INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

PFE

N° d'ordre : PFE-INFO-2016-26

Projet de Fin d'Études

présenté par

Maud LERAY

Élève ingénieur de l'INSA Rennes

Spécialité INFO

Année universitaire 2015 - 2016

Improvements and new features for the web application

Lieu du Projet de Fin d'Études

Konnektid

Tuteur du Projet de Fin d'Études

Simone Potenza

Correspondant pédagogique INSA Rennes

Bruno Arnaldi

PFE soutenu le 05/09/2016

Logo de l'entreprise :

KONNEKTid

Contents

1 Acknowledgements

First of all, I would like to thank Michel Visser and Simone Potenza, co-founders of Konnektid, for hiring me as a trainee in their company and for giving me the opportunity to learn web development.

I address special thanks to Simone Potenza, for his precious support as my supervisor, and for the variety of projects he allowed me to work on.

I am also very grateful for the good work atmosphere, the mutual aid, and all the nice moments spent together with the Konnektid team.

Furthermore, I would like to thank Bruno Arnaldi, my referent teacher during this traineeship, for his advising.

Finally, I thank my engineering school: the « Institut National des Sciences Appliquées » (INSA) of Rennes, and more specifically the Computer Science department, for the knowledge acquired during my formation but also for allowing me to conduct such an interesting internship.

2 Company presentation

Konnektid is a startup, founded in January 2013 by Michel Visser, the current Chief Executive Officer (CEO), with the goal of transforming neighborhoods into universities. A few months later, Simone Potenza joined the project as co-founder and Chief Technology Officer (CTO). The company's website, `https://www.konnektid.com/`, was live in April 2014.

2.1 Concept

Konnektid is a skill-sharing platform, empowering life-long learning and community building. It connects people who want to learn with people who can teach, and who live nearby. This last condition is what makes Konnektid special: people find each other online, but then meet face-to-face to exchange knowledge. These offline meetings are called « konnektions ».

The website features a slogan, which is « Everyone has a skill worth sharing », and a logo visible on figure 1.

Figure 1: Konnektid logo, representing conversation bubbles.

Two different kinds of members can be found in the community:

- Professional teachers, who pay a monthly amount for using the platform;
- Regular members, who use the website for free.

Professional teachers pay a monthly fee to use the platform, and get an public profile featuring their name and location but also their experience and certifications. They can create courses or workshops, which they charge for and are able to share on main social media. Plus, they have access to the community in order to find potential students.

On the other hand, regular members use the service free of charge. They own a private profile containing their name, their location and eventually some skills that they can teach. They can send general requests, notifying their neighbours that they want to learn something specific. If they see that somebody offers that skill, they can also send a personal request to that person. And if they are willing to pay, they can book courses or workshops offered by professional teachers.

2.2 Current situation

The Konnektid team has expanded, and it has changed a lot over the years with occasional freelancers and trainees. At the moment it consists of a dozen of people, from very different origins and backgrounds, some of them working part-time or remote.

The website has also grown, and it now counts more than 15.000 members. These users mostly live in the Netherlands, in the big cities such as Amsterdam, Rotterdam or Utrecht.

Konnektid's current office is located in the center of Amsterdam, in Rockstart's building. Rockstart [1] is a startup accelerator, meaning that they help startups kickoff, develop and grow through funding and mentorship. It results in a great community of people with the same interests in technology and entrepreneurship, who are willing to help each other.

The notion of professional teachers described in SUBSECTION 2.1 is quite recent on the website: it is part of Konnektid's business model. The company's major goal for this year is to test it, and to validate it in order to become sustainable. This requires the building of many new functionalities, which was one of the main goals of the internship detailed in SECTION 3.

[1] `http://www.rockstart.com/`

3 Goals of the internship

The main goal of the internship was the implementation of new pages and features for the platform, needed to prove Konnektid's business model. For this same purpose, some of the original elements required improvements or refactoring. From a personal point of view, these various tasks were meant to discover web development and the related technologies, which was the main objective.

3.1 Web development

Most of the achievements of the traineeship were directly linked to the website, so it was a great way to discover and learn web development. `https://www.konnektid.com/` is mostly built in JavaScript, HTML and SASS [2] so those programming languages, almost unknown at first, needed to be mastered. Although a major part of the assigned tasks were focused on frontend development, it was expected to work on backend duties from time to time, so in general to be flexible and eager to learn.

Konnektid uses some of the most recent web development and JavaScript tools available, such as ReactJS, Redux, Node.js, and more. They are presented in details in SECTION 4.

3.2 Company building

Being part of a startup team is a rich experience for many reasons, and one of them is learning the challenges of building a company. Even as an intern, it was expected to be part of the process, and not in a passive way: indeed, it is highly appreciated to be able to give ideas and opinions. This could happen at several occasions, including brainstorms, meetings and demonstrations of new features. The project management and team organization are explained in SECTION 6.

Moreover, as resources are limited and the number of employees is quite small, it was fast to be in the position of decision making. This teaches independence and autonomy, but also implies to take responsibility for choices and actions that were made.

[2] An extension of CSS, more powerful than the basic language [2].

4 The web application

This section describes the main characteristics of the website, and the technologies it is built with.

4.1 Single-page application

`https://www.konnektid.com/` is built as a single-page application (« SPA »). A SPA distinguishes from other web applications by loading one unique web page, and the totality of its necessary code, right from the start. After that initial load, all of the logic (presentation, business, routing...) is handled by the client side, in Konnektid's case through JavaScript. And when the user interacts with the page, only the affected parts of the user interface (« UI ») are updated.

This is achieved by using a specific architecture introduced on FIGURE 2, in which the data (« Model » layer) is separated from its presentation (« View » layer)[17].

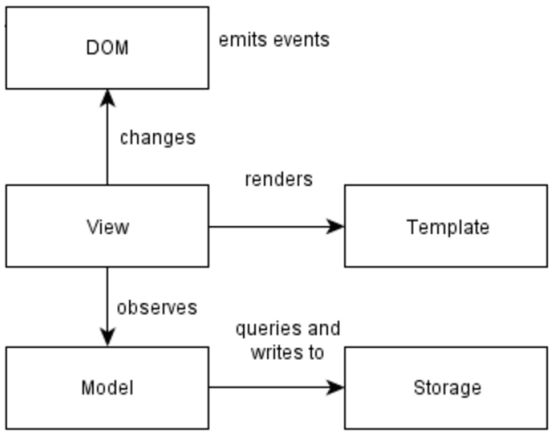


Figure 2: The usual structure of a single-page application.

A few more details about the different parts of the architecture:

- DOM [3] is write-only: no data is read from it, it just outputs HTML, operations and events;
- Models handles all the data and state of the application;
- Views update the UI, depending on DOM events and models, as pictured on FIGURE 3.

[3]Document Object Model: provides a representation of the document structured as a tree, with access methods.

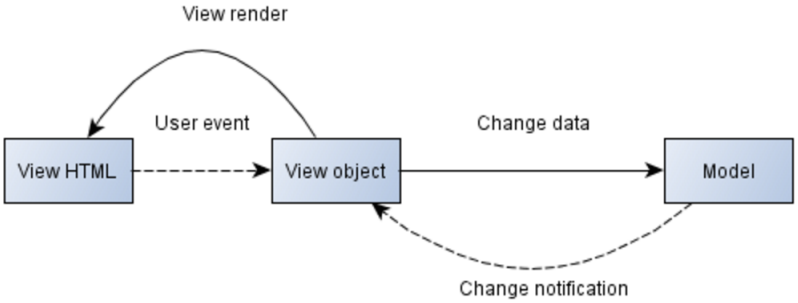


Figure 3: Possible interactions between the View and the other layers.

SPAs support communication between multiple components, each of them having intermediate states (menu open, item selected, etc), which is difficult to implement with other approaches such as server-side rendering. However, this affects the run-time state of the application (i.e. what the application looks like when it is running in a browser), as the whole state cannot fit inside one URL. For SPAs, the level of details in URLs is reduced, usually by focusing on the primary activity of the page and skipping the secondary activities (for instance modals) that will be reset to default state in case of reload.

By avoiding page reloads, SPAS bring a more responsive and fluid user experience and a better overall performance. There are various techniques and frameworks available for implementing SPAs. Among them, Konnektid uses several that are described in SUBSECTION 4.2.

4.2 Frameworks and libraries

The website is principally developed in JavaScript, HTML and SASS. It features a certain number of frameworks and librairies: the main ones are described in the present section.

4.2.1 ReactJS

ReactJS is an open source frontend JavaScript library, used to create reusable components that can be composed in order to build user interfaces. It differs from other libraries by its automatic way of keeping the DOM up-to-date. ReactJs implements a « virtual DOM », i.e. a lightweight representation of the view, from which the markup is created and inserted into the actual DOM. Then for each component, at initialization a *render()* method is called, that generates the corresponding slice of DOM. Then, whenever the data changes or events are emitted, *render()* returns a minimal set of modifications to apply to the DOM. This makes web applications much faster, and easier to build [7].

ReactJS components act just like functions: they take data as input and output the corresponding UI in return. They can manipulate two different kinds of data:

The state It is a set of internal data, that can only be updated by the component itself using the *setState()* method. All React components do not necessary have state: if they do, they are called « stateful » (or « smart »), in opposition to « stateless » (or « dumb ») for the others. Stateful components access their state via *this.state*.

The props These properties are passed down to the component by its parents, and can be accessed by calling *this.props*. They cannot be modified by the component, but to avoid issues if not provided it is possible to define default values.

So ReactJS can be summarized as: **component(props, state) => UI**

A good practice it to keep the number of stateful components as low as possible [9]. Konnektid follows this advice, by implementing most of its components presentational. When a state is really needed, a container is created on top in order to handle the logic. This parent can then pass properties down to the stateless component which only handles the rendering. This technique respects another recommanded convention: the single responsibility principle (each class or module should only have one task to handle).

Most developers (including the ones at Konnetid) use React with JSX, a preprocessor step that adds XML-like syntax to JavaScript. It makes the code easier to debug, and faster thanks to the optimization performed during compilation [10]. FIGURE 4 shows that code written in JSX is much more readable than raw JavaScript.

```
var Nav;
// Input (JSX):
var app = <Nav color="blue" />;
// Output (JS):
var app = React.createElement(Nav, {color:"blue"});
```

Figure 4: An example of JSX code and its JavaScript output.

4.2.2 Redux

Redux is a predictable container for JavaScript applications [11], in other words, it handles the logic and the data so that React components only have to deal with the presentation. It stocks the whole state of the application into one object tree called « store ».

The store provides methods to access the state (*store.getState()*) and to modify it (*store.dispatch()*). The second method takes a parameter, an « action », which is a JavaScript object describing what happened and containing at least a type (usually a constant). Actions are produced by specific functions called « action creators », as shown on FIGURE 5.

```
function addTodo(text) {
  return {
    type: ADD_TODO,
    text
  }
}
```

```
{
  type: ADD_TODO,
  text: 'Build my first Redux app'
}
```

Figure 5: An example of action creator and one of its possible outputs.

To specify how the application's state is transformed by actions, Redux uses « reducers ». They

are pure functions [4] with no side effects, that take the previous state and an action, and output an object containing the new state. The previous state is never modified, so that it can be returned in default case (e.g. for an unknown action). The Redux behaviour is illustrated on FIGURE 6.

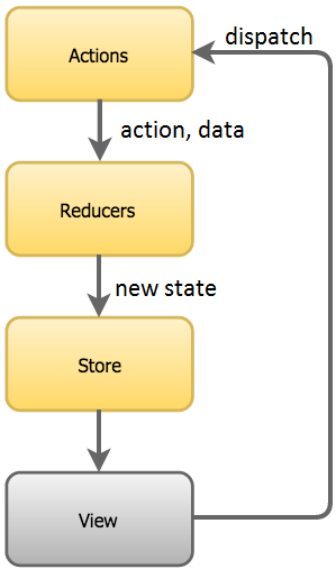


Figure 6: A simplified diagram of how Redux works.

To summarize, Redux works as follows: **reducer(previousState, action) => newState**.

4.2.3 Node.js

Konnektid's server-side code is mostly based on Node.js, an asynchronous event-driven JavaScript runtime. It is particularly efficient because it uses Google Chrome's open source V8 engine, that is able to quickly analyze and execute JavaScript code [3].

Node.js's non-blocking code architecture is one of the reasons why it is particularly fast: the program does not need to wait to finish an action before launching a new one [4]. This is achieved by associating each input/output operation (such as a database query) to a callback function that will be fired as soon as the operation is over. In between, the system can work on other tasks, as illustrated on FIGURE 7.

```
var callback = function (error, response, body) {
    console.log("Success!");
});

request("http://www.website.com/file.zip", callback);

console.log("I can do something else in the meantime...");
```

Figure 7: An example of non-blocking code with a callback function.

[4] Functions that always return the same result given the same argument(s), because they have no internal state.

Another strong suit of Node.js is its package ecosystem, *npm*, which allows developers to share solutions to particular issues [14]. Anyone can reuse these pieces of code in their own projects, then *npm* facilitates checking on updates that were made and downloading them.

4.2.4 GraphQL

GraphQL is used by Konnektid to communicate between the client and the server. To be understood by GraphQL, the queries must respect a specific format illustrated on the left side of FIGURE 8. They are then interpreted by the server and provide a result: an object containing all the required fields, similar to the right side of FIGURE 8.

Figure 8: An example of GraphQL query and its result.

There are a few advantages to use GraphQL instead of REST APIs [8]. First of all, as noticeable on FIGURE 8, the query is shaped just like the data it returns: as a hierarchy of fields. This makes it more natural to write and also easier to understand. Another particularity of GraphQL is that its queries are encoded in the client and not in the server, allowing to return exactly what the client asks for and nothing more. Moreover, GraphQL is strongly-typed: it controls both the correctness and the validity of the syntax of each query before executing it.

In the current section the main technologies have been introduced, now SECTION 5 describes the development workflow and the use of Github inside the Konnekid team.

5 Development workflow

To manage and host the code of the web application, Konnektid's developers use Github. It is an application built on top of Git, a distributed version control system popular for its branching and merging functionalities [5]. As Git is originally based on command line, Github provides a clearer interface to visualize all Git events such as commits, merges, and more.

Konnetid uses private repositories to store code, which are similar to public repositories except that they are not accessible to everyone: users need permission from the repository's owner if they want to see and modify its content.

5.1 Branching structure

The Konnektid repository is organized in two permanent branches, each of them with a specific configuration:

production contains the latest stable version of the application, deployed at `https://www.konnektid.com/`.

alpha is a pre-release of new features, useful to provide further tests, fix remaining bugs, and make sure that everything is compatible with *production*. When developers want to release, and if everything is approved and fully tested, they merge *alpha* in *production* and deploy. The rest of the Konnektid team also has access to this branch, deployed at `https://www.alpha.konnektid.com/`. This allows them to see and try out the latest functionalities, and eventually suggest some last-minute changes to be made (for instance in the copywriting).

When a developer wants to start building a new functionality, he/she branches out from *alpha* to create a local branch called « feature branch ». When the code is ready, this branch is merged in *alpha* and then removed. Feature branches should be short lived, ideally not more than a few days, otherwise they cause integration issues because of the tasks other developers finish and merge in the meantime.

Before implementing an important modification in the code (for instance, changing the general navigation), a « feature flag » is created. It is a boolean, defined in the configuration files, that decides whether or not the new feature should be displayed in the branch. This technique assures that the critical changes will not affect the other branches.

A code difference does not automatically move from *alpha* to *production*. First, it has to pass a certain number of tests before being committed and pushed. Then, it is manually reviewed: this is clarified in SUBSECTION 5.2.

5.2 Reviewing process

When a piece of code is ready to be merged, first in *alpha*, the developer who implemented it opens a pull request [5] for it, with a « state/need review » tag. Then, another member of the development team will take a look at the changes. This verification has three main goals:

[5] User-friendly interface for notifying others of the completed feature and discussing proposed changes.

- Verify that the changes fulfill their goal;
- Check the quality of the code (for instance the indentation);
- Make sure that the merge will not cause any conflicts.

If something is wrong, the reviewer replaces the previous tag by a new one, « state/need improvement » with a comment in the pull request to describe what needs to be enhanced. This allows the reviewee to efficiently iterate on it. Otherwise, if everything is fine, the reviewer adds a « state/approved » tag and merges the branch in *alpha*.

This process is rather efficient but very time-consuming, and not completely reliable as humans can make mistakes. To reduce this risk, Konnektid is introducing continuous integration (« CI ») with Jenkins, an open source automation server [12].

CI is a practice based on frequently integrating local code into the shared repository, where it is verified by an automatic build. It is meant to reduce integration issues, and if there are any it helps detecting them earlier. At the moment, CI in Konnektid is very basic and only performs a few tests on *alpha*, but the goal is to improve it step by step in order to achieve continuous integration, continuous delivery (« CD ») and fully automated testing. CD aims at safely releasing changes in *production* as soon as a pull request is merged, so that it is faster to get feedback from the users and therefore easier to update.

Github is a very efficient communication tool for developers, and it really facilitates team work and code sharing. However, there are also non-developers in the Konnektid team, who do not use Github but still work on the same goals and projects. SECTION 6 explains how this is managed, and how communication is handled.

6 Project management

Project management here refers to the overall organization of the Konnektid team.

6.1 Goal setting

Konnektid's objective for the year is to validate their business model. In order to achieve this, it is divided in monthly sub-tasks, called Objectives and Key Results (« OKRs »). Their progression must be easy to quantify (usually by a percentage) and they should be reasonable (feasible in one month) but also ambitious: a good OKRs score is around 70%. An example of OKRs for the developers could be to implement a new home page for the website: it contains many subtasks (e.g. create a mockup, implement a static version...) but it is considered 100% done only when fully functional and released in *production*.

Once per month, two OKRs meetings are organized with the whole team. The first one is an evaluation of the previous month's OKRs, so that everyone knows what the others have accomplished. If the OKRs score is low, this reunion is also a good opportunity to discuss it: what went wrong, why, and how to avoid it in the future. The second meeting is dedicated to the definition of new OKRs for the next month.

In addition, developers have weekly meetings to discuss their projects and code-related issues.

6.2 Tools

It has been explained in SECTION 5 that Konnektid manages its code on Github, and it is also an efficient communication tool for developers. But in order to interact with the rest of the team, two applications are used:

Asana, which facilitates defining tasks: they are easy to comment, divide in subtasks, share, assign to someone, mark completed... And it is possible to add files to them, which is especially useful for screenshots when it comes to design or frontend development. Then, tasks can be organized into projets, whose progress is visually clear and status update is simple [1]. Finally, Asana allows creating dashboards, that are groups of projects and/or tasks that belong together. Konnetid for example has a « Dev board », that gathers all the development tasks currently being worked on. Another interesting feature is that Asana can be used together with other services, such as Github for linking tasks to corresponding pull requests.

Slack, which is a chat system including file transfers, desktop notifications, and more. It also allows to create channels (public or private) that only alert the people belonging to them whenever a message is posted there. Just like Asana, it is possible to connect Slack to different tools [16]. For instance Jenkins, previously mentioned in SUBSECTION 5.2, can be integrated and then post notifications directly on Slack.

Now that the main techniques, tools and technologies have been introduced, SECTION 7 presents the most important achievements of the traineeship.

7 Internship achievements

This section describes the main tasks accomplished during the traineeship, along with the reason why they were needed, the way they have been implemented and the knowledge that they brought.

7.1 Building of UI components

As mentioned in SUBSECTION 4.2, ReactJS allows the creation of encapsulated components, most of them appearing several times on the website. They can be very basic (e.g. buttons) or more elaborate (e.g. modals). For this reason, it is a good idea to create a library of generic components, to be reused as much as possible. This saves a lot of development time, and also assures homogeneity between the different pages of the application.

Several developers, including myself, built this library in a folder named « ui ». One of the components that I have created is the date picker, for users to choose a date (day, month, and year) from a calendar. Since this is a rather complex component, I first performed a research on existing solutions inside the open source community. It is a great resource, as many developers publish efficient (and free) ReactJS plugins to solve most common issues. They are usually hosted on Github, with a demonstration link of how they behave and a fast downloading option via npm.

I found a few available date pickers, and in the end, I decided to use a plugin called *react-datepicker* [6], based on several criteria. One was verifying that the repository is still being maintained, for instance by checking the frequency of commits in the past few months. This generally means that the plugin is up-to-date with ReactJS's latest version, and that the owner can quickly help in case there is any problem or bug. Also, if the repository has a lot of stars, it guarantees that lots of people are using it and that they are satisfied. Another sign of a good-quality plugin is a low number of open issues in the repository.

Then, to start the implementation, I added a « DatePicker » folder inside the « ui » one. In this new folder, I first created two files: one named « DatePicker.js », which contains the code of the component, and one named « index.js » to export the component and be able to reuse it. At the top of DatePicker.js are a few *import* statements, as seen on FIGURE 9: ReactJS, the previously downloaded plugin, and then « DatePicker.scss ».

```
import React, { Component, PropTypes } from "react";
import ReactDatePicker from "react-datepicker";
import styles from "./DatePicker.scss";
```

Figure 9: The needed imports for the DatePicker UI component.

The last import file, located in the same « DatePicker » folder, contains local CSS styles for the date picker that will not be used anywhere else in the application. Local classes are applied to the component by using {styles.className}, where *className* refers to the name of the wanted class. Each class contains styles for a specific part of the component, assigned to local CSS variables, as visible on FIGURE 10 for styling a header.

[6] `https://github.com/Hacker0x01/react-datepicker`

```
.header {
    background-color: $datePicker-header-background-color;
    padding-top: $datePicker-header-padding-top;
    color: $datePicker-header-color;
}
```

Figure 10: An example of local CSS class for styling a header.

On FIGURE 10, *.header* is the name of the local class. Below it, in dark red are the CSS properties (color, padding...), set with local CSS variables (here displayed in orange, preceded by $). These variables are unique, and assigned to a value in a separate file called « _config.scss », located in the same folder. An example is shown on FIGURE 11.

```
$datePicker-header-padding-top: 0.8 * $unit !default;
$datePicker-header-background-color: $color-green-0 !default;
$datePicker-header-color: $color-gray-1 !default;
```

Figure 11: An example of « _config.scss » file for the previous *.header* class.

To set these values, Konnektid uses global CSS variables (e.g. *$color-green-0*) that are defined in CSS files at the root of the « ui » folder. They are default values meant to be reused in every component, to make the website more harmonious. *$unit* is also a default variable, corresponding to ten pixels.

This system of local CSS modules is easier to read, as all styles for a component are grouped in one place, and it also makes sure that changes made in the CSS for one UI component will not affect the other ones [15]. Using this, I could customize *react-datepicker*, and the final result is presented on FIGURE 12: it now features the Konnektid default font, and some recurrent colors of the application (specific green, gray and purple).

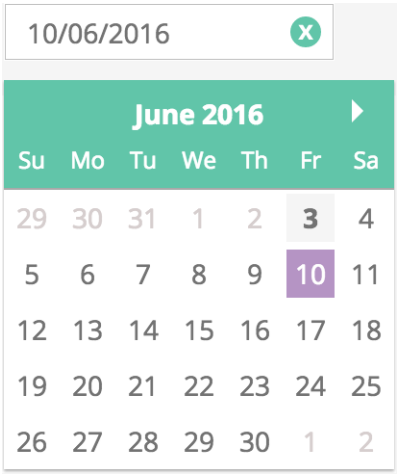


Figure 12: The final DatePicker presentational component.

This assignment was an efficient way to learn JavaScript and some good ReactJS practices.

Moreover, it was a great introduction to the ReactJS community and all the support that it brings, especially with the numerous free plugins that are shared.

It is important here to note that the pages that already existed on the website do not use this new library of UI components. Instead of rebuilding them all, which would have been terribly time-consuming, we were able to keep them in separate folders: « public » for public pages such as the homepage, « app » for private pages [7] such as the activity feed. They are now referred to as *old* pages whereas the most recent pages, built with the UI library, are called *new*.

7.2 Implementation of new pages

It has been explained in SUBSECTION 2.1 that Konnektid's main goal for the year is to test and validate their business model, which relies on professional teachers. This requires the implementation of many new pages and functionalities, and lots have been built during the internship. Among them, one will be described here: the course page.

It refers to the page used by teachers to create, edit and publish a course. It had to be built from scratch, for both desktop and mobile, based on designs made by Konnektid's former designer. It is now released and used by professional teachers, and an example is visible in ATTACHMENT B for desktop and in ATTACHMENT C for mobile.

The page features two main elements:

The top section which contains the main information about the course (picture, title, price...) and the teacher (avatar, name...). The teacher name is a direct link to his or her profile. This section also provides a button « Enroll now » for the students to book the course.

The bottom section which is divided in two parts. The right part takes most space on the page and gives an in-depth description of the course (methodology, requirements...). The left part, thinner, shows practical information about the course (format, length...) and another « Enroll now » button with a reminder of the price. Below this are the sharing functionalities: Facebook, Twitter, email, and WhatsApp on mobile.

The initial step in the implementation was to set up the course route in the router, in order to access the course page. Then in the application, when the URL for a course is matched in the router, the corresponding route handler is rendered and displays the « Course » component inside of the website layout. Later on, when the UI is finished and the logic is added using Redux, the route handler renders the « Course » container instead of the component.

To build the « Course » component, the first challenge was to decide how to divide it in subcomponents, each of them implemented in their own folder with local CSS styles as explained in SUBSECTION 7.1. For instance, the biggest bottom part shown on FIGURE 13 is a component called « CourseDescriptionCard ».

[7] Pages accessible only when the user is logged in.

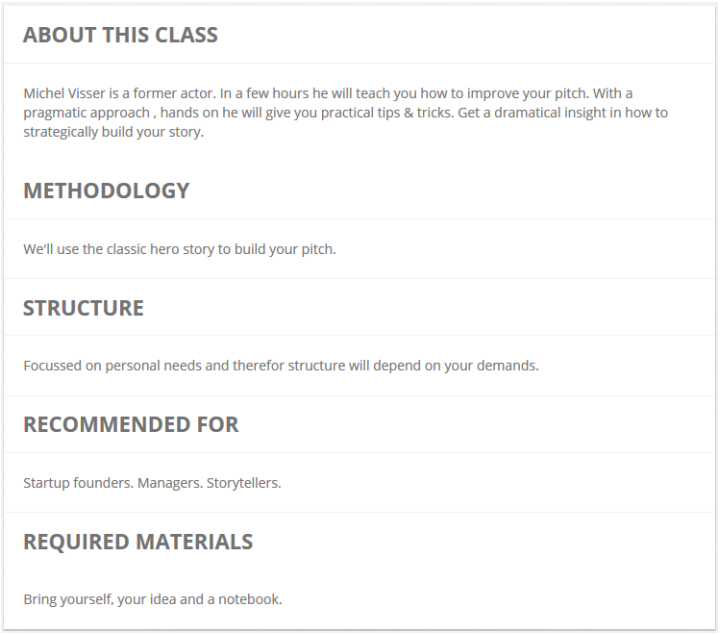


Figure 13: An example of « CourseDescriptionCard ».

We can see that it contains several sections, that are similar in presentation and only differ by their title and content. So it made sense to create a reusable « CourseDescriptionItem » sub-component, to render each of them with title and content passed down as properties. This reasoning respects the single responsibility principle previously mentioned in SUBSECTION 4.2 and avoids duplicated code. These are two of the many good practices for writing maintainable code [18], i.e. code that is easy to read, to modify and to extend.

After finishing the interface of the page, I was asked to add inline editing features to it. This means that elements can be edited in-place, in the context where they will be published. I used an editor framework for ReactJS called DraftJS [8], which enables the creation of rich content such as bold/italic text or lists of items. An example of editable « CourseDescriptionItem » is shown on FIGURE 14. When mousing over the content, a gray frame appears around it to draw attention, and the teacher can just click on it to start writing as explained by the green tip on top. For now it is empty, so the placeholder « If you leave this empty... » is displayed.

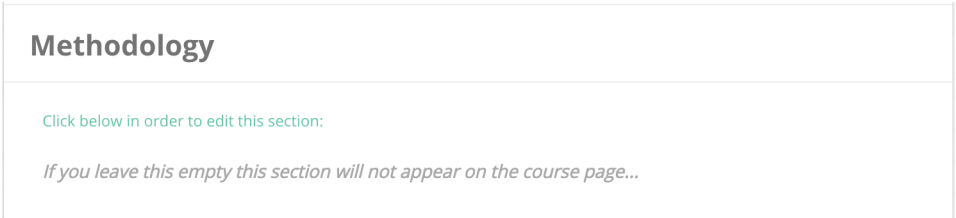


Figure 14: A « CourseDescriptionItem » in edition mode.

DraftJS is based on an « Editor » component, whose state is stored in an « EditorState »

[8] `https://facebook.github.io/draft-js/`

object holding all the information about the content: the text and its decoration (bold, italic...), the selection... A method *onChange*, passed down as property, is called to update the state every time the user changes the content of the editor. Moreover, it is possible to define customized key bindings, by using the *keyBindingFn* and *handleKeyCommand* properties. The first function returns a command string depending on the pressed key, then the second one takes that string as input and outputs the corresponding changes on the editor content.

This page was the first static page I fully created, and it made me realize how useful the components library mentioned in SUBSECTION 7.1 is. I had a lot of similar requirements during the traineeship, so at the end I was able to quickly deliver well-structured frontend pages, and even UI for entire flows such as booking a course with online payment.

7.3 Refactoring of existing pages

Adding funtionalities to the website sometimes involves updating old pages: for example, I changed the professional teachers presented on the homepage, in the « Book a professional teacher » section, to highlight the newly released teacher and course pages. In this case, revising the former code was enough. But sometimes, a whole refactoring is needed, and then the best option is to entirely rebuild the page using the UI library.

This is what happened for the teachers landing page, which describes Konnektid's offer for professional teachers in order to convince people to become one. It has been decided to improve it, because the conversion rate (i.e. the number of users signing up as teachers compared to the number of visitors of the page, obtained via the analytics) was not satisfying.

To get started, we had a brainstorm session to determine what contents were needed, and how they should be structured. We came up with a wireframe[9], and since there was no designer back then, this was the only support I had for development. So I quickly built and presented a first draft, to make sure I was heading in the right direction and to get a first round of feedback.

Based on this, I could iterate the process and gradually improve the interface until it was validated, for both desktop and phone. Then, to verify that we were sending the right message, the copywriting was reviewed together with the community manager. She also helped me out by collecting quotes from current professional teachers, who were then integrated in the page as social proof of the concept. This first version was released in *alpha*. A while later, a designer joined the team and we worked together on improving the User Experience (UX) and the UI of the page.

In the meantime, the registration process for professional teachers was also being refactored, but it was not yet ready. So I had to implement a temporary sign-up flow, very similar to the former one visible on FIGURE 15.

[9]Basic skeleton of the page, representing the main UI elements and how they work together.

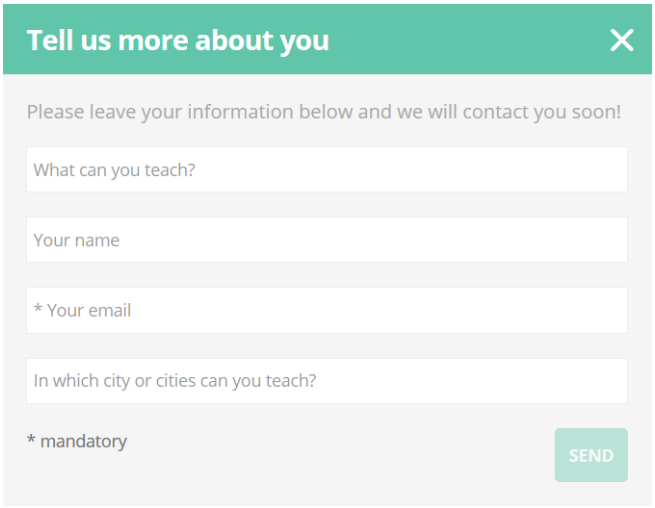


Figure 15: The former registration form for professional teachers.

It is a modal with four fields to fill in: the name and email of the applicant, the skill(s) to teach, and the city or cities where he/she could teach. When clicking on « Send », an email is sent to the Konnektid team with all the provided information. If everything goes well, a new screen appears to indicate the success, and the request can be treated by Konnektid (i.e. get in touch with the person, verify the certifications...). If there is a problem (for instance while sending the email), an error screen appears to inform the applicant that the request could not be sent.

I had to re-create this modal with the UI components of the new library, as a component called « BecomeTeacherModal ». It contains a form (« BecomeTeacherForm ») receiving the screen as property, which can have three values: *form* (default), *success* or *error*. It also has a container to manage the state of the modal (i.e. when to show it or not) and of the form (i.e. which screen to display) thanks to a Redux reducer that I implemented along with its tests.

The form itself was built with *redux-form* [10], a plugin specially designed for managing form state in React using Redux. I used it in the container on top of the « BecomeTeacherForm » component, to perform the validations of the form. For example, the email address is mandatory, and also needs to respect a certain format (e.g. test@test.com). The code of the container, testing the email value, can be seen on FIGURE 16 where *isEmail* is a utility checking the email format.

[10] `http://redux-form.com/`

```
export default reduxForm({
    form: "becomeTeacherForm",
    fields: ["applicantName", "email", "skills", "cities"],
    validate: ({ email }) => {
        const errors = {};
        if (!email || email.length <= 0)
            errors.email = "Please enter your email address";
        else if (!isEmail(email))
            errors.email = "Please enter a valid email address";
        return errors;
    }
})(BecomeTeacherForm);
```

Figure 16: The « BecomeTeacherForm » container, using *redux-form*.

Then, the error to display (if any) and the validation of the email can be used in the component, respectively via *email.error* and *email.valid*. The error will appear in red under the corresponding input, to inform the user that something is wrong and give improvement indications. The validation is used to able or not the « Send » action, as on FIGURE 15 where the button is disabled because the email field is empty.

The remaining task for this project was to create a GraphQL mutation to notify Konnektid of the registration request. To do so, I started by creating « notifyRegisteredTeacherMutation.js », which takes as input a « TeacherRegistration » object containing all the fields of the form and their values. When « Send » is clicked, the mutation is called in the « becomeTeacherModal » reducer, via the *completeRegistrationProcess* method visible on FIGURE 17.

```
export function completeConversionProcess(fields) {
    return {
        type: [
            COMPLETE_CONVERSION_PROCESS,
            COMPLETE_CONVERSION_PROCESS_SUCCESS,
            COMPLETE_CONVERSION_PROCESS_FAIL
        ],
        promise: client => client.mutate(`
            ($fields: TeacherRegistration!) {
                newTeacher: notifyRegisteredTeacher(fields: $fields)
            }
        `, { fields })
    };
}
```

Figure 17: The function handling the registration request.

This asynchronous function fires the *COMPLETE_REGISTRATION_PROCESS* action, and if the mutation is successful sends *COMPLETE_REGISTRATION_PROCESS_SUCCESS*, updating the screen to *success*. Otherwise, *COMPLETE_REGISTRATION_PROCESS_FAIL* is

returned and the screen becomes *error*.

Then, in « notifyRegisteredTeacherMutation.js », I had to handle three things:

- Internally send an email to the Konnektid team, to inform them of the registration request;
- Create a task in Asana with the applicant's data, in a dedicated project;
- Send an email to the teacher-to-be, as a confirmation that the request has been sent.

The final version of the new teachers landing page is available in ATTACHMENT D for desktop and in ATTACHMENT E for mobile. It was great to follow the creation process from the very beginning until the end, and to be fully responsible for it, even if it was challenging to build everything from scratch with technologies I had never used before.

7.4 New navigation

After discussions with the designer, it appeared that the navigation in the private pages of the application was unclear and too complicated. So, it has been decided to refactor it, and I took care of the implementation (both in mobile and desktop) based on design mockups. I will only describe the process for the old pages, as the one for new pages is basically the creation of UI components already detailed in SUBSECTION 7.1.

First, since this is a consequent change, I created a feature flag (technique described in SUBSECTION 5.1) named *new_navigation_enabled*. I set it to *true* in the *development* (for features branches) and *alpha* configuration files, but to *false* in *production*. This means that no matter what happens, the changes will only appear in *alpha* and will not affect *production* (the actual code of the website). I also added it to « legacyConfig.js », visible on FIGURE 18, which takes the configuration for new pages and converts it into configuration for old pages.

```
export default config => ({
    ...config,
    newNavigationEnabled: config.new_navigation_enabled
});
```

Figure 18: The content of « legacyConfig.js ».

This configuration can then be passed down to components as *config* property, thanks to a helper called *connectLegacyConfig*. The feature flag is then accessed via *this.props.config.newNavigationEnabled* and needs to be checked at every code modification, to be certain that changes will be applied only if this flag is set to *true*.

I started by updating the old pages, whose code is not as structured as the one for new pages, because it does not use presentational components nor local CSS styles. Instead, it features a lot of stateful components, and global CSS files stored in a separate « styles » folder. These files contain lots of classes, most of them being used in several components, and this really makes CSS styles harder to find and to modify, especially when somebody else wrote them.

In desktop, the former header (FIGURE 19) was relatively similar to the new one (FIGURE 20).

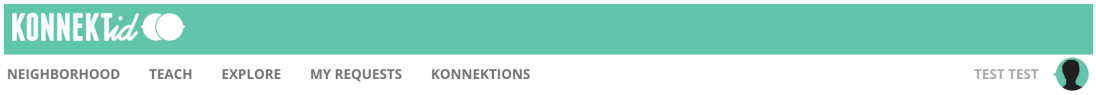


Figure 19: The former private header for desktop.

Figure 20: The refactored private header for desktop.

The old header was generated based on « private-menu.js », a file containing all its elements shaped as objects with two fields: the label to display (e.g. « My requests ») and the associated URL (e.g. « /requests »). I had to rebuild this file, as « new-private-menu.js », to remove unused sections and to include the icons, found in the Font Awesome library [11]. For this, I added to each object a field called « icon », as seen on FIGURE 21.

```
{
    url: "/teach",
    label: "Teach",
    icon: "graduation-cap"
}
```

Figure 21: An object of « new-private-menu.js ».

The icon itself is then defined for each menu item in a span HTML tag, by setting its className property (i.e. its CSS styles) as pictured on FIGURE 22.

```
<span  className={cn(`fa-${item.icon}`, "fa fa-lg")}/>
```

Figure 22: The definition of a menu item's icon.

A utility called *classnames* [12] (referred to as *cn* on the figure) is used here. It takes several CSS classes, and blends them into one, sometimes under certain conditions. Here for instance, it combines « fa » which points out to the Font Awesome library, « fa-lg » which defines the size of the icon, and *fa-${item.icon}* which indicates the icon to display (it would be « fa-graduation-cap » for the object on FIGURE 21).

Once this was done, I had to implement the « My activities » header section, and its dropdown visible on FIGURE 20. I created a new component for it (« KndActivitiesMenu ») and a CSS class (*activities-menu*) to place it relatively to the last header section. This also involved a few changes

[11] `http://fontawesome.io/icons/`
[12] `https://github.com/JedWatson/classnames`

in the last section's CSS class (*navatar-menu*). Then, I included the activities menu inside the « KndPrivateHeader » component, as seen on FIGURE 23, where *renderActivitiesMenu()* renders the « KndActivitiesMenu » component.

```
{config.newNavigationEnabled &&
    <div id="activities-menu">
        <a className="toggle" onClick={this.openActivitiesMenu}>
            <span>My activities</span>
            <span className="fa fa-lg fa-comment-o"/>
        </a>
          {this.renderActivitiesMenu()}
    </div>
}
```

Figure 23: The rendering of « My activities » menu.

Finally, KndPrivateHeader is stateful, and handles by itself the logic of showing (or not) the dropdowns it contains. It already had in its state a variable named *showProfileMenu*, used to handle the profile dropdown. This variable is updated by calling either *closeProfileMenu()* or *openProfileMenu()*, fired when clicking on the user's name. To be able to toggle the new activities menu, I extended the state with *showActivitiesMenu*, and implemented the related methods *closeActivitiesMenu()* and *openActivitiesMenu()*.

After updating the desktop navigation, I worked on the mobile one (still in old pages). This time, it was completely different from the old version: it used to be a burger button on the top left, toggling an offcanvas menu with all the possible sections. Now it is a header with full-width dropdowns, as shown on FIGURE 24. I thought this would be harder to implement than the desktop version, but in the end it was actually faster because it did not involve a lot of modification in the code, but mainly addition.

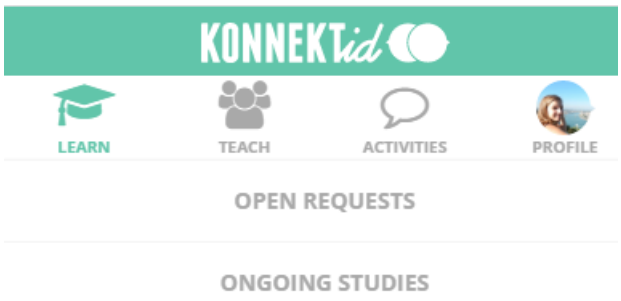


Figure 24: The new private navigation for phone.

First, I created a static version (« KndPrivateMobileMenu »), without the dropdowns, and its corresponding CSS class (*mobile-main-menu*). It was then integrated to « KndPrivateHeader » via *renderMobileHeader()*, with a previous check of the flag, similarly to the activities menu previously described for desktop.

Then, I built the « Activities » and « Profile » dropdowns, respectively added to « KndPrivate-Header » via *renderMobileActivitiesMenu()* and *renderMobileProfileMenu()*. Since they both show up full-width under the header, I created a wrapper CSS class (*mobile-menu-wrapper*) to define this behavior. Finally, I had to expand the state again, by adding the *showMobileProfileMenu* and *showMobileActivitiesMenu* variables, along with their corresponding opening and closing functions.

I was responsible for this new navigation project, which means that I had to keep the other developers informed of the progress by regular status updates. Once it was finished, I organized a demonstration for it, so that the whole team could see it and give feedback. Some of the suggested changes were then implemented. This experience taught me to take responsability for the code I produce, and to accept critics on it despite the time spent to write it.

7.5 Web analytics

Web analytics consist in collecting data about a website's usage (e.g. which buttons are most clicked) and traffic (e.g. how many page views per day). Konnektid mainly uses them as a market research tool, to gather information about the users and their experience with the web application. Almost all possible interactions on the platform are tracked and analyzed, to determine where improvements are needed but also to provide statistics as proof of the business model.

When it is decided to refactor an existing feature, analytics help to test whether the changes actually had an impact, and if it is a positive one. This process can be iterated until satisfying results are obtained. Also, after the addition of a new functionality, analytics provide valuable insights on how often it is used, how well it achieves its goals, and how it should be enhanced.

During my internship, I had to implement the analytics in the new pages, from a list of all the related events that needed to be tracked. I will here describe the building process for only one of them: make a booking from a course page.

Analytics in new pages are closely related to the Redux actions introduced in SUBSECTION 6.2. Indeed, when a user interacts with the website, a corresponding Redux action is created and along with it an event is fired. For example, if a member decides to book a course with a professional teacher, he confirms it by clicking an « Enroll » button. As soon as this is done, the function *completeEnrollProcess* from FIGURE 25 is called.

```
export function completeEnrollProcess({ courseId, courseTitle, coursePrice, teacherId, teacherName }) {
    return {
        type: COMPLETE_ENROLL_PROCESS,
        data: {
            courseId,
            courseTitle,
            coursePrice,
            teacherId,
            teacherName
        }
    };
}
```

Figure 25: The action creator called when a user books a course.

This function is a Redux action creator: it returns a Redux action, containing a type (*COMPLETE_ENROLL_PROCESS*) and some data (course title, teacher name...). The action is then mapped to the analytics by the *mapActionsToAnalytics* function, that I had to implement. It takes the action as input, and outputs an object whose content depends on the action type. The part which handles the course booking action is visible on FIGURE 26.

```
const mapActionsToAnalytics = action => {
    switch (action.type) {
        case COMPLETE_ENROLL_PROCESS:
            return {
                method: "events",
                args: {
                    category: "Course booking",
                    action: "Make booking",
                    value: action.data.coursePrice,
                    data: action.data
                }
            };
        default:
            return null;
    }
```

Figure 26: The function mapping Redux actions to analytics.

The returned object contains two fields, the method (always « events » no matter the action) and the args. The args represent the data needed for the analytics, that will be displayed and analyzed by two tools:

Google Analytics offering numerous services: calculation of conversion rates, estimation of

sales, information on used devices... Plus, it provides help for marketing and advertising [6].

Mixpanel focusing on funnels, i.e. visual representations of how users move through a series of events. This aims at finding out where users abandon and why [13].

In old pages, analytics sometimes need to be improved, either when an extra information requires measurement or when an existing functionality has been modified. This was the case when I added an « Invite friends » call-to-action (« CTA ») after a member offered to teach somebody (screen on FIGURE 27).

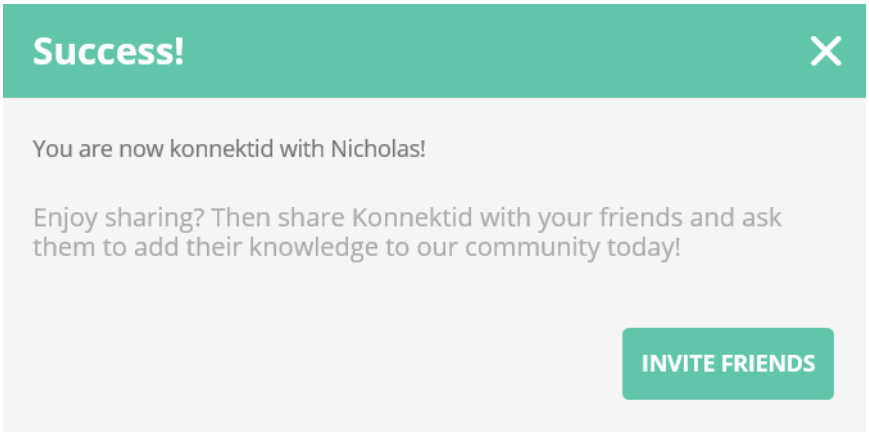


Figure 27: The screen appearing after making an offer, now with an « Invite friends » button.

Konnektid wanted to track the clicks on this new CTA, to compare its performance with the ones of other « Invite friends » buttons dispatched on the website. So, I had to figure out how the analytics are fired in the old pages, that do not use Redux.

Instead, a helper called *connectAnalytics* passes down the *analytics* property to the component, that is then accessed via *getAnalytics()*. It contains a certain number of methods, each one corresponding to a possible event. In this case I used an existing one, *inviteFriendsClicked*. It takes as parameter a constant representing the source of the event, i.e. from where in the application it has been fired. The possible values for this constant are defined inside the « constants » folder, in « InviteFriendsConstants.js ». This file contains all possible sources for the Invite friends event, so I added *AFTER_OFFER* to it, as visible on FIGURE 28.

```
Sources : keyMirror({
    GETTING_STARTED_FLOW : null,
    HOME_PAGE  : null,
    UNLOCK_YOUR_CITY     : null,
    AFTER_OFFER : null,
    AFTER_EMAIL_OFFER : null,
    DIRECT_ACCESS : null
}),
```

Figure 28: All the possible sources for the « Invite friends » event.

I could then define *inviteClicked()* (FIGURE 29), which is fired every time a user clicks on « Invite friends » after making an offer. It sends the event to the analytics.

```
inviteClicked() {

    // send the event to analytics
    this.getAnalytics().inviteFriendsClicked(InviteFriendsConstants.Sources.AFTER_OFFER);
}
```

Figure 29: The *inviteClicked()* method, used to send analytics.

Analytics are very important for Konnektid, and for web applications in general, so it was great to be able to manipulate them and understand how they work. It was a whole new concept to me, and I was impressed to see how they are used, not only to enhance the website's UX/UI but also to prove the business model. Implementation-wise, it was especially interesting in the new pages, because using Redux for analytics is a rather modern technique.

7.6 New application

At the beginning of August, it has been decided to split the Konnektid application into two websites: one focused on free use and community building, and the other containing the market place (i.e. professional teachers and course offering) for members willing to pay for learning.

A whole new Git repository has been created, with all the required infrastructure, including the UI components library. Some of my last achievements as a trainee were frontend pages for the new website, such as the homepage (ATTACHMENT F). I also implemented user interfaces for entire flows, like « Find a teacher »: it is a new functionality, proper to the market place, which allows members to specify what they want to learn and how (for instance, private or group classes) so that Konnektid can find a suitable teacher for them.

Technically speaking, these tasks were not particularly challenging as they were similar to the ones described in SUBSECTIONS 7.1 and 7.2. But the creation of this second website made me realize how fast startups evolve, and how decisive choices must sometimes be made when building a company. When working in such a dynamic environment, it is important to be adaptable, creative, and not afraid to try out lots of solutions before succeeding.

8 Conclusion

This traineeship has been a very rich experience, for many reasons that have been detailed in the previous sections and that are summarized here in the general conclusion.

Technically, it brought me a lot of knowledge in a discipline that I had not experienced before: web development, and more specifically single-page applications. I have improved my skills in languages such as JavaScript, HTML and SASS, and understood the concepts of frontend and backend. I learned the main good practices and principles for writing clean, understandable, maintainable code, and got to use some modern frameworks and libraries (ReactJS, Redux, GraphQL and more). This increased my curiosity about new technologies, and I got the chance to attend some very interesting conferences about them.

Moreover, I now know better how to use Github as a communication tool for technical issues, and how to deal with local branches, pull requests and multi-environmental development.

Being part of a small team made me realize the importance of communication and collaboration in project management. I often worked with the designer and the community manager, which was also the occasion to learn more about their job and skillset. And since Konnektid is currently trying to prove their business model as a market place, I discovered the techniques that you can use to do so: web analytics, marketing tools, etc.

From the company point of view, I believe that I produced a lot of value in terms of new features and improvements in the web application. I mostly worked on frontend tasks, and was quickly able to deliver interfaces and components with good quality code and intelligent structure. But I was also able to work on other assignments, involving different technologies and sometimes backend problematics. I learned how to be independent and was able to decide by myself which task to start next, depending on the month's objectives for Konnektid. I was also involved in brainstorming sessions and meetings where my opinion and ideas were truly appreciated.

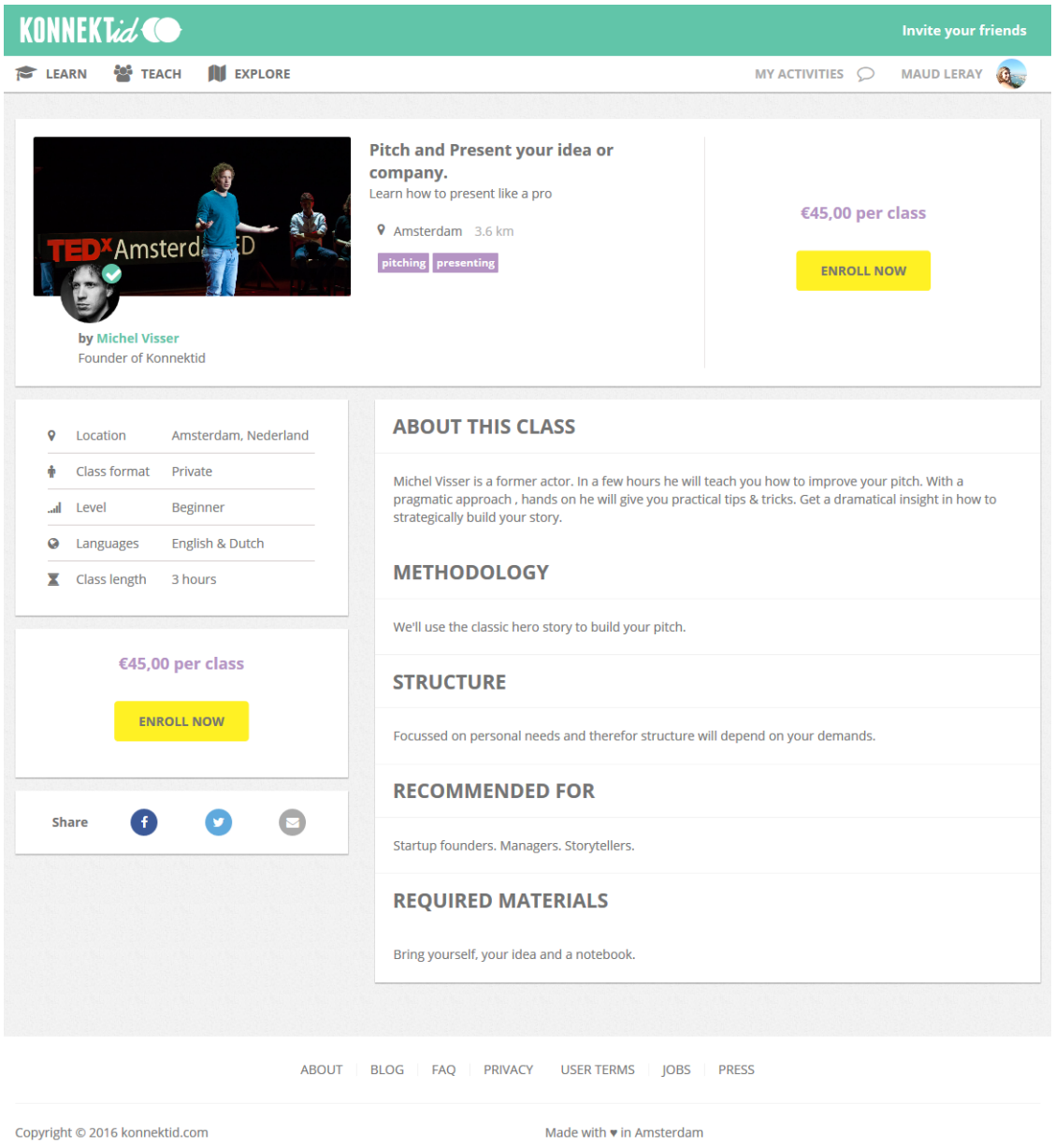
As a conclusion, the most valuable asset for me was to deal with such a huge variety of tasks, from building interfaces to implementing analytics or creating wireframes. It really made me think about what I like, and what I want to do after the internship.

A Global planning

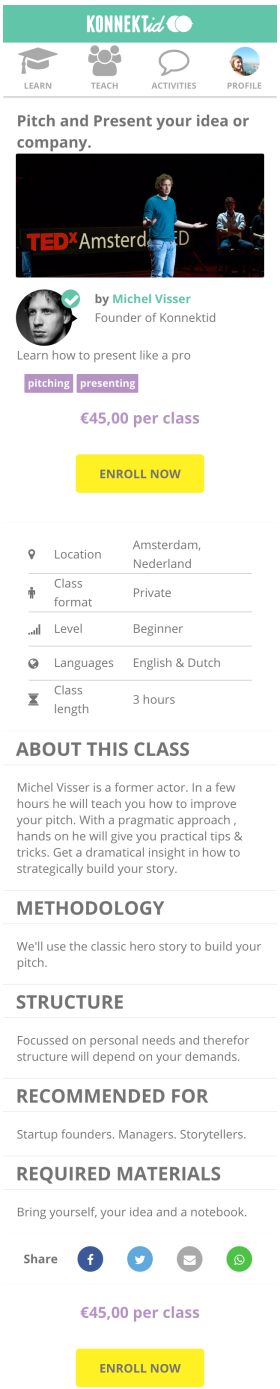
TABLE A presents an overview of the main development achievements of the internship, per month. Many smaller tasks have been omitted, as well as the meetings, demonstrations and brainstorming sessions that were also an important part of the planning.

March	Setup and tool discovery
	Lots of small tasks (copywriting, debugging, emails improvement...)
	Addition of scrolling button in activity feed
	Privacy statement update
	Creation of User terms page
April	Creation of Unlock your city email
	Account deletion modal improvement
	UI components building (Label, Alert...)
	Course page UI implementation, with inline editing
May	Implementation of reviewing process for teachers
	Handling non-existing pages and URLs
	Global public/private layout refactoring
	Mobile menu fixing (close on outside click)
June	Wireframing for matchmaking flow
	UI components building (date and time pickers)
	Overall navigation refactoring
	Analytics addition to new pages
July	New teachers landing page creation
	SEO improvement for /courses page
	Booking flow with payment UI building
August	Administrator UI implementation
	Temporary sign-up flow addition to new teachers page
	Modal UI component refactoring
	Find teacher flow UI building

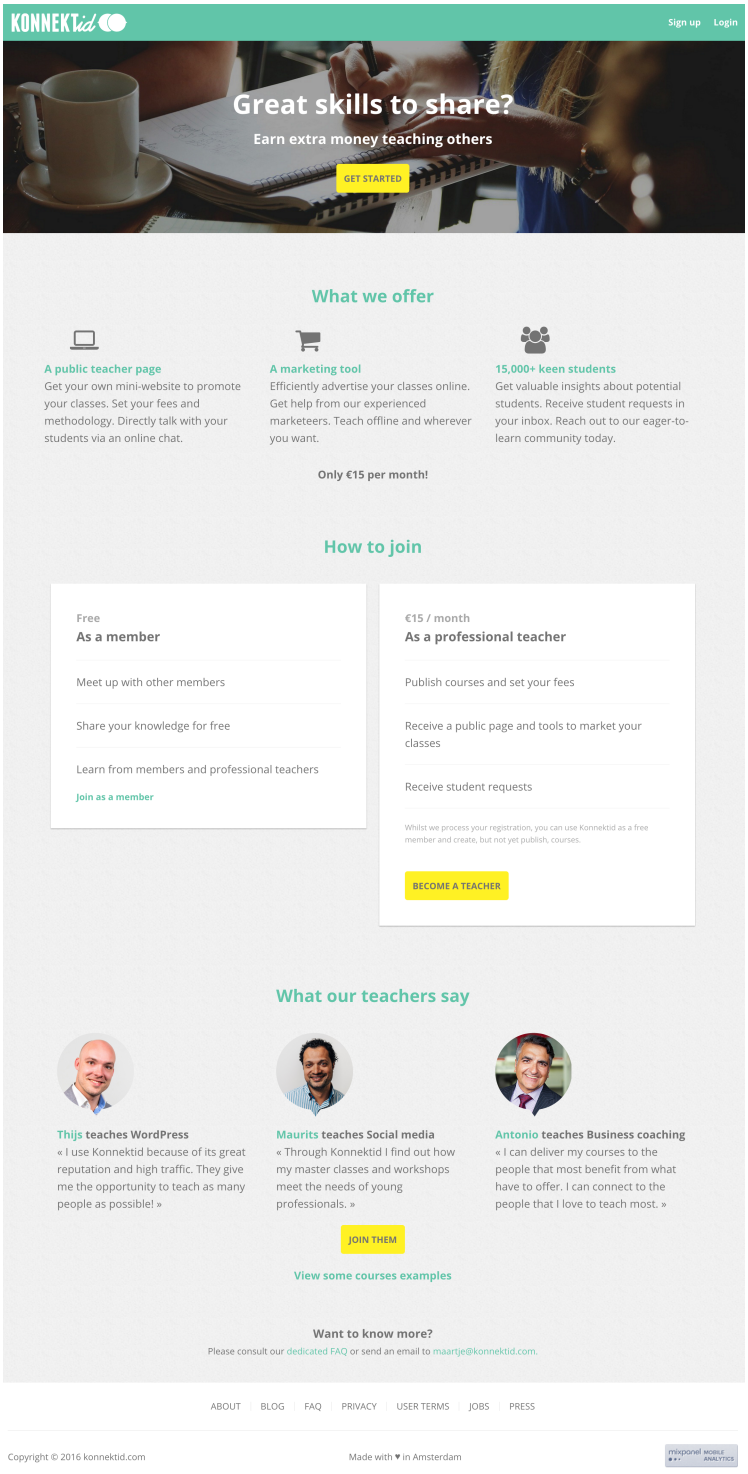
B A course published on Konnektid's website (desktop version)



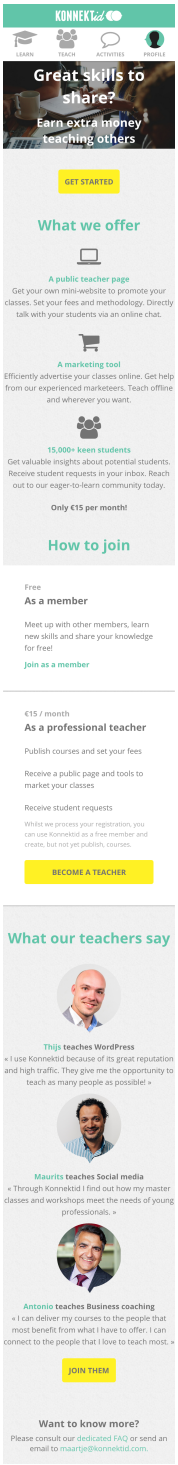
C A course published on Konnektid's website (mobile version)



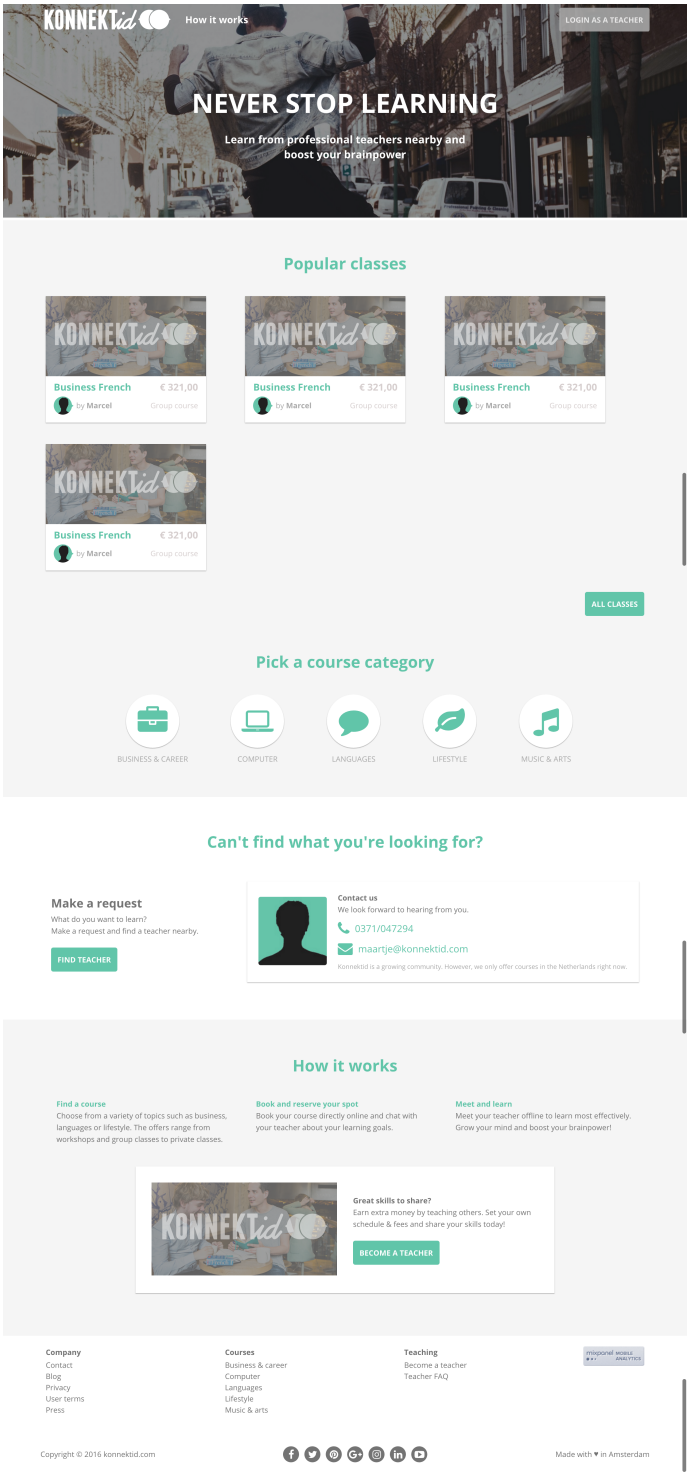
D Refactored teachers landing page (desktop version)



E Refactored teachers landing page (mobile version)



F Homepage for the new market place (desktop version)



References

[1] Asana. Achieve great results. https://asana.com/product. [Online; accessed 11-08-2016].

[2] Numerous contributors. CSS with superpowers. http://sass-lang.com/. [Online; accessed 15-08-2016].

[3] Google Developers. What is V8? https://developers.google.com/v8/. [Online; accessed 10-08-2016].

[4] Node.js Foundation. About Node.js®. https://nodejs.org/en/about/. [Online; accessed 10-08-2016].

[5] Git. About. https://git-scm.com/about. [Online; accessed 15-08-2016].

[6] Google. Why Google Analytics. https://www.google.com/intl/en_ALL/analytics/why/index.html. [Online; accessed 11-08-2016].

[7] Pete Hunt. Why did we build React? https://facebook.github.io/react/blog/2013/06/05/why-react.html, 2013. [Online; accessed 04-08-2016].

[8] Facebook Inc. GraphQL Introduction. https://facebook.github.io/react/blog/2015/05/01/graphql-introduction.html. [Online; accessed 10-08-2016].

[9] Facebook Inc. Interactivity and Dynamic UIs. https://facebook.github.io/react/docs/interactivity-and-dynamic-uis.html. [Online; accessed 04-08-2016].

[10] Facebook Inc. JSX in Depth. https://facebook.github.io/react/docs/jsx-in-depth.html. [Online; accessed 04-08-2016].

[11] Facebook Inc. Redux. http://redux.js.org/index.html. [Online; accessed 04-08-2016].

[12] Jenkins. Jenkins. https://jenkins.io/. [Online; accessed 10-08-2016].

[13] Mixpanel. Funnels let you see where you lose customers. https://mixpanel.com/funnels/. [Online; accessed 11-08-2016].

[14] Inc. npm. What is npm? https://docs.npmjs.com/getting-started/what-is-npm/. [Online; accessed 10-08-2016].

[15] Robin Rendle. What are CSS Modules and why do we need them? https://css-tricks.com/css-modules-part-1-need/. [Online; accessed 22-08-2016].

[16] Slack. Team communication for the 21st century. https://slack.com/is. [Online; accessed 11-08-2016].

[17] Mikito Takada. Single page apps in depth. http://singlepageappbook.com/, 2013. [Online; accessed 04-08-2016].

[18] Anthony Williams. Writing Maintainable Code. https://www.justsoftwaresolutions.co.uk/articles/maintainable_code.html. [Online; accessed 17-08-2016].

Résumé

Ce stage de fin d'études de 6 mois a été réalisé à Amsterdam dans une startup appelée Konnektid. Il s'agit d'une plateforme d'échange de connaissances entre voisins. Le challenge actuel pour l'entreprise est de valider leur modèle financier, basé sur des enseignants professionnels qui payent pour publier leur classes sur le site web et y trouver des étudiants proches de chez eux.

Cela implique de créer de nouvelles fonctionnalités au sein de l'application, et ce fut l'objectif principal du stage.

Les principaux langages utilisés pendant le stage sont JavaScript, HTML et CSS. J'ai aussi découvert un certain nombre de librairies modernes telles que ReactJS ou Redux. Mes tâches étaient très variées, même si la plupart consistaient à créer les interfaces pour de nouveautés du site web, à partir de maquettes. Par exemple, j'ai aussi implémenté un système d'analytiques permettant de relever les actions des utilisateurs sur le site, ce qui aide à détecter les zones ayant besoin d'améliorations.

J'ai aussi appris à utiliser Github de façon efficace malgré une configuration plus complexe, avec plusieurs environnements de développement.

Finalement, faire partie d'une startup permet de réaliser l'importance de la communication et de la collaboration lors de la gestion de projet. L'équipe est petite, mais l'entraide est très présente et l'avis de chacun est apprécié lors des réunions et des brainstorms.

Abstract

This 6-month internship took place in Amsterdam in a startup called Konnektid. It is a skill-sharing platform specifically built for neighbours. The company's objective for the year is to validate their business model, based on professional teachers who pay to publish courses on the website and to find students nearby.

This involves the creation of new features in the application, and this was the most important goal of the traineeship.

The main langages that were used are JavaScript, HTML, and CSS. I also got to use some modern libraries such as ReactJS or Redux. My tasks were very diverse, even if most of them implied building interfaces for the new pages and features of the website, based on mockups. For instance, I also implemented the analytics used to collect actions performed by the users of the application, which helps defining where improvements are needed.

I also learned how to use Github in an efficient way despite a more complex configuration with several development environments.

Finally, being part of a startup made me realize the importance of communication and collaboration in project management. The team is small, but there is a lot of mutual help and everyone's opinion matters during meetings and brainstorms.

INSA

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839 - 35708 Rennes Cedex 7
Tél. +33 (0) 2 23 23 82 00 - Fax +33 (0) 2 23 23 83 96
www.insa-rennes.fr