry to super happy.

Image dropping

Enrich your steps with more context by using pictures or images! You can quickly and intuitively add them to your cards by dragging and dropping them on top!

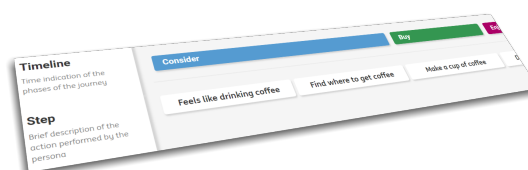


Images on cards

Images can be added to any text card and then further copied to other steps by dragging and dropping.

Phases lane

To illustrate where in the *loyalty loop* a customer journey lives make use of the Phases lane. This lane allows you to make a beautifully color coded time-line.





Overview and templates

So many URLs to remember? Don't worry. The overview page keeps track of all your customer journey maps so you can quickly access them. Want to create a new journey? The templates are defined right here.

Card options

More advanced card-options can be accessed using the card's context menu. You can highlight a single card when it is very important. Other options such as uploading images, removing images and deleting cards also exist.



11

Conclusion

Overall, we think we completed the Bachelor End Project very successfully. In just ten short weeks, we specified a project, got familiar with the ING work environment, and did five development sprints to implement *Mapp*, with two formal and many informal evaluation moments with Customer Journey Experts in between. In this conclusion, we'll briefly describe how well we feel we delivered *Mapp* as a useful tool; how we feel about *Mapp's* software quality; and how the process went in our view.

Product Quality

We are extremely satisfied with our final product quality. As Chapter 10 shows, *Mapp* looks and functions like a useful tool for Customer Journey Experts, with lots of handy features and customization. We're especially proud of the following features:

- **Real-time collaboration:** Every time we reveal to a user that we can edit the journey from our laptops while they're editing it at the same time, they're amazed!
- **Journey mapping flexibility:** The fact that there are multiple, customizable templates; and that users can edit lane styles, reorder lanes, and add/delete new lanes of different types means that *Mapp* is extremely flexible for the many different ways the different Customer Journey Experts work.
- **Drag-and-drop image placement:** Another *wow* moment for users is that they can simply drag and image from their desktop or from a folder into *Mapp*, and that it automatically uploads and attaches to the card they drop it on.
- **Emotions lane:** Users love dragging the emotion slider up and down and seeing the emoji attached to the slider change.

As Chapter 9 shows, Customer Journey Experts are also very excited about *Mapp*. Eva, who has trained over one hundred CJEs at ING and continues to train CJEs on a monthly basis, said "If I could, I would use it tomorrow—and show the CJEs at my training sessions!"

Software Quality

In terms of software quality, we're also very happy with our results in spite of our inability to use server-side continuous integration (see Section 5.3).

Front-End

On the front-end, Polymer proved to be a fantastic framework for *Mapp*: all our components are neatly separated and nested, and we were able to use components from ING's The Guide component library to build things like dialogs and text input boxes. In spite of the fact that four people worked on the

front-end code, we were able to keep code quality high using pre-commit hooks for linting and pre-push hooks for front-end tests. The front-end tests, which run using Selenium, test things such as:

- Whether the sidebar correctly expands and collapses when the user clicks the appropriate button.
- Whether steps can be deleted and whether a confirmation dialog is prompted before this actually happens.
- Whether the correct events are fired when the users edits a field in the `Info` element.

Although these tests cannot be run server-side, running them on our development machines before pushing commits to be merged with `dev` or `master` allowed us to make sure much of the front-end's core functionality worked before merging.

Back-End

On the back-end, Spring in combination with Hibernate were essential to successfully and rapidly, prototype the back-end server. The integration with WebSockets Spring offered allowed for the codebase to remain small (< 1000 LOC) and score high on maintainability. Writing small services for each functionality mapp required made it very easy to change, add and delete features to the back-end. Some team members mostly working on the front-end could easily add features to the back-end with little knowledge of the server architecture. As the codebase was relatively small and services consisted of small pieces of code, writing unit tests was not hard to do.

We are very happy with the tests we wrote that were capable of simulating a client connecting to the server, being able to fully inspect and verify all interactions with the server for most features we added.

End-To-End Testing

We were unable to do any end-to-end testing because of our client-server communication setup: since this all worked via JSON events, there was no OpenAPI / Swagger REST API specification that we could test against, for example. Instead of doing this in an automated way, we ensured that code on `dev` worked by manually testing commits to be merged. Of course, as with any manual testing, this meant that sometimes there were bugs on `dev` that we did not check for. Because of this, we spent the last few days of our last development sprint fixing bugs.

Process Evaluation

We believe that our agile, sprint-based process is the most important reason for our success in building *Mapp*. The constant feedback from CJs, both informally during each sprint and formally during the midterm and final evaluations, helped us prioritize what to build and what not to build.

Our weekly Skype meetings with Andy were also very useful in this aspect: decisions like switching to a socket-based architecture for real-time collaboration as soon as possible were only possible because we constantly got feedback.

Ethical Evaluation

Since our project did not interact with any sensitive or private customer data at ING, we believe the ethical implications of *Mapp* are minimal: our project simply makes an existing process more accessible and collaborative by putting it an intuitive online environment. Since *Mapp* also does not significantly automate any jobs at ING, we also do not believe our tool will cost anyone their ability to work or make a living.

Recommendations

We are convinced that our product has potential to have great, positive, impact on the way Customer Journey Experts work at ING. This claim is backed by Eva and Stephanie, who are both responsible for educating Customer Journey Experts and managing Customer Journey Experts respectively. We

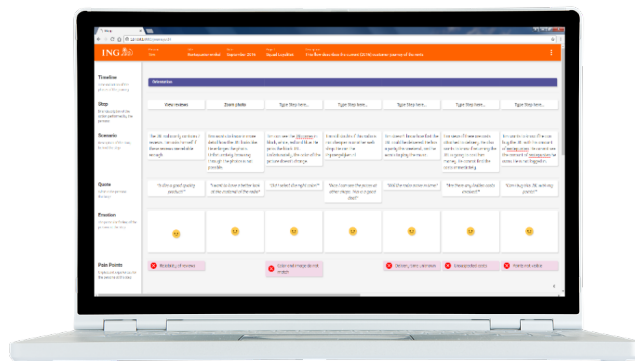
recommend that ING continues development on *Mapp* and further increase the usability of the tool by taking the feedback we gathered from numerous Customer Journey Experts into account.

One of our team members, Thomas, will continue to work at ING. We hope that Thomas will be able to maintain *Mapp* until resources become available for it to be maintained by another team.

Mapp: ING Customer Journey Mapping

In association with **ING Nederland**
Presentation on the **4th of July**

ING wants to offer their customers the best experience possible. To achieve this goal, ING's Customer Journey Experts (CJEs) constantly map and analyze the way customers use ING services in a Customer Journey Map. These maps however, are hard to share and collaborate on. ING needs an online tool in which they can, together with multiple people, build and maintain Customer Journey Maps. During our research phase we visited many different *squads* and found out that no single solution fits all needs. That is why we made our tool as customizable as possible with features such as: colors, text decorations, highlighting and templates.



We worked in bi-weekly sprints for which we selected work from a top 50 issues board that we ordered by importance and difficulty. The final product, *Mapp*, allows CJEs to define, share and collaborate on customer journeys. CJEs can illustrate their customer's steps using text, images, emotions, checkboxes and timelines. To share their work they can export as PDF and print in any size. And finally to collaborate they can simply share their journey's URL. The product was user validated during a large midterm and endterm test, as well as during short weekly tests. All of the chapter leads we talked to were super excited and are soon marketing the product in their teams!

Team

- **Thomas Kluiters:** I am interested in software quality and testing, Machine Learning, and back-end development
Main contributions: Back-end and back-end testing
- **Leon Overweel:** I am interested in Machine Learning and building online tools, and Polymer
Main contributions: Front-end data flow and lane styling
- **Daniël Vos:** I am interested in Machine Learning, software quality and security.
Main contributions: Front-end and front-end testing
- **Jelle Vos:** I am interested in Machine Learning, Computer Vision and Design.
Main contributions: Front-end and specialized lanes

Client: **Han Markslag, ING Nederland**

Supervisor: **Andy Zaidman, Delft University of Technology**

Contact: **Leon Overweel**, at l.overweel@gmail.com

Repository: <https://repository.tudelft.nl/>

Bibliography

- [1] K. Beck et al. *The Agile Manifesto*. Tech. rep. The Agile Alliance, 2001.
- [2] *One framework. Mobile & desktop*. URL: <https://angular.io/>.
- [3] *React*. URL: <https://reactjs.org/>.
- [4] *The Progressive JavaScript Framework*. URL: <https://vuejs.org/>.
- [5] *Polymer*. URL: <https://www.polymer-project.org/>.
- [6] *About the Polymer Project*. URL: <https://www.polymer-project.org/about>.
- [7] *RESTful Web Services in Java*. URL: <https://jersey.github.io/>.
- [8] *spring.io*. URL: <https://spring.io/>.
- [9] URL: <http://hibernate.org/>.
- [10] Stefan Bechtold et al. *JUnit 5 User Guide*. URL: <https://junit.org/junit5/docs/current/user-guide/>.
- [11] *Tasty mocking framework for unit tests in Java*. URL: <http://site.mockito.org/>.
- [12] *Testing*. URL: <https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html#integration-testing>.
- [13] The PostgreSQL Global Development Group. *The World's Most Advanced open source relational database*. URL: <https://www.postgresql.org/>.
- [14] *MongoDB for GIANT Ideas*. URL: <https://www.mongodb.com/>.
- [15] URL: <https://www.mysql.com/>.
- [16] URL: <https://www.sqlite.org/index.html>.
- [17] *The PostgreSQL Licence (PostgreSQL)*. URL: <https://opensource.org/licenses/postgresql>.
- [AGPL] Free Software Foundation. *GNU Affero General Public License*. Version 3. Nov. 19, 2007. URL: <https://www.gnu.org/licenses/agpl-3.0.html>.
- [18] *Transactional DDL in PostgreSQL: A Competitive Analysis*. June 2012. URL: https://wiki.postgresql.org/wiki/Transactional_DDL_in_PostgreSQL:_A_Competitive_Analysis.
- [GPL] Free Software Foundation. *GNU General Public License*. Version 3. June 1, 1991. URL: <http://www.gnu.org/licenses/gpl-2.0.html>.
- [19] *SQLite Copyright*. URL: <https://www.sqlite.org/copyright.html>.
- [20] *OpenAPI Specification*. URL: <https://swagger.io/specification/>.
- [21] Martin Fowler and Martin Fowler. "Service Layer". In: *Patterns of enterprise application architecture*. Addison-Wesley, 2015.
- [22] Henrik Kniberg. *Spotify engineering culture (part 1)*. Sept. 2014. URL: <https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/>.
- [23] *The WebSocket Protocol*. URL: <https://tools.ietf.org/html/rfc6455>.
- [24] *The simplest way to test and deploy your projects*. URL: <https://travis-ci.com/>.
- [25] *Jenkins*. URL: <https://jenkins.io/>.
- [26] *Husky: Git hooks made easy*. URL: <https://www.npmjs.com/package/husky>.
- [27] *ESLint: The pluggable linting utility for JavaScript and JSX*. URL: <https://eslint.org/>.
- [28] *stylelint*. URL: <https://stylelint.io/>.

-
- [29] Polymer. *Polymer Linter*. June 2018. URL: <https://github.com/Polymer/tools/tree/master/packages/linter>.
 - [30] *Browser Automation*. URL: <https://www.seleniumhq.org/>.
 - [31] JS Foundation. *jQuery User Interface*. URL: <https://jqueryui.com/>.
 - [32] *Chai Assertion Library*. URL: <http://www.chaijs.com/>.
 - [33] *the fun, simple, flexible JavaScript test framework*. URL: <https://mochajs.org/>.