Hypatia LMS: Open Realtime Education

Joan Siddharta Mira Martos
Multimedia degree
Web engineering

Ignasi Lorente Puchades
Carlos Casado Martínez

17th January 2017

FINAL ASSIGNMENT DETAILS

Work title:	*Hypatia LMS: Open Realtime Education*
Author:	*Joan Siddharta Mira Martos*
Website:	*https://theonapps.github.io/hypatia/*
Assistant teacher:	*Ignasi Lorente Puchades*
Teacher:	*Carlos Casado Martínez*
Due date:	*09/01/2017*
University program:	*Multimedia degree*
Area of work:	*Web engineering*
Language:	*English*
Keywords	*Education, eLearning, management, web, LMS, CMS, React, Firebase, development, JavaScript, learn, teach,*

Project summary

Hypatia is a FREE, open source, realtime Learning Management System based on Facebook's React framework, Google's Firebase platform and Slack.

Hypatia aims at solving the issues that current LMSs have, like bad and slow UX, old and cluttered designs and unresponsive layouts. The way these issues will be solved is by using the most advanced web technologies, like the shadow DOM for fast rendering, web sockets for realtime communication, JSON API for standardised and formatted data, integration with 3rd party APIs and the latest HTML5, CSS3 and JavaScript features to build a functional and beautiful UI following the best practices.

Hypatia is designed with universities and academies in mind. There's no size limit. It can be used for small academies (5 - 100 students) and for big universities (1K - 100K). The system will be scalable, allowing modules to be enabled/disabled based on the requirements of the organisation.

The ultimate target of this project is to offer a solution to the open source community that replicates in the LMS World the pleasant experience built and developed by the Wordpress community. Flexible enough to suit the needs of the majority of educational projects by being scalable, easy to use/learn and opened to be used, modified and improved.

Hopefully, Hypatia will enable people around the world to make education more accessible and enjoyable, especially for those who live in developing countries.

Summary

Figure's list

1. Introduction

1.1 Context and project justification

The reason why I chose to build a Learning Management System is because I keep seeing how things could be done better in many e-learning related websites. The current systems do not use agile communication, which makes student and teacher interaction particularly inefficient. The experience feels slow, with server related errors appearing often, annoying popup windows, cluttered UI with non-relevant data…

This problem is highly relevant, as online education is becoming more and more popular, especially in developed countries, where society is used to interacting with computers in a daily basis. Not having an updated and well taken care of e-learning ecosystem, makes people's lives more difficult, as they have to balance a highly productive job and personal environments with slow and unpleasant online learning experiences.

Having e-learning websites that don't keep up with the rest of the web evolutions, makes the educational experience less enjoyable, which, in some cases, can make students give up their courses or entrepreneurs fail with their ventures.

To fix this situation, we need to improve the offer out there. We need to make the e-learning software community vibrant and competitive. That's the reason I want to build a platform inspired on successful and modern realtime products, like Slack, Trello or Google Docs and integrate them in a rich, full-flavored and exciting e-learning platform.

There are other LMS already in the market [1] with APIs and modern stacks, like Moodle (23% of market share), Canvas LMS, Blackboard (41% of market share), Matrix, D2L, Kaanu, etc. Unfortunately, they don't offer good UX or they don't make it easy for individuals to gain access to their software and/or modify it to offer a more customised experience for their students.

1.2 Objectives

- The most important goal is to offer a full-stack application to the open source community that enables developers and entrepreneurs to create their own e-learning sites

- The back-end will be driven by Google's Firebase platform, which will simplify the infrastructure maintenance and scalability, making it easy for developers to start and deploy their apps

- The front-end will be initially built using Facebook's React framework, although in a future, it can be ported to React Native to take advantage of the native Android and iOS features.

- The second most important goal is to create a very polished UX by creating a well curated information architecture, easy to follow journeys and have as many user testing revisions as possible

- The simplicity of the navigation and the prioritisation of content is key. Only the most relevant content for a given section will be displayed. This will ensure that the UI remains uncluttered

- The application will implement as many real time features as possible, allowing users to be notified and to communicate with other users in an agile manner. No more old style forums, emails or other slow and cluttered systems to talk to each other or get updates from the system

- Include multi-language support, social user authentication and analytics

- Allow users to create themes for the UI. This will help other users with less web development experience to take advantage of the community to improve the look & feel of their e-learning sites

- Implement a basic admin area to allow teachers/admins edit the courses/subjects' contents, calendars, users, payments, etc

- The essential journey should allow a user to signup/login, read about the available courses/subjects, join a course/subject (with or without paying), read/download the resources, interact with the classmates and teacher, read/download the assignments and watch the calendar

- **Phase II**: create a set of templates and guidelines to allow developers create plugins to enhance the functionality of the platform. These plugins could be new features, like the ability to solve quizzes or integrations of 3rd party services like Paypal, Stripe, Google Docs, Flickr, Dropbox, Slack, Trello, etc

1.3 Method and philosophy followed

The only possible strategy to accomplish the objectives, is to start a new project from scratch. It wouldn't be possible to match the UX, design and realtime expectations by using an existing product.

Nonetheless, we don't need to create every single unit of information or configure and maintain an infrastructure module by module. One of the most important mantras in computer science is to "not reinvent the wheel" and by that, we mean that it's a good practice to take advantage of platforms, libraries, frameworks, boilerplates and other systems that have a high level of consistency and/or support to speed up our development time and increase the quality and potential success of our project.

In any case, we have to make sure that the 3rd party tools or systems that we use, are all very well supported and the people behind their maintenance have a high level and long term commitment with them.

Some years ago, I learned a lesson from my mentor that it is still very relevant in the always-evolving world wide web: "**we have to use the Internet as a CMS**". This line resonated with me for a long time, as it is a statement that takes some courage to follow. It enforces you to trust people and organisations with critical features of your application.

The initial thoughts when choosing the technology stack for a project are always about being cautious and how those decisions could impact the life of the product. Sometimes it's a gamble, we never know if a company is going to close or be bought by another one. The recent history has shown us that things like this happen, like what Steve Jobs did to Flash…

The lesson therefore is to choose wisely and try to follow the standards, like the ones from the W3C, as they tend to be the most reliable and stable reference. In the second level of this imaginary scale of trust would be big companies like Google, IBM, Microsoft, Apple, Adobe, etc. Even if they are still at risk of terminating products and services, they tend to give a lot of support when that happens.

What we should try to avoid (when building a professional product) is to rely on libraries/tools/frameworks with very few people behind them or not committed enough. This is a risky situation, as they might leave the project anytime without notice, leaving us stuck and having to invest precious development time refactoring the app.

After this quick introduction about tech stack decision making, I would like to explain my approach for Hypatia.

In the production of this project, there are four main different areas that require attention: design, UX, front-end and back-end. There are also

other aspects to take care of, like planning, documentation, QA, user testing, deployment and infrastructure.

The approach for the first two main areas, design and UX, consists on researching about several e-learning platforms already in production, like the UOC website, the Moodle LMS, Canvas and Blackboard.

I also want to analyse some realtime tools like Slack or Trello and websites like, RentalCars.com or Booking.com, to find out more about how they implemented this realtime feel to their UI.

Once I have enough information about the basic content needed to create a full learning journey, I will create a basic information architecture, which will consist on a sitemap and a set of wireframes for the different view templates.

Finally, in terms of UX and design, there's an important decision I will have to make at some point regarding editing content. One option is to allow users to edit the content inline and the other option is to create a basic admin area where they can edit the content.

At the same time, when the initial research is done, I will also create some UML diagrams to map the architecture and the database structure. This will help to clarify and confirm the tech stack initially selected.

If none of the above changes the initial plan, the front-end will be architected using Facebook's React framework and the Redux pattern. Of course, the stack consists of other libraries, but these two are the most representative.

Regarding the backend, initially I thought about using a Node.js, Express and MongoDB stack, but after learning about Google's Firebase [3][4], I feel that this could help to simplify a project that is already quite ambitious in terms of complexity and size. At the same time, I think Firebase would also help other developers to manage a stack that otherwise might be too big for the kind of small teams this project is aimed to.

In the last weeks, I have been researching about react boilerplates [5], especially the ones targeting start-ups. This is a very common resource to speed-up the setup of an MVP, which it's precisely what we need to do with Hypatia.

In the following website [5], there's a list of the most popular React boilerplates around the world at this moment. Once I analysed the ones in the top of the list, I came to the conclusion that 'Este' [6] was the most appropriate one, as it supports most of the objectives, including Firebase.

Nonetheless, after evaluating the high complexity of Este, I have realised that it would take me a long time to learn the author's approach. For that

reason, I decided to build my own boilerplate based on my previous experience with other React projects.

There are also other boilerplates with very interesting stacks, including GraphQL and a deeper isomorphic/universal integration, but these might increase the complexity of the stack and therefore of the project.

Regarding the BaaS offered by Firebase, there's not much research required, as they have a fantastic documentation with many code examples and a really easy to use API. One of the aspects that needs a bit of research is regarding the integration of Firebase with Redux and React, as we need a seamless approach to load and save data.

After some time researching, I have found a library that will be a perfect fit. The developers behind it seem to be committed with the project and reply quickly on any inquire.

1.4 Planning

1.4.1 Resources list

- Laptop with Internet connection and Unix terminal (Apple MacBook and Mac OS 10.11)
- Text editor (Brackets) with JavaScript linter
- Node.js, Git, SourceTree and NPM installed
- Google Chrome with React and Redux Dev Tools
- Accounts in the following online websites: Firebase, Github, Facebook, Google
- Adobe Creative Cloud, especially Photoshop and Illustrator
- Domain name (optional)
- A handful of users for user testing

1.4.2 Tasks list

Research

- Analyse the current available LMS products and the features they provide to get ideas and get the concepts clear
- Create a mood wall and mock-ups/sketches to get inspired for the designs
- Build a sitemap and study the effectiveness and consistency of the user journeys, hierarchies and taxonomies
- Think about a logotype and a style guide
- Look for useful resources online that could help speed up the design process, like free icons and fonts

Setup and docs

- Setup the required hardware and software
- Create a GitHub repository for the codebase
- Build my own React boilerplate
- Read the Firebase tutorials and docs
- Review objectives, planning and write docs & diagrams

Prototyping

- Translate the sitemap into a clickable prototype with basic HTML and CSS using just headings and placeholders for the pages
- Build wireframes for the different pages, sections and modules
- Test front-end libraries to find out which one is more suitable. I.e. Bootstrap, Material design, GreenShock, Font Awesome
- Test with different graphic design approaches, type fonts, colours, blocks, shapes, animations in order to polish the style guide
- Check the progress with the teacher and apply suggestions/changes to the prototype

Initial design and development stage

- Translate the wireframes into fully integrated designs using Photoshop or Illustrator
- Import the required icons from the selected resource
- Start building the React components represented in the wireframes. Focus first in the common modules: navigation, panels, flyouts and overlays
- Add 3rd party dependencies as required
- Maintain an ordered repository with correct branching
- Setup Firebase hosting and create a staging environment
- Push the prototype to check that the deployment process works well and that the app works fine in the staging environment
- Check the progress with the teacher and apply suggestions/changes to the designs or app architecture

Intermediate design and development stage

- Continue translating wireframes into design decks
- Continue building React components
- Setup Firebase database structure and perform CRUD operations
- Setup Firebase storage and upload assets
- Initial integration of Redux to connect database with front-end components
- Check the progress with the teacher and apply suggestions/changes to the designs or app architecture

Final design and development stage

- Finish translating wireframes into design decks
- Finish building React components
- Complete the Redux integration
- Build an admin area to manage the data in Firebase
- Check the progress with the teacher and apply suggestions/changes

QA and user testing

- Enter demo content and test the journeys
- Conduct a user testing session followed by a revision of the results. Repeat until the level of quality is acceptable
- Perform a security and performance audit across the application
- Fix bugs
- Deploy to staging the alpha versions
- Check the progress with the teacher and decide which changes are worth implementing, put in the backlog or discard

From alpha to beta version

- Review sitemap, wireframes, designs and components
- Apply new changes into the codebase
- Go through another QA and user testing session
- Fix bugs
- Deploy to staging the beta version
- Check the progress with the teacher and decide which changes are worth implementing, put in the backlog or discard

Go into production mode

- Prepare the production environment and link a domain name
- Deploy to production the release candidate version
- Update the documentation
- Test user journeys
- Apply hot fixes
- Check the results with the teacher and decide the roadmap for the future

1.4.3 Gantt diagram

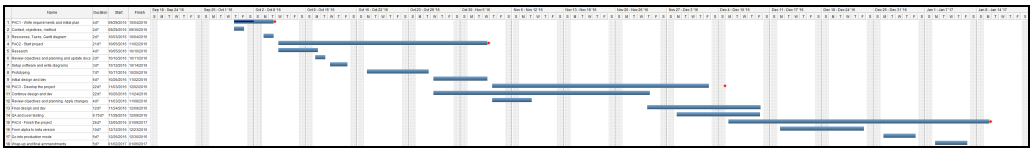

Figure 1. Project Gantt diagram (see annex for original file)

1.5 Short summary of the products developed

- NPM Package configuration with 60 plugins
- Automatic NPM version push support
- Firebase deployment tools configuration
- Firebase database design and rules configuration
- Firebase authentication configuration with email support
- Firebase hosting support for asset management (images, files, etc)
- React scaffolding with Webpack, Babel and other tools
- SASS compilation support
- SVG sprite compilation support
- Development and production environment settings
- React router integration
- Redux integration
- Redux, Firebase and React integration
- Logotype design and different implementations
- Sign up and sign in forms with overlays
- Responsive top navigation
- Dynamic breadcrumbs
- Dynamic dropdown panel for calendar, chat, hub and help
- Chat integration with Slack using bots
- Responsive page footer
- Responsive search panel with expanded view
- Responsive side navigation menu with submenus and animations
- Responsive home page hero animation
  Responsive home page courses and posts loading
- Responsive dashboard page layout and CRUD operations
- Responsive user account forms with Firebase integration
- Responsive listing page for all content types
- Responsive detail page with 1, 2 or 3 columns layouts for all content types
- Responsive admin panel page with dynamic URLs and browser history
- Admin panel integration with Firebase hosting assets (file uploading)
- Admin panel CRUD operations with Firebase database
- Date picker integration
- UI notification module for error and success messages
- UI Modal box component to confirm actions
- Responsive full page loader spinner and small loader
- Error 404 not found page
- Markdown WYSIWYG editor integration with autosave and spell checking
- Demo content for courses, subjects, modules, etc (images and copy)
- Domain registration (theon.io)
- Domain configuration with email forwarding
- Github organisation page (theonapps)
- Github repository configuration
- Prezi alike presentation with impressJS

2. Application design

Hypatia is in itself, a classic modern web application with an agile continuous refactoring process. Even if its technology stack is modern, the development methodology and design process is quite traditional.

In the preproduction stage, we are going to divide the work in two different areas: **design/UX** and **architecture**.

The design and UX area, is going to study and figure out what are the most important features for the target audience. Then it will organise them in a sitemap and create wireframes for each section / feature. This approach will allow to see the big picture of the project and have a visual reference of how big it is, which will help to redefine the timings and resources required to accomplish it.

The architecture area will analyse and propose a technical approach for the infrastructure required, the database structure to hold the business data and other mechanisms to access and update the data via APIs.

2.1 Design / UX

After an initial phase of analysing the features provided by the current LMSs in the market and other websites in the education sector, I created a basic information architecture to support the creation of a MVP.

Below there is a list of wireframes and a sitemap that should provide enough detail about the main features, different page layouts, UI elements and the basic interactions, like toggle, open/close, CTA, etc.

The colours used in the wireframes don't represent any system or code. It's just for presentation purposes and to divide the contexts.

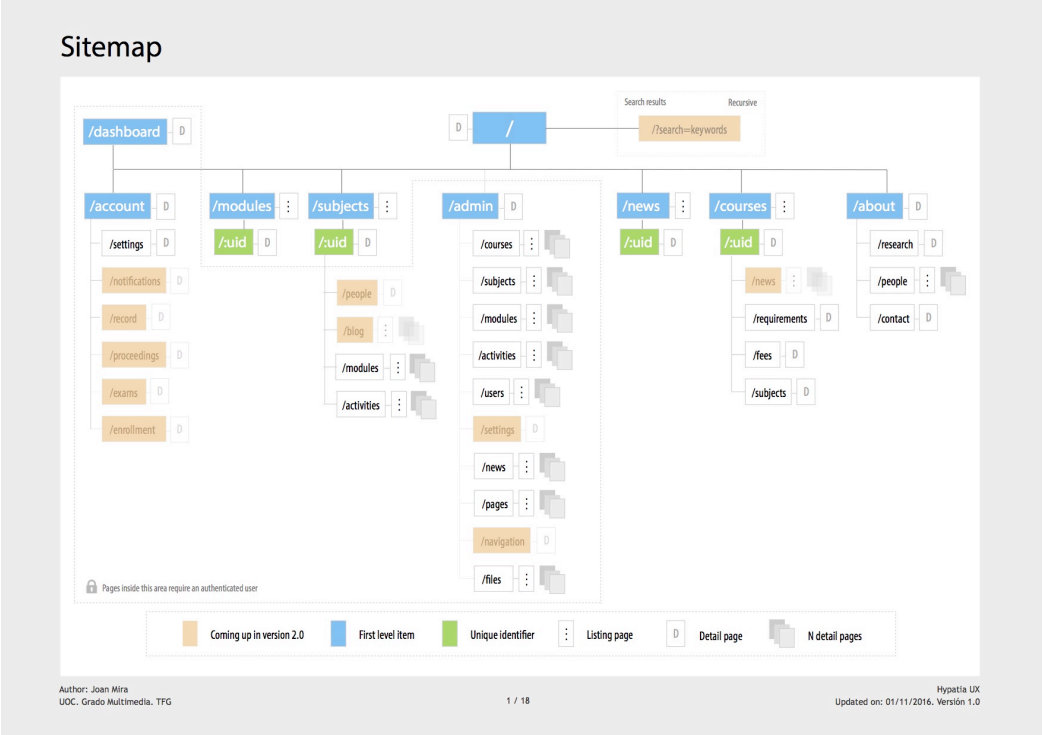

Figure 2. Sitemap with MVP and future features

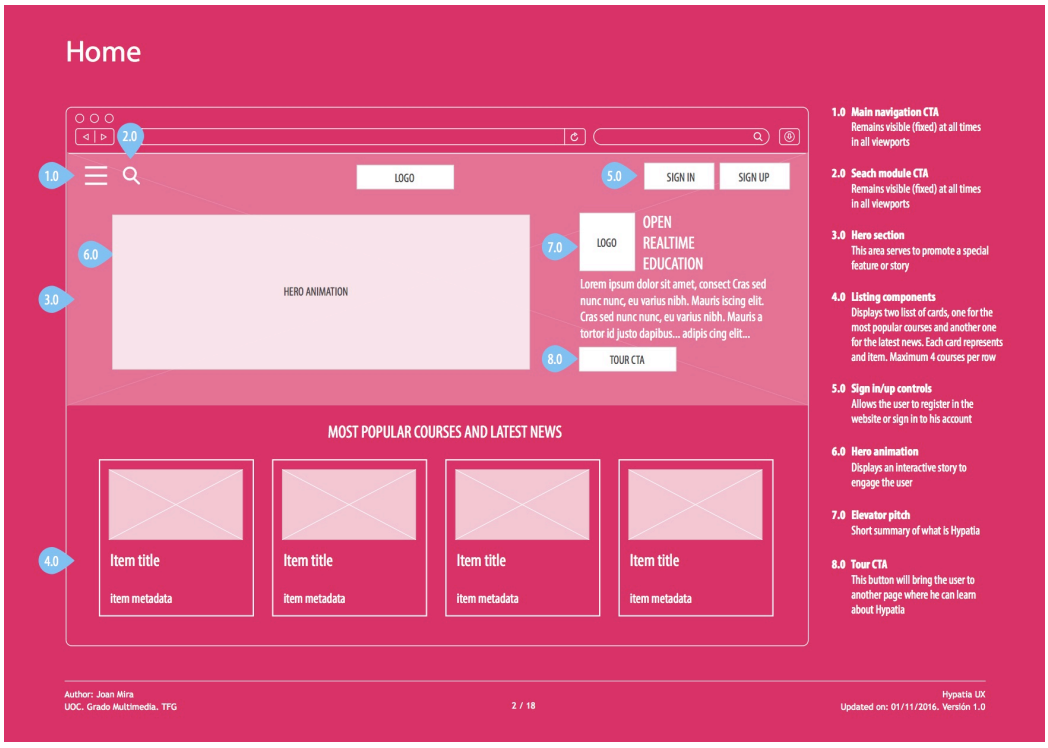

Figure 3. Home page. To display the most popular courses and latest news from the organisation

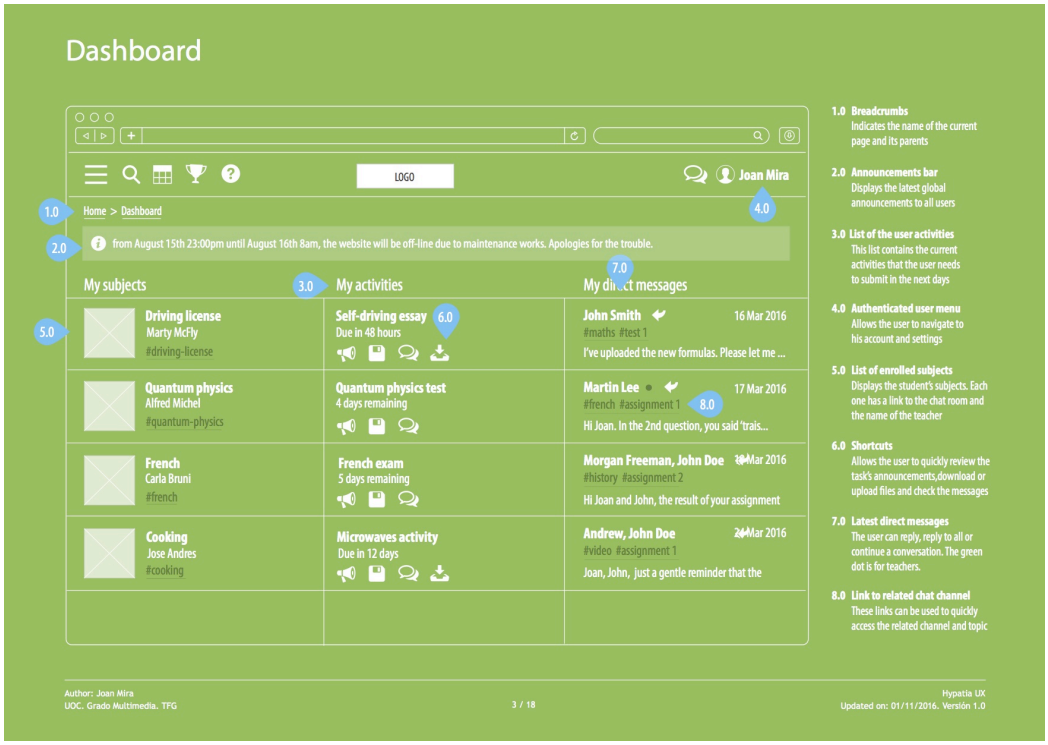

Figure 4. A dashboard page, to allow authenticated users review the most significant day-to-day tasks, like latest announcements, pending submissions, close deadlines and latest messages

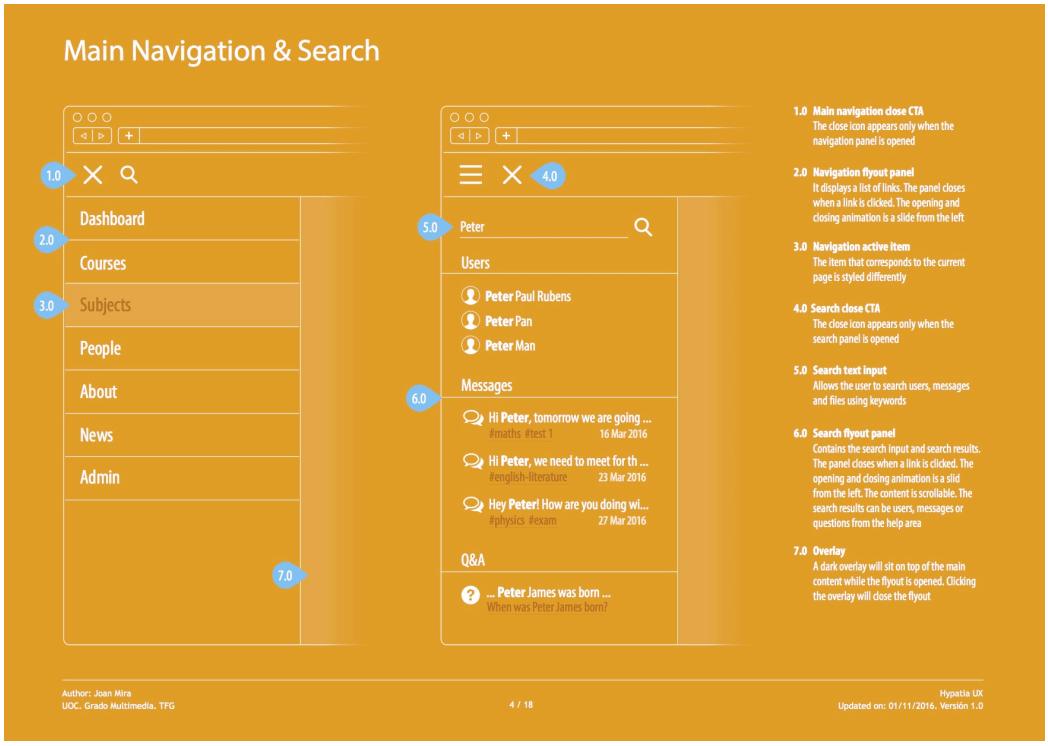

Figure 5. A main navigation, to allow the user the discovery of relevant pages with information about the organisation. A search module, to allow the user finding other users, messages and questions with answers

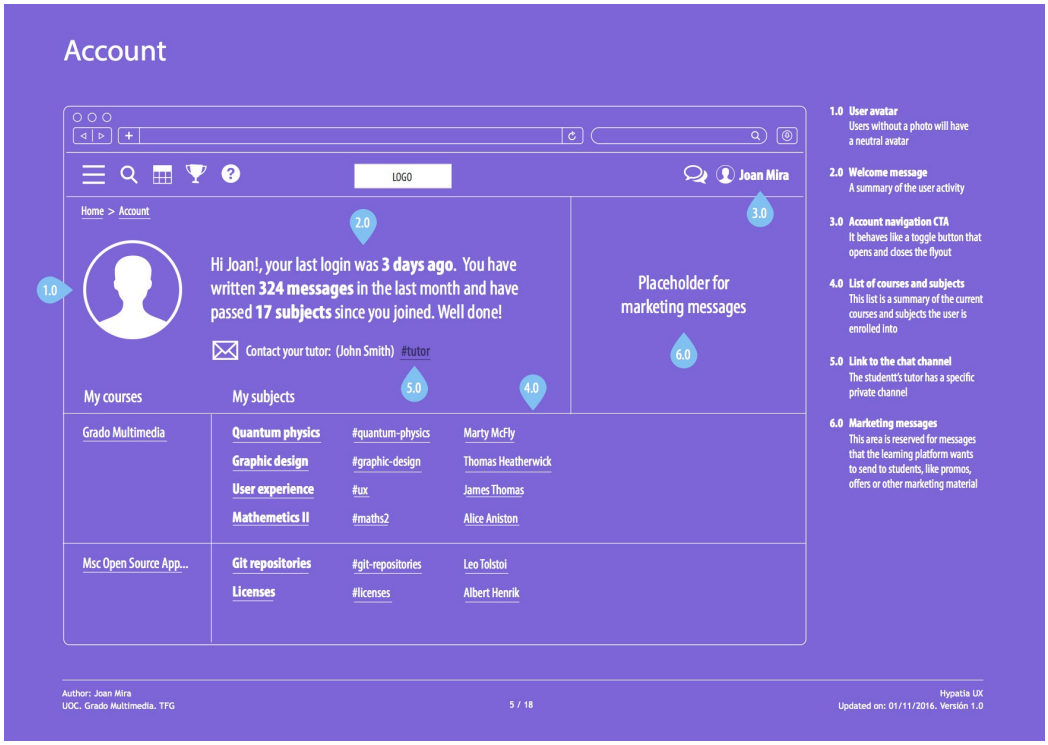

Figure 6. Account page, to allow the user to customise his details, settings, preferences and access the organisation administration

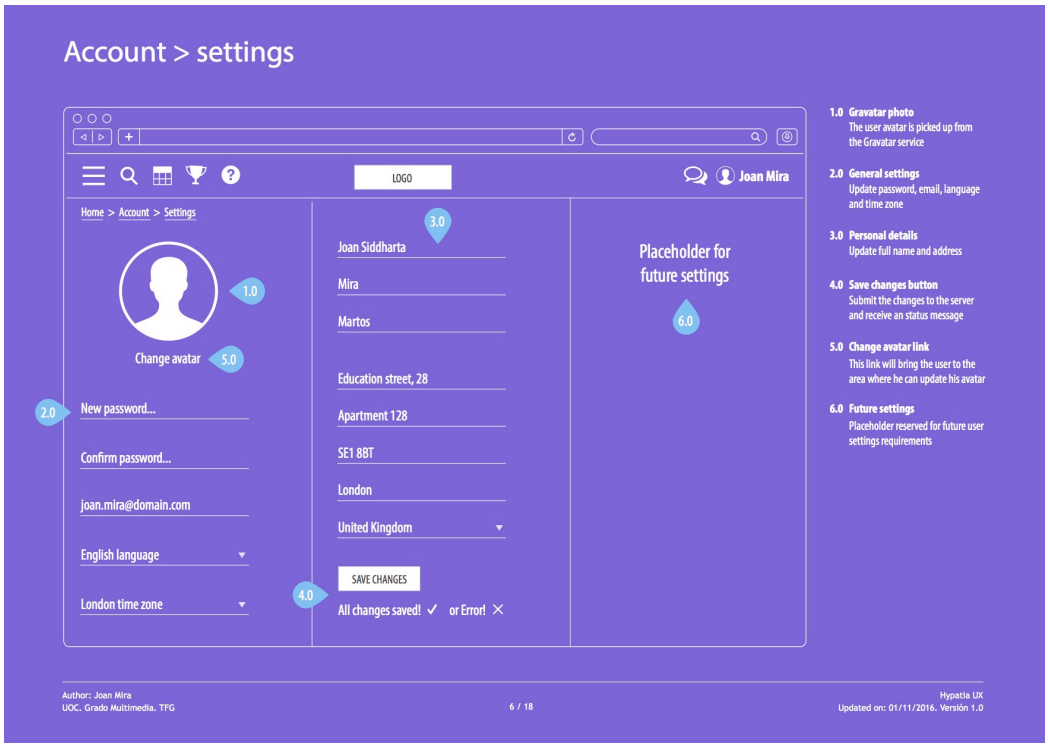

Figure 7. Account settings page

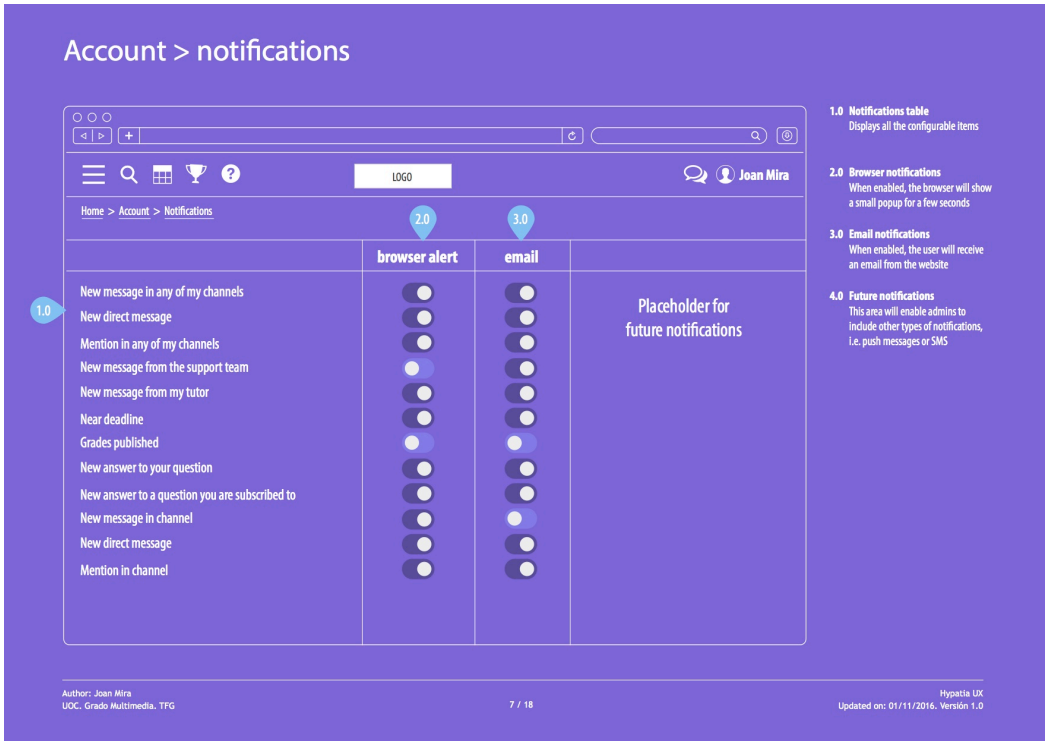

Figure 8. Account notifications page

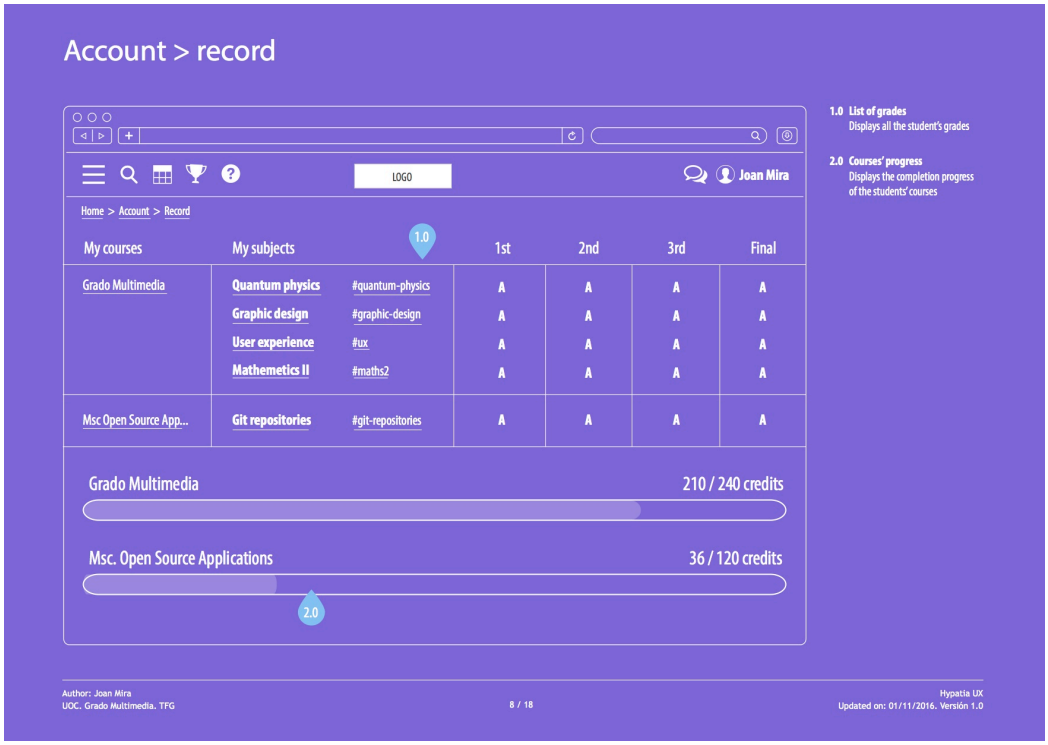

Figure 9. Account record page

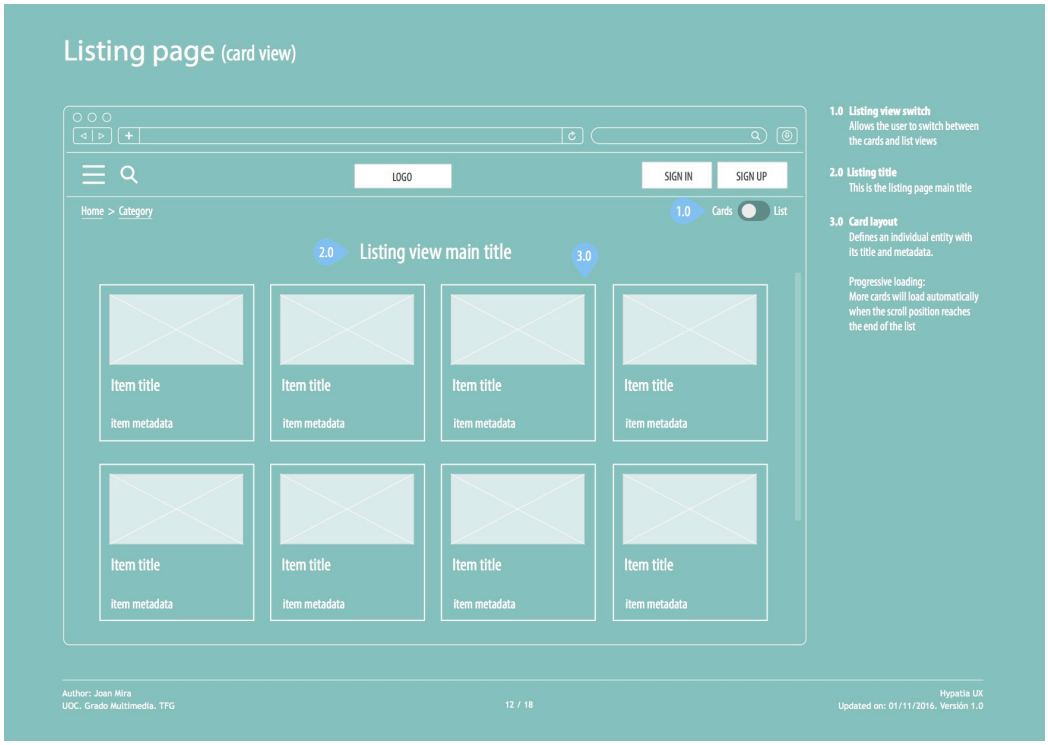

Figure 10. Listing page in card view (reusable template). It can be used by several pages to display a collection of items with pagination

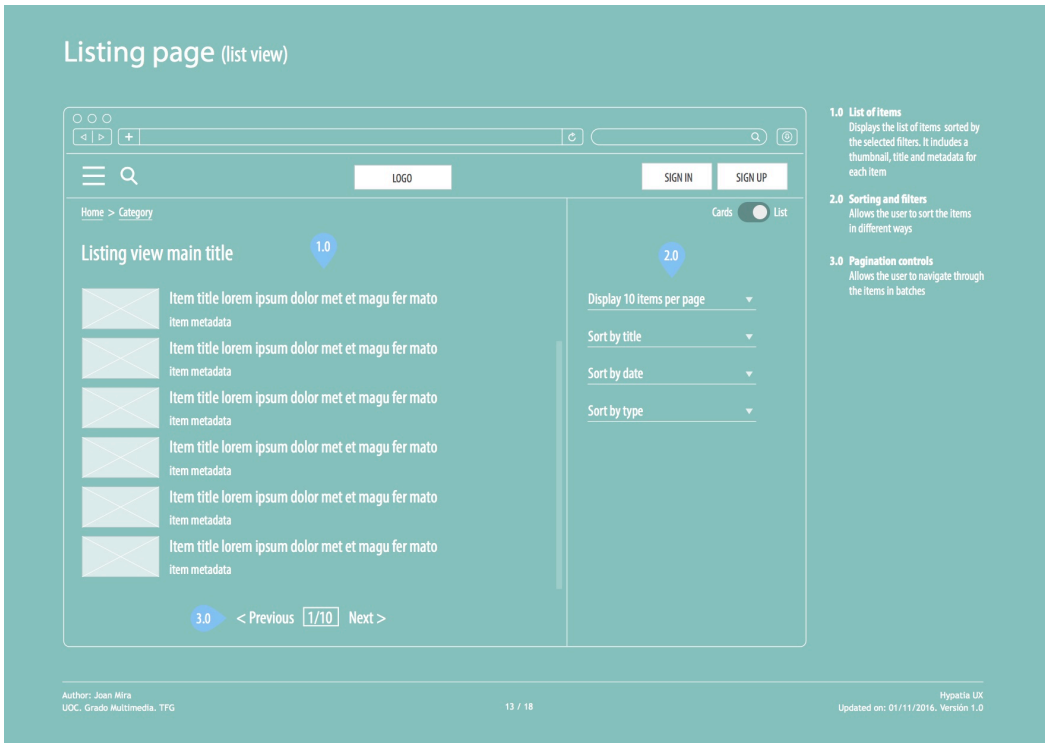

Figure 11. Listing page list view (reusable template)

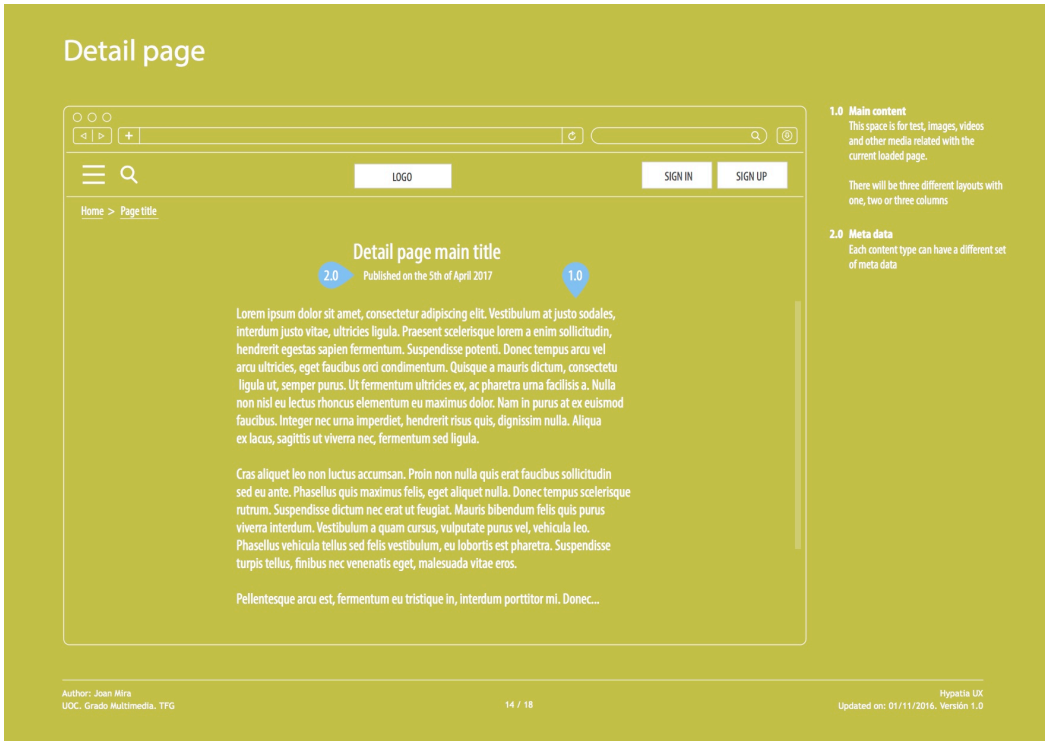

Figure 12. Detail page (reusable template). It can be used to display content entered previously with a markdown editor in the admin area

Figure 13. Chat and messaging module (accessible from any page). Module to allow users to communicate with each other using groups and channels associated with subjects for organising the conversations

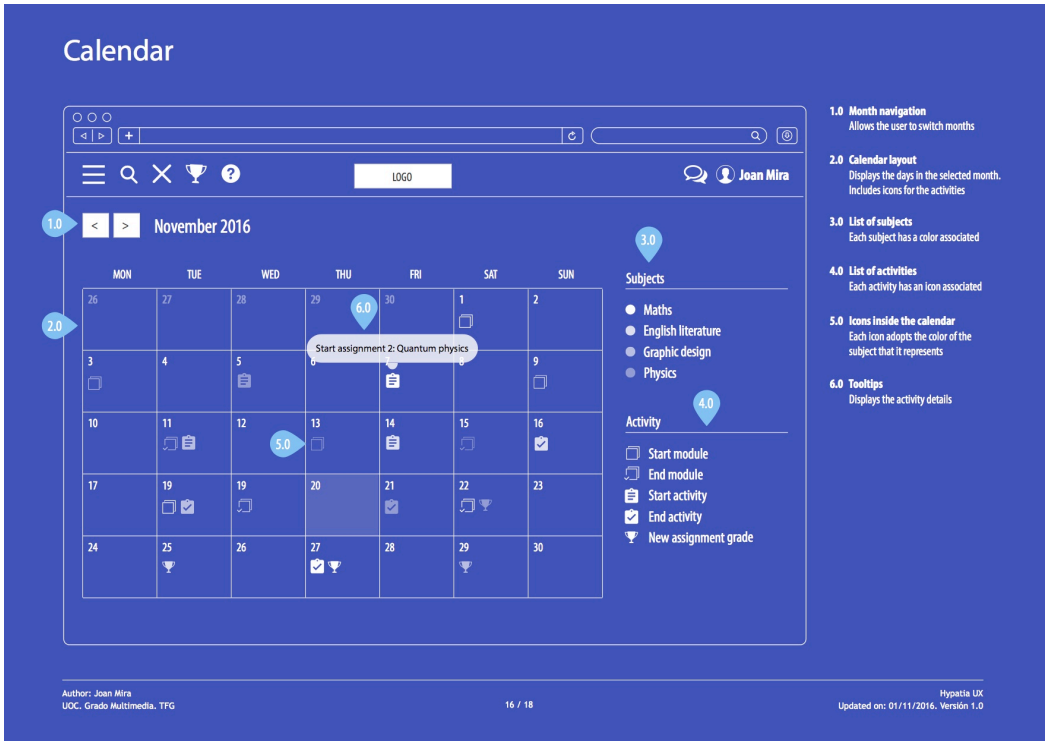

Figure 14. Calendar module (accessible from any page). To allow users to have a wide picture of the important dates to submit activities

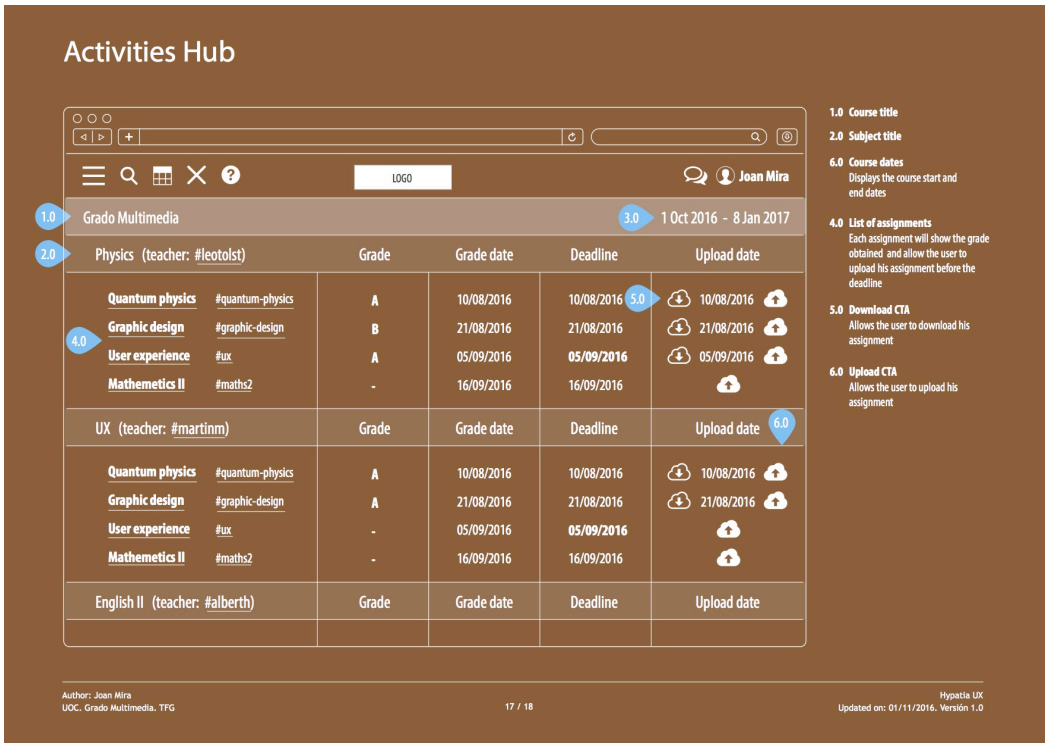

Figure 15. Activities Hub module (accessible from any page). To allow users to review the grades of their current subjects and submit the files for their activities

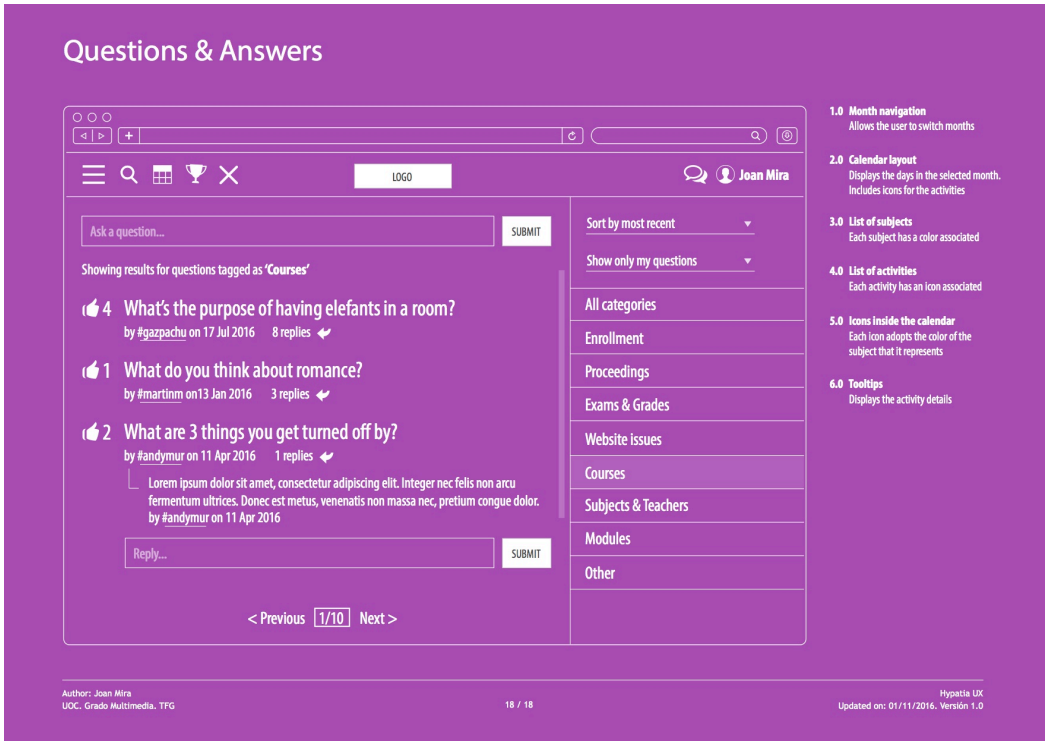

Figure 16. Questions and answers module (accessible from any page)

Each wireframe contains a common module called **top navigation** (topnav). The purpose of this module is to offer a set of CTAs to allow the user to show & hide the most important modules or pages. In this case, the main navigation, search, calendar, activity hub, Q&A, chat and account.

All the wireframes share a neutral plain design. The text fonts, icons, colours, backgrounds, highlights, shapes and spaces have not been chosen with any specific graphic design purpose in mind. They can change in a later phase.

Some of the pages, like the the user account proceedings, exams and enrolment, have been postponed for a future version, as they require insights from users interacting in a production level experience to find out what are the most relevant aspects for these sections.

2.2 Architecture

One of the most significant characteristics of Hypatia is its realtime feel. The website needs to be quick, productive, help users to interact with each other in an easy and intuitive way. To achieve this feeling, the architecture has to reflect this rapid communication system. For this reason, Firebase became the most valuable BaaS partner for this project.

Firebase allows the app to have a clear separation between front-end and back-end by proving an excellent API for CRUD operations, user authentications and notifications. At the same time, Firebase also provides web hosting, asset storage, analytics and crash reporting, which are all great functionalities to help with the maintenance and content management of the app.

Other options would have been to use a custom MEAN stack with a dedicated and bespoke Node.js backend, an express server, a Mongo DB and also a React frontend. This approach might give more flexibility and independence to the project, but at the high cost of having to build an API from scratch and manage a web server by ourselves.

Regarding the API and database, we could have opted as well for more modern and cutting-edge technologies, like Facebook's **GraphQL** or Apollo. They are great platforms, but very new to the community, which makes the offering a bit isolated from other services and platforms. They still need to evolve and merge with the community for one or two more years.

Finally, we could have opted as well for other frameworks like Meteor or Sails, which provide sockets and realtime features. They are both very good options, but I find them too opinionated in terms of front-end and the dependency with the framework seems to be very strong, which would affect future releases of Hypatia.

Regarding the front-end, as I mentioned previously, originally, I selected a React boilerplate called Este [6]. After reviewing the documentation and analysing it in more detail, I decided not to use this boilerplate. My objections were mainly based in two factors:

- The strong support for iOS and Android platforms
- The number of cutting-edge modules used that I had to learn and test

My intention with Hypatia is to build a very responsive web app. Not a hybrid. I believe that using web technologies whenever the nature of the project allows it (i.e. apps without 3D animations and strong CPU requirements) helps a great deal the amount of code that has to be supported and maintained.

Having to support both, native and web code, it's an ordeal, specially for very small teams or solo ventures like mine. For this reason, I will always **vouch for building a strong and well-crafted responsive mobile web app rather than two mediocre apps**, one for the web and one for native platforms. Nevertheless, I would like to explore the iOS and Android native implementations of Hypatia in the future. That's also the reason Firebase could remain a good partner…

On the other hand, the number of modules in the boilerplate were too cutting-edge. It would have taken me a long time to figure out how each one of these modules work before I could start building components properly. This would make the learning curve impossible to bear and risking the project schedule due to **being too ambitious**.

For these reasons, I finally decided to build my own boilerplate. I will start small, just with the basic modules: webpack, react, react router, redux, babel and sass. Then I will add new functionalities one by one. The idea is to build first the scaffolding with all the placeholders for the pages in the sitemap. Then build the database structure in Firebase and finally start connecting both pieces using the API.

Below there is a list of all the architectural diagrams and figures created to support the MVP:

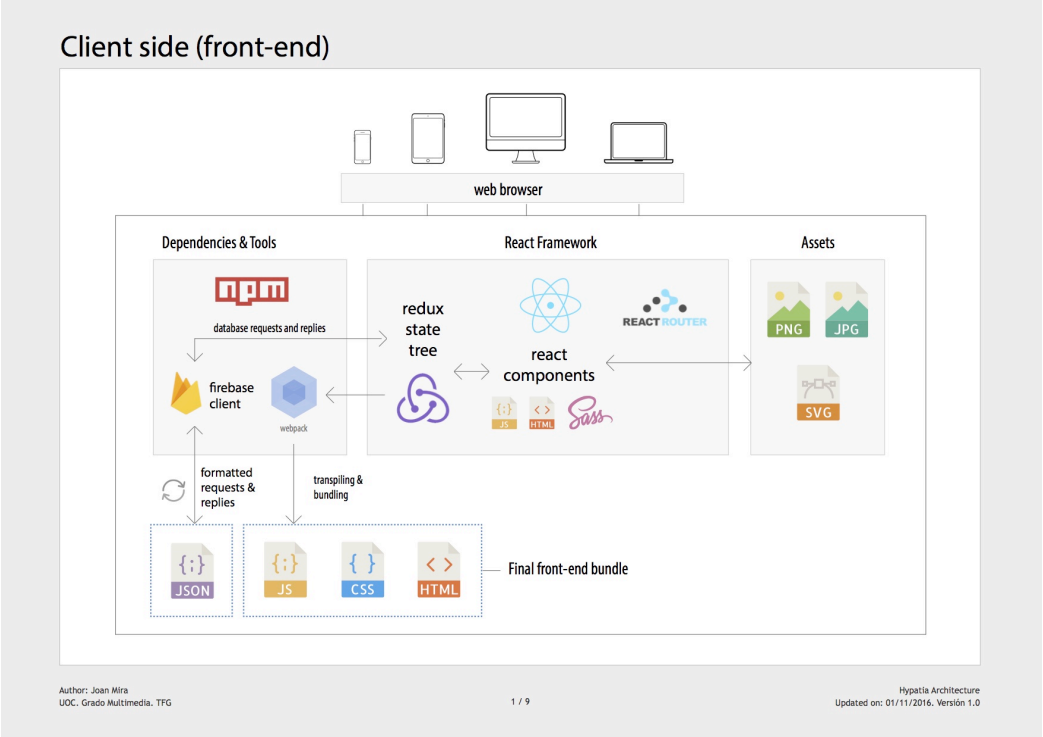

Figure 17. The client side (front-end) diagram specifies how the front-end technologies interact with each other to produce a bundle that can be served in the web server

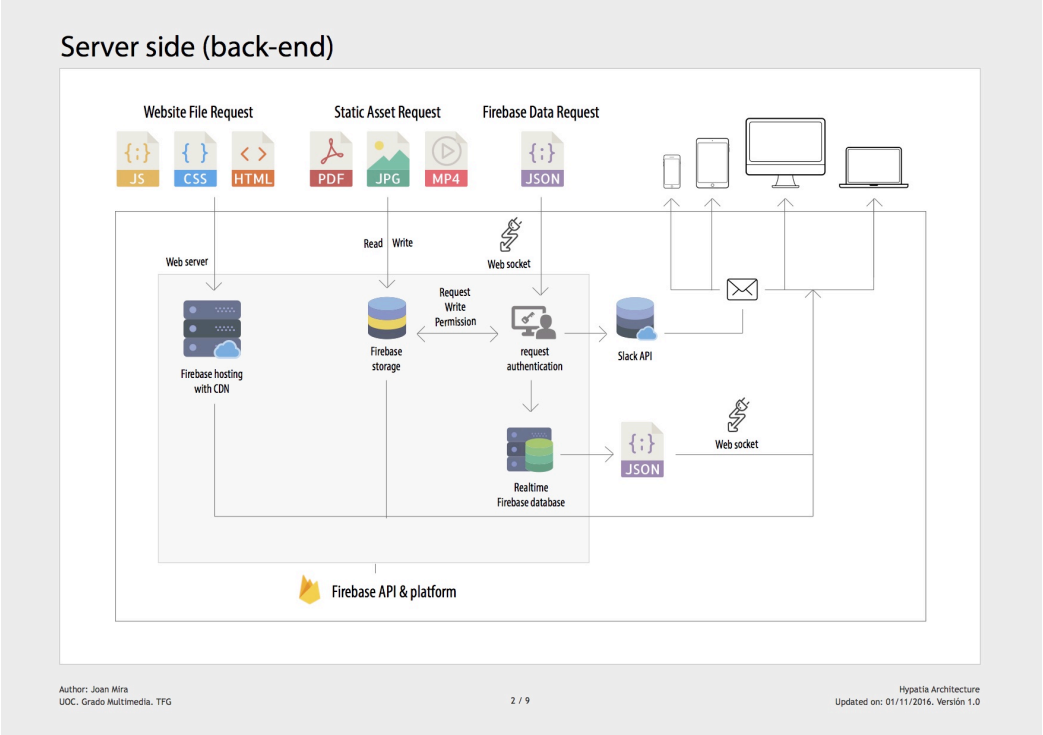

Figure 18. The server side (back-end) diagram indicates how Firebase's services will interact with the front-end by sending JSON data and other assets or by authenticating users

The following Firebase **database examples** are just *snippets* to specify the structure that the NoSQL database nodes should have.

As recommended by Firebase, we have to flatten (denormalize) the data. The target we need to achieve is to access the objects as fast as possible, even if that requires a small level of redundancy.

Although the messaging system will be driven by the Slack integration, I am also adding a proposal of how these messages could be stored in the DB if the users decides not to use Slack.

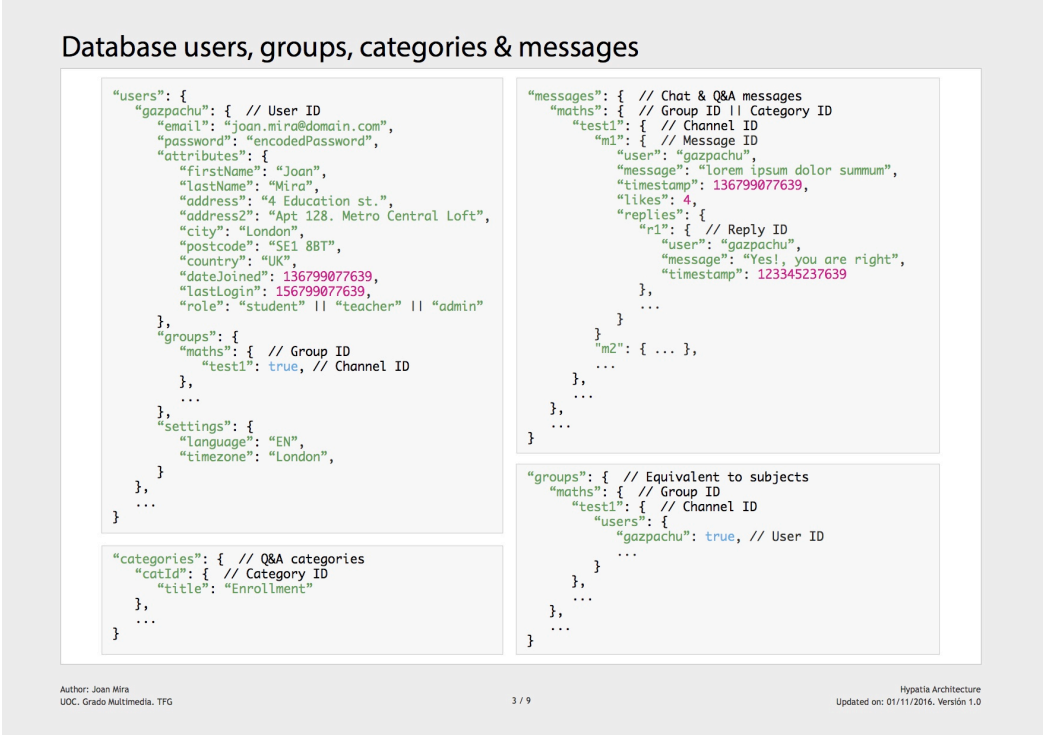

Figure 19. Storing users and an alternative DB structure for messages (if Slack is not used)

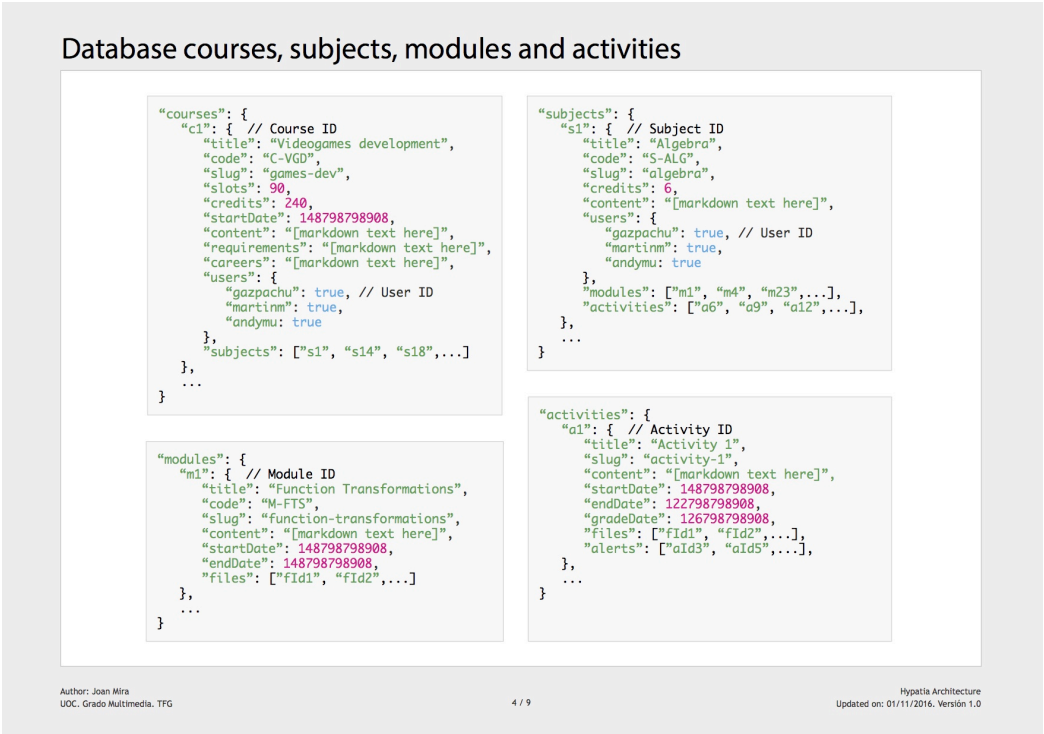

Figure 20. Storing courses, subjects, modules and activities. Courses have subjects and subjects have modules and activities

Figure 21. Storing blog posts (news), static pages and alerts (aka announcements)

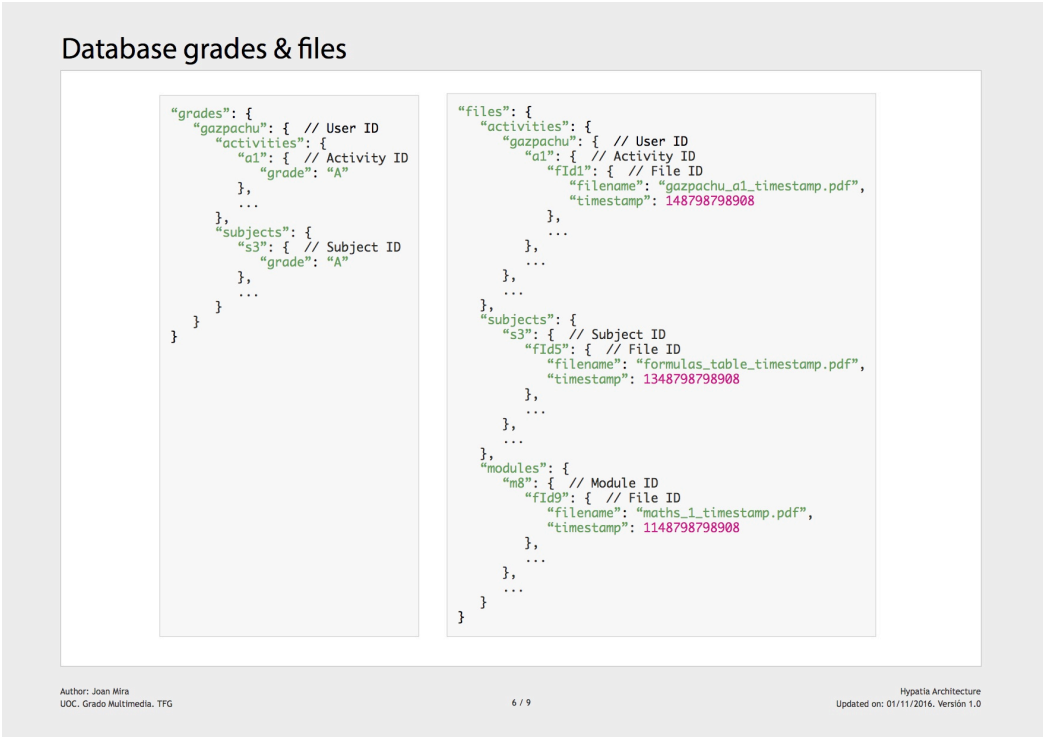

Figure 22. Storing activities and subjects' grades and files

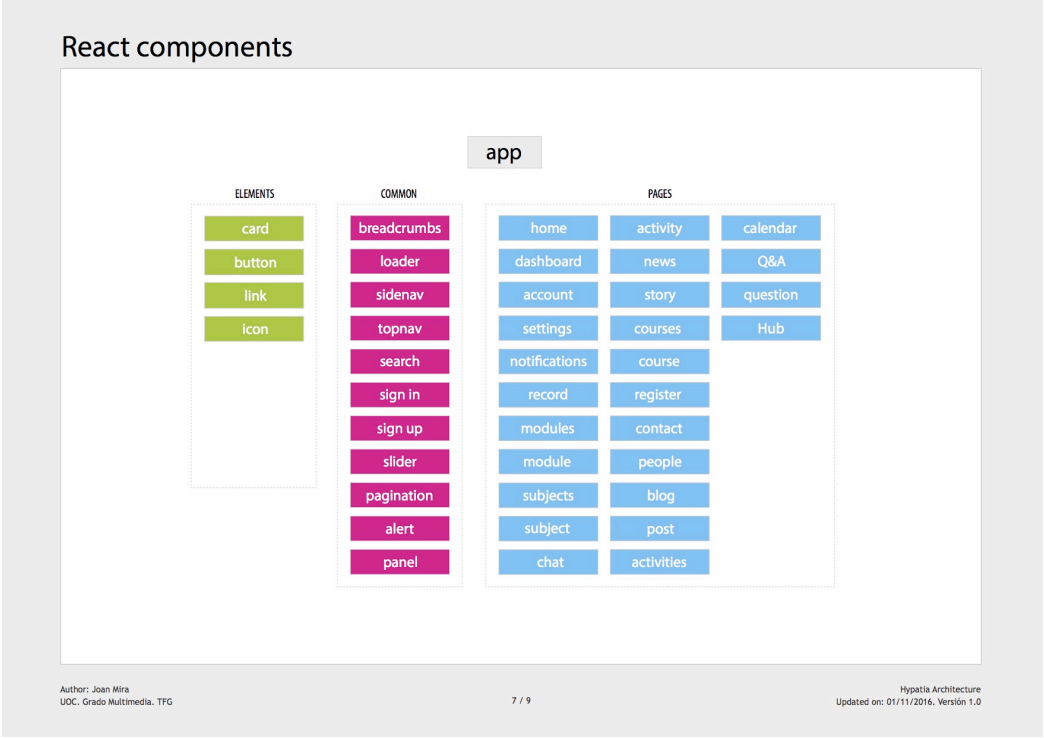

Figure 23. List of classes that the front-end will have to implement to reflect the features specified in the wireframes. Some of these components will have a direct connection with the database, while other will be just small reusable units

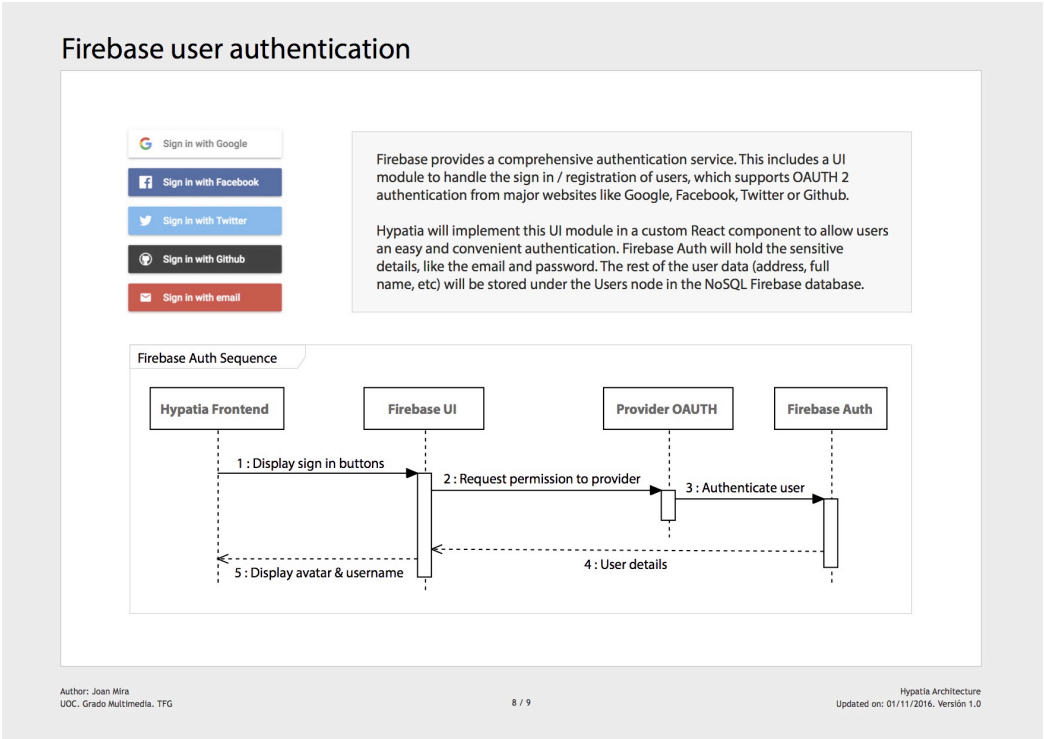

Figure 24. The Firebase authentication sequence diagram specifies the flow of interaction when performing a user authentication operation. For this particular feature, we will use Firebase UI, which follows best practices and will reduce the development time

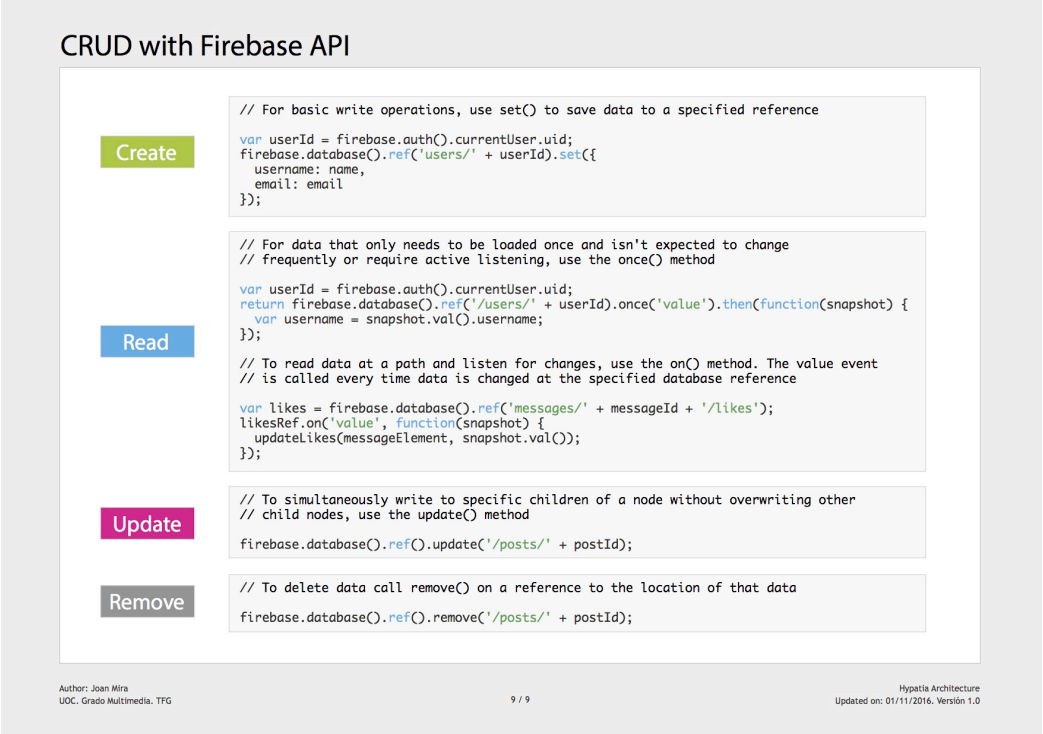

Figure 25. The Firebase API CRUD operations, we can find examples of how to perform the majority of data transactions between the UI and the database

At this point, with the design and architecture plans ready, the next step is to build a small prototype where we can navigate from one page to another and read the headings of each page. Having this scaffolding of placeholders will help us to start mapping the components, create all the necessary files and start testing and integrating other modules one by one.

2.3 Use cases

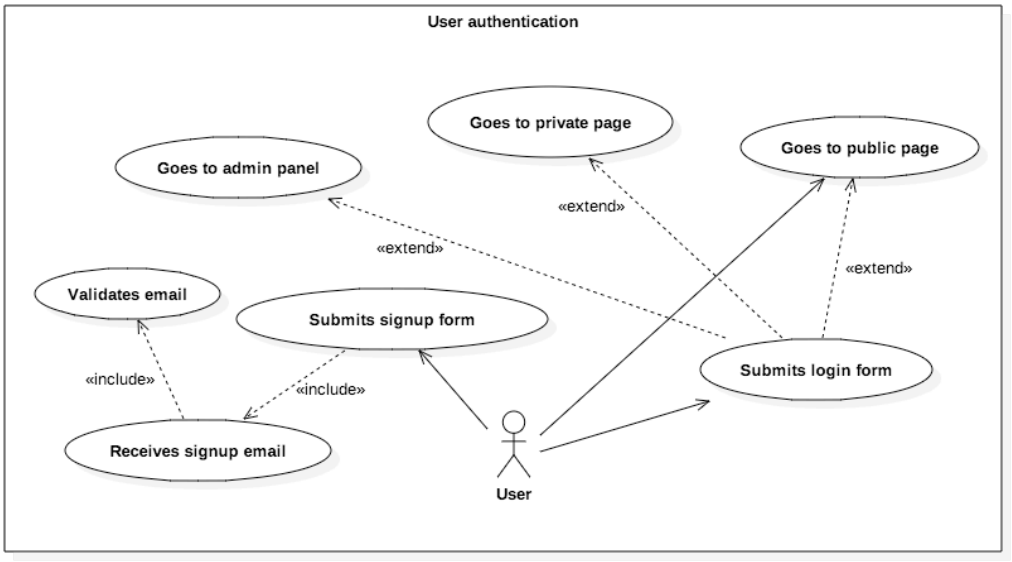

Figure 26. User authentication

Use case: user authentication
Main actor(s): user
Scope: any page
Stakeholders and interests:
User: wants to register an account to access private pages
Preconditions: the user needs to be logged-out
Minimum guarantee: the system will notify the user of any error during the sign-up or sign-in procedures
Success guarantee: the system will start a user session and allow the user to navigate to his account and other private pages
Main success scenario:

1. The user lands in any page and presses the sign-up link
2. The user completes and submits the sign-up form
3. The system displays a notification to inform the user that he needs to validate his email address
4. The user receives an email and presses the link to validate his email
5. The email validation page informs the user that his email is now validated
6. The user completes and submits the sign-in form
7. The user can now navigate to any public or private page

Extensions:

2.a. The data in the form is not correct
1. The system displays a message to indicate the user the data that needs to be modified
2.b. The email address is already in the DB
1. The system indicates the user that the email address is already registered
2. The user can reset the password by pressing the link
4.a. The user doesn't receive the email to validate his email

1. The user tries to sign-up or sign-in again, the system resends a new validation email

5.a. The validation link has expired

1. The user tries to sign-up or sign-in again, the system resends a new validation email

6.a. The email or password doesn't match

1. The system displays an error message and allows the user to enter again the login details
2. The user request to reset his password by pressing a link

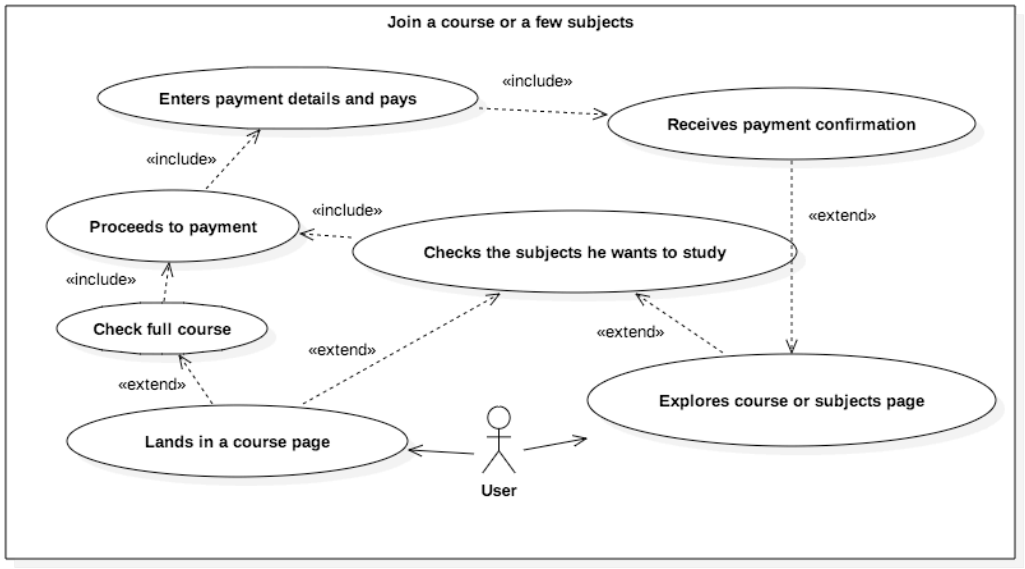

Figure 27. Join a course or a few subjects use case

Use case: join a course or a few subjects
Main actor(s): user
Scope: course page
Stakeholders and interests:

User: wants to enrol in a full course or just in a few subjects

Preconditions: the user needs to sign-up, validate his email and sign-in
Minimum guarantee: the system will record the attempt to join the course by the user
Success guarantee: the system will add the course and the selected subjects (if applicable) to the user account and will show the results of the payment operation (if applicable)
Main success scenario:

8. The user lands in the page of the course he wants to join
9. If the course is not divided in subjects, the user will attempt to enrol the full course by pressing a button to confirm
10. If the course is divided in subjects, the user will check the ones he wants to enrol and press a button to confirm
11. A payment gateway will appear (if applicable) and the user will confirm his payment details

12. The system will process the request and return the results of the operation
13. The system will register the user in the selected course or subject(s) and display a confirmation message

Extensions:

2.a. The enrolment date has already passed
1. The system displays a message to indicate the new date

3.a. Some subjects are not available this semester
1. The system displays a message to indicate it

4.a. The payment gateway doesn't show up
1. The system returns to the course page and displays an error message from the payment provider

5.a. The payment gateway doesn't accept the payment
1. The system displays an error message and allows the user to enter again the payment details

6.a. The connection to the DB is lost
1. The system displays an error message and invites the user to try again the operation later

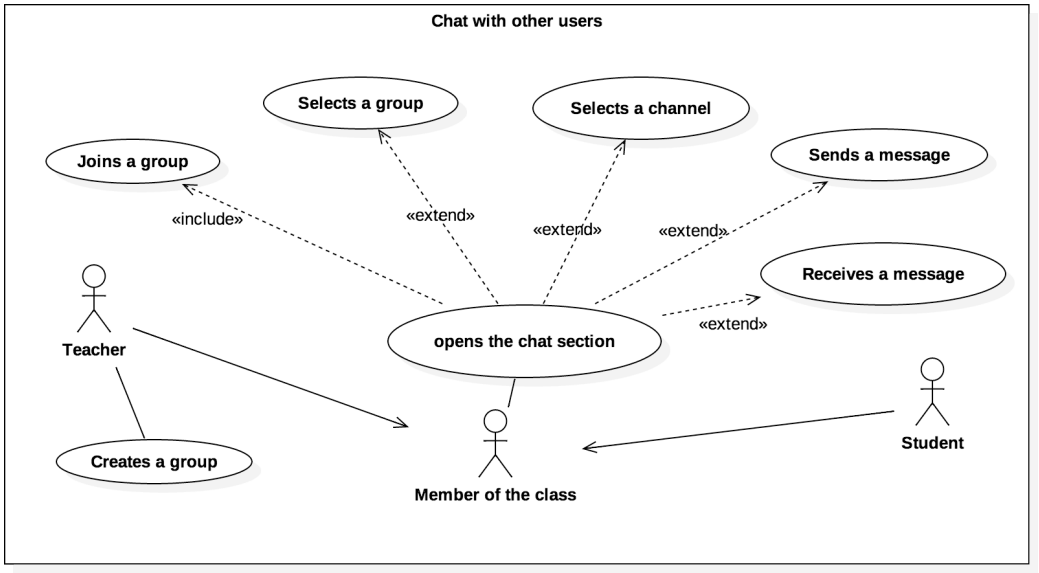

Figure 28. Chat with other users use case

Use case: chat with other users
Main actor(s): user (teacher and student)
Scope: any page
Stakeholders and interests:
Teacher: wants to create groups, channels and add students
Student: wants to chat with the teacher and other students
Preconditions: all users need to be authenticated
Minimum guarantee: the system will display an error message when the messages cannot be sent and/or received

Success guarantee: the system will allow all users to chat in different channels and groups, using the Slack client or the web client

Main success scenario:

1. The teacher creates a new group for his subject
2. The teacher creates a new channel in the group
3. The teacher invites to the group to all the students enrolled in his subject
4. The student opens the chat module from the top navigation
5. The student can see the groups associated to the subjects he is currently enrolled and their channels
6. The student selects a group and a channel where he wants to chat
7. The student sends a message
8. The student receives a message

Extensions:

1.a. The Slack service is down

1. The teacher receives a message alerting him that the service is currently down and to try again later

2.a. The channel is already created

1. The system displays a message to indicate that the channel is already created

3.a. The connection to the DB is not working

1. The system alerts the teacher that he cannot invite other users at this moment and to try again later

5.a. The user cannot see the groups corresponding to the subjects he is enrolled to

1. The system alerts the user that needs an account in Slack to use this service

7.a. The message is not delivered

1. The system displays an error message and invites the user to try again the operation later

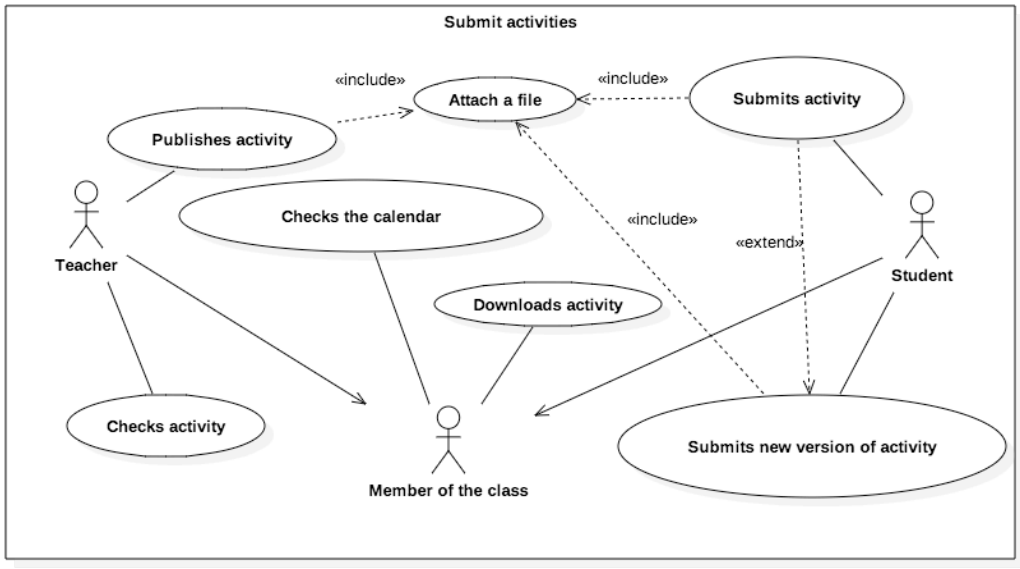

Figure 29. Submit activities use case

Use case: submit activities
Main actor(s): user (student and teacher)
Scope: calendar and activities hub modules
Stakeholders and interests:

Teacher: wants to publish activities and review them once the students complete them and upload them
Student: wants to download and upload activities

Preconditions: all users need to be authenticated. Students need to be enrolled to the subject which the activity belongs to.
Minimum guarantee: the system will inform the user if the file upload was not successful
Success guarantee: the system will register the file upload by the student or teacher with a timestamp
Main success scenario:

1. The teacher uploads a file to the activities hub
2. The system informs the teacher that the file is correctly uploaded
3. The student notices a new notification in the calendar
4. The student checks the notification in the calendar and opens the activities hub
5. The student downloads the file
6. The student uploads a new file

Extensions:

1.a. The file upload is not successful
    1. The system displays a message to alert the user
5.a. The file download doesn't start
    1. The system displays an error message after a few seconds
6.a. The file upload is not successful
    1. The system displays a message to alert the user

3. Application development

All the source code and installation instructions can be found in the Github repository of the project: https://github.com/theonapps/hypatia. In the following paragraphs, I will be using links to files in this repository to explain significant parts of the codebase.

Once the application design releases the first version of the information architecture and wireframes, the development process can start. The idea is to begin by selecting a boilerplate that will help us speed-up the scaffolding of the application. In this case and after a few considerations previously mentioned, it was decided to use a simple react and redux boilerplate [7], which will give us more freedom to scale.

This boilerplate is not perfect by any means for our project, which means that some modifications and additions will have to be done in order to setup the build process in a way that will match our needs.

3.1 The boilerplate and the extra modules

The initial setup of the boilerplate consists of the following modules:

- **Node.js**: this a JavaScript runtime environment that we need in order to work with NPM
- **NPM**: this is a node.js module to help us import libraries into our project. More info here
- **Webpack**: this is a bundling tool. It allows us to include some automatic features during our development process, like generating CSS from our SASS files, transpiling the ES6 code into ES5, building the source code, autoprefixing our CSS code, minifying our code, etc
- **Webpack-dev-server**: to have a local web server
- **React router**: this module is very important in our application. It allows us to create routes for our pages and synchronise the router with other modules like redux or the browserHistory
- **BrowserHistory**: this module is required to have friendly urls in the browser and make them work with the back and forward buttons of the browser
- **Redux**: this is also a very important module that handles the global state of our data, so we can share it across all the components
- **Babel**: this module transpiles the JavaScript ES6 into ES5. There are still some browsers that cannot work with ES6
- **React Hot Loader**: this module auto reloads the browser whenever a source file changes, which makes the coding experience more enjoyable
- **Testing with Mocha and Chai**: these libraries will help us to test the source code and cover as much possible error scenarios as possible

Aside from that, we are going to add the following external dependencies:

- **Firebase**: the official package to interact with Google's Firebase API
- **Redux-react-firebase**: a 3rd party library to integrate Firebase and Redux
- **Axios**: a library to perform AJAX requests
- **jQuery**: a library to manipulate the DOM
- **Classnames**: a module to perform complex class modifications in React
- **Copy-webpack-plugin**: this will help us copy assets to the production folder
- **Webpack-version-file-plugin**: to export the NPM version and be able to read it in the front-end
- **Slack**: a module to connect with the Slack API
- **GH-Emoji**: a module to handle emoticons
- **Momentjs**: a library to work with dates
- **Lodash**: a library of utilities for JavaScript common operations
- **React-ga**: google analytics support
- **React-helmet**: for page titles manipulation and SEO
- **React-router-redux**: to integrate react-router with redux
- **Showdown**: a markdown parser
- **SimpleMDE**: asimple WYSIWYG markdown editor
- **React-date-picker**: a simple date picker for React
- **React-select2-wrapper**: a dropdown component with search
- **SVG-sprite-loader**: to render svg icons
- **Autoprefixer**: to add vendor prefixes to all the compiled CSS
- **MD5**: to encrypt passwords and other data

All these libraries are included in a file called **package.json**, which is used by NPM as a registry of all the dependencies that the project has. Although it works also as a configuration file for the project.

This system, allow us to pack only our source code, keeping all the dependencies in external servers, which makes things much easier in terms of file size and maintainability.

3.2 The folder and file structure

The main folder is called **src**. It contains all the JavaScript files in our project. Considering the fact that we are going to be using Firebase as the backend, these are going to be all front-end related files.

Inside src, we have **index.html**, which is going to be the only pure HTML file in our project. Remember that this is a SPA, therefore we only have one physical HTML page.

At the same level, we can find the **app** folder, and inside we can find **actions**, **components**, **constants**, **reducers**, **index.jsx** and **store.jsx**. Let's review them one by one:

- **Components**: includes all the React components, being **app.jsx** the main one and parent of all the others. The rest are divided in subfolders that correspond to sections in the app. The ones inside **common** are used across the app in different places, that's the reason they are 'common' to different sections.
- **Actions**: here we can find the actions that will trigger changes in the redux state
- **Constants**: there is a unique file that contains definitions and constant values, like messages or action names
- **Reducers**: here we can find the functions that will change the redux state
- **Index.jsx**: this important file specifies the routes of the app, the browser history and the initialisation of the React app
- **Store.jsx**: this file has the configuration to setup redux and firebase to work together

Another important folder is **static**. This one contains all the static assets, like SVGs, PNG images, favicons and other icons, fonts and other external CSS files.

The **webpack** folder contains all the configuration for the webpack module, including the specific modifications for the development and production environments.

The **test** folder includes all the unit tests, although they will not be developed in an initial phase.

The **design** folder includes the original Adobe Illustrator and Photoshop projects of some graphic assets created specifically for Hypatia, like the logo or banners.

The **data** folder includes a backup of the Firebase database in its initial state. This is useful for other users that want to use Hypatia and they need to generate the correct database nodes to make the app work.

Finally, the **dist** folder contains all the generated files when running npm run build, that need to be deployed to production. This folder is not included in the repository, as it's removed and regenerated every time we want to deploy.

There are other configuration related files in the root of the folder, like

- .**gitignore**: used to ignore files we don't want to include in the repository
- .**babelrc**: to setup the configuration of Babel
- .**eslintrc**: to setup the configuration of eslint

- .**firebaserc**: to setup the configuration of firebase deploy
- **browserconfig.xml**: for Microsoft related browsers to load icons
- **database.rules.json**: default Firebase database rules
- **firebase.json**: required by Firebase hosting
- **manifest.json**: required to load icons on Android devices
- **readme.md**: required by Github to provide the repository info

The boilerplate had a file called **server.js**, which was used to create a simple express web server in production. After a few tests with Firebase hosting, it was noticed that this is no longer needed, so I deleted it from the repository. Firebase already creates the required web server to make the app work.

3.3 Setup the repository, clean up and set the router up

In the initial phase of the project, the actual part where we start working with the files, we need to create the repository. I chose Github because it's the most popular website for open source projects and has a lot of useful features, like the issues tracking, the wiki, etc.

Once the repository is up and cloned into my computer, I copied over all the files from the boilerplate and started to clean it by removing all the unneeded stuff. At the same time, I was making sure not to break anything.

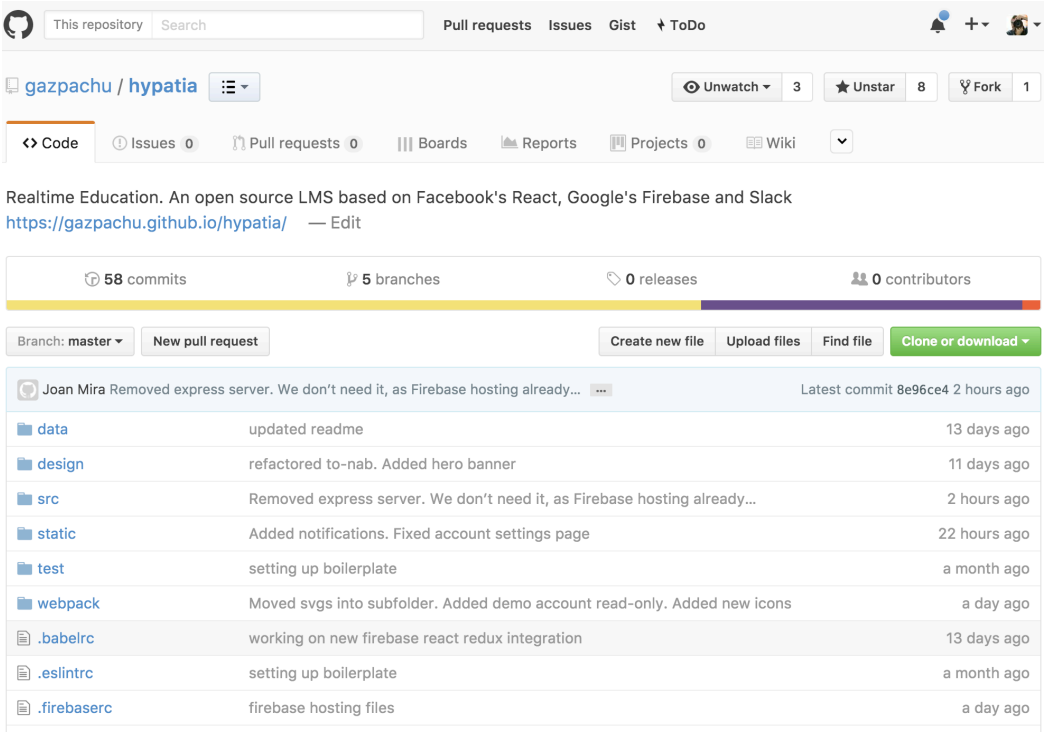

Figure 30. Github project repository screenshot

After the clean-up, I started to add the new dependencies and rearrange a few things in the folder structure.
Then I started defining the routes and creating empty components. The idea was to have some sort of basic navigation that will load pages and on each page a heading rendered by the component attached to the page.

Note: the repository was initially setup in my personal github account but I ended up moving it into the organisation account I created: theonapps

3.4 The anatomy of a React component

A React component [2] can have many different levels of complexity. The smallest interpretation would be a JavaScript function that renders something with a series of properties, for example an Icon. Then we could have a component that does a few more things, like perform some calculations, read inputs or use local state. Finally, we could have some smart components that handle data by connecting themselves to a Flux pattern, like in our case Redux.

In Hypatia we are going to use the three types of components, being the former the most used version. Here is an example of how it looks like:

```
// At the top of the file we import all the dependencies
import React, { Component, PropTypes } from 'react';
import { setLoading } from '../../../actions/actions';
import classNames from 'classnames';
import {connect} from 'react-redux';
import $ from 'jquery';
import Icon from '../lib/icon/icon';

// Set the default values for the component properties
const defaultProps = {

};

// Define the types of the component properties
const propTypes = {
     isDesktop: PropTypes.bool
};

// Create the component Example
class Example extends Component {

     // Here we can define local state variables
     constructor(props) {
          super(props);
```

```
            this.state = { foo: ‘’ }
      }

      // // Perform some stuff when the component is ready
      componentDidMount() {
      }

      // Here is what we will see in the screen
      render() {
            return (
              <section className={`example-component
${this.props.whatever}`}>
                        <h1>Hello World</h1>
              </section>
            )
      }
}

// Add the previously defined propTypes and defaultProps
to the component
Example.propTypes = propTypes;
Example.defaultProps = defaultProps;

// Set the actions that we want to use in the component
const mapDispatchToProps = {
      setLoading
}

// Map the containers from Redux to our component, so we
// can use them as props
const mapStateToProps = ({ mainReducer: { isDesktop, user
} }) => ({ isDesktop, user });

// Connect the component to Redux and export it
export default connect(mapStateToProps,
mapDispatchToProps)(Example);
```

3.5 The top navigation and side navigation

In terms of navigation, as specified in the wireframes, there are two main components: the **topnav** and the sidebar **navigation**.

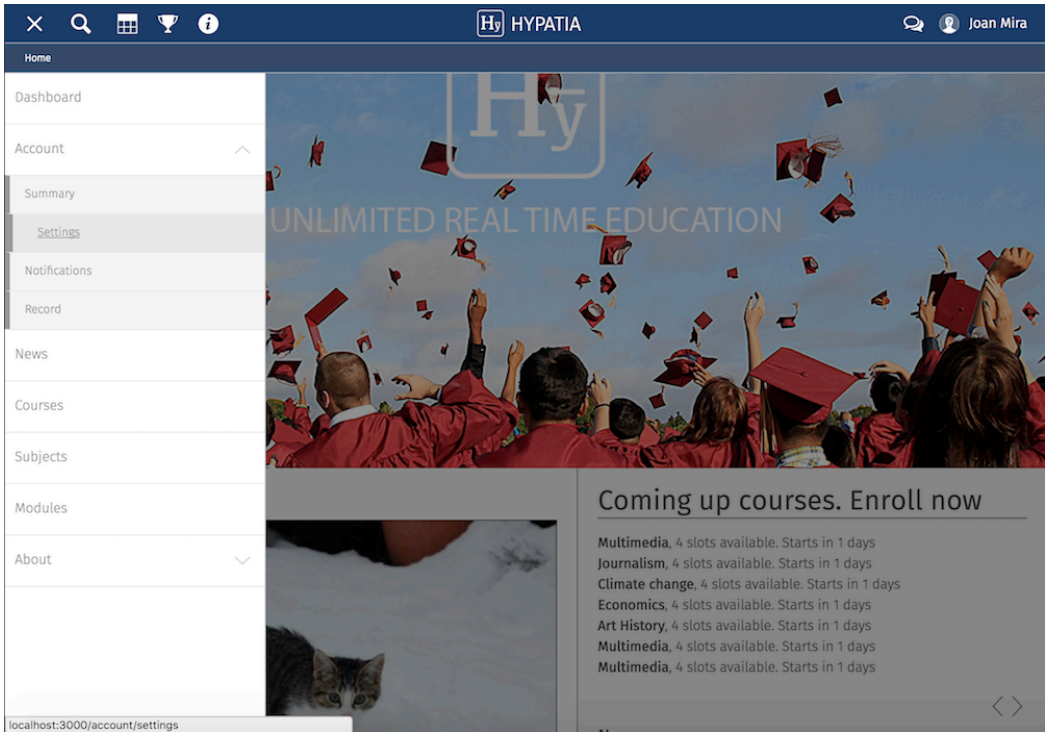

Figure 31. Navigation and v1 of home page screenshot

The topnav, is actually the parent of the sidebar navigation. I decided it this way because both components will always be present at all times and, in terms of communication between the two components, it's better to have both of them connected, so we can pass events from one to the other easily. Like when we need to toggle the sidebar navigation by pressing the hamburger icon in the topnav.

The topnav also holds the sign-up and sign-in forms. These two are very small, as we want the user to be able to sign-up quickly without the need to complete a long form.

In the topnav there are also a few icons that will toggle the dropdown panel with the selected section. These can be, the calendar, the grades, help or chat. These icons are only visible when the user is authenticated.

To handle the authentication and sign-up of users, Firebase [14] provides a method that takes care of everything. We just have to send the correct parameters and catch the success or error response to act accordingly.

More info here: https://firebase.google.com/docs/auth/web/manage-users

At this moment, I implemented these user management following the Firebase documentation [14], although in a near future, I will refactor it to follow the process specified in the redux-react-firebase auth example [16].

As an example of how I'm handling the sign-up of users, this is the method that takes care of it:

```
handleSignup = (e) => {
     e.preventDefault();

     if (this.pwSignup.value === this.pw2.value) {
          $('.js-btn-signup').hide();
          $('.js-signup-loader').show();

          const email = String(this.emailSignup.value);


          firebase.auth().createUserWithEmailAndPassword(e
     mail, this.pwSignup.value).then(function(user) {
               this.saveUser(user);
          }.bind(this)).catch(function(error) {
               $('.js-btn-signup').show();
               $('.js-signup-loader').hide();
               this.props.setNotification({message:
String(error), type: 'error'});
          }.bind(this));
     }
     else {
          this.props.setNotification({message:
PASSWORD_MATCH_ERROR, type: 'error'});
     }
}
```

The handleSignup method receives a click event from the sign-up button. Then it compares the value of the password inputs to check that is equal. If that's not the case, it renders a notification with the error message.

If the passwords match, then it proceeds to hide the sign-up button, shows a small loading spinner, parses the email input into a string and sends it with the password to the Firebase API [14].

If Firebase [14] accepts the data, it returns an object with the new user, which then is sent into another method to create the node for its profile. The small loader spinner hides and the sign-up button is shown again.

If Firebase [14] doesn't accept the authentication (i.e. the user already exists), then it returns an error, which gets rendered into a notification (a small popup panel on the screen).

The sign-in process is very similar.

After a redesign process, the top navigation styles changed. Now it blends with the background and adopts the colours from it, which makes it more dynamic and customisable.

Figure 32. Top nav with the new styles screenshot

3.6 The home page

In the home page, we are requesting to Firebase the most popular courses and the latest posts/news.

To load the posts in the Home component, we follow the method provided by the redux-react-firebase library, which consists on adding decorators to the class:

```
@firebase( [
     'posts'
])
@connect(
     ({firebase}) => ({
     posts: dataToJS(firebase, 'posts'),
     })
)
```

These decorators perform all the operations automatically, which include contacting the Firebase API [14], requesting the posts nodes and adding them to our redux state.

This means, we can render the posts using a classic JavaScript map function:

```
posts.map((post, id) => this.renderItem(post, id))
```

And then we can render each individual post item in the renderItem function. Notice the use of the showdown library to parse the markdown:

```
const converter = new showdown.Converter();

return <li key={id} id={id} ref={`post-${id}`}
className="featured-post">
     <h1className="post-title"><Link
to={`/news/${post.slug}`}>{post.title}</Link></h1>
     <img className="post-image" ref={`post-${id}-img`} />
     <div className="post-meta">
          <p>By <span className="post-
author">{post.author}</span> on <span className="post-
data">{moment(post.data).format('D/M/YYYY')}</span></p>
     </div>
```

```
      <div className="post-content"
dangerouslySetInnerHTML={{__html:
converter.makeHtml(post.content)}}>
      </div>
</li>;
```

3.7 Breadcrumbs and loader

The breadcrumbs component is very simple. It basically parses the URL into a more readable string and renders it in the top nav. To perform this parsing, it listens to changes in the browser history. Then it adds the result string into the redux state, so that it's available to the rest of components in the app.

Figure 33. Detail screenshot of breadcrumbs and icons

The loader component is also very simple. Its purpose is to display a full-screen loading spinner whenever the app is performing an asynchronous operation that needs to wait until the server responds. To show & hide the loader, we use this simple action from anywhere in the app:

```
this.props.setLoading(true); // or false
```

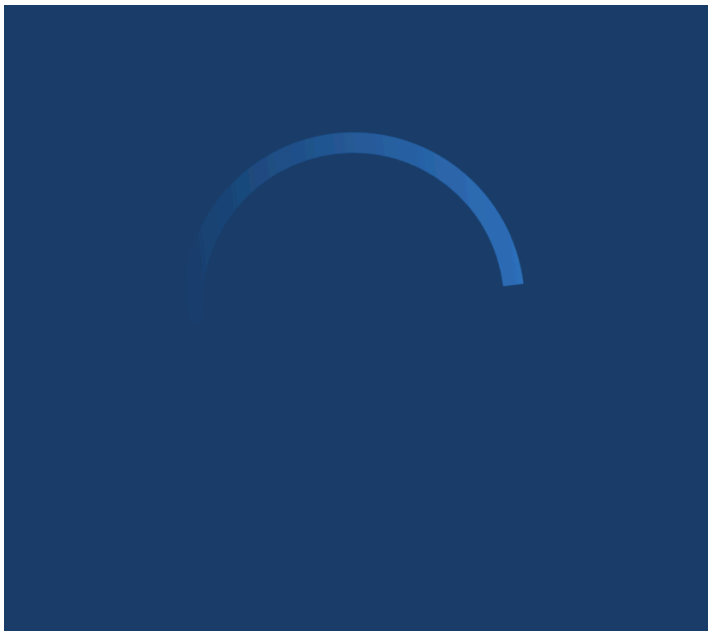

Figure 34. Screenshot of the loader spinner

3.8 Notification and Icon

The notification component is used to display a small popup on the top right side of the screen whenever we need to tell the user about the result of an action. It's usually displayed after a Firebase request is fired and it responds with a success or error message.

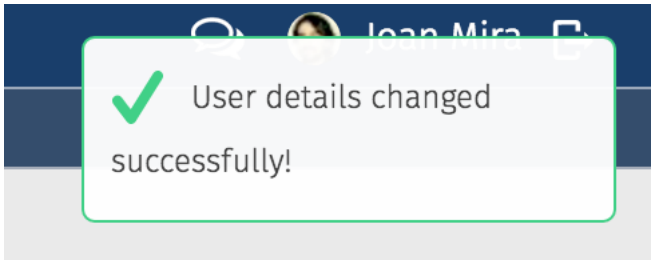

Figure 35. Screenshot of the notification popup module

To call this component, we just need to type the following:

```
this.props.setNotification({message: USER_INFO_CHANGED,
type: 'success'});
```

or for an error:

```
this.props.setNotification({message: String(error), type:
'error'});
```

The icon component, is just a helper to render SVGs icons. In this case, the type of component we are using is not a class but a function based component. This is how it looks like (a simplified version):

```
export default function Icon({glyph, width, height,
className='icon', style}){
     return (
          <svg className={className} width={width}
height={height} style={style}>
               <use xlinkHref={ glyph } />
          </svg>
     );
}
```

As you can tell, there's no class or render method. It's just a function that receives some properties and returns a piece of mark-up. This is how we call it:

```
<Icon glyph={Close} className="icon close-chat" />
```

3.9 The chat component

This is probably the most complex component in the app. It's basically an integration of the Slack API [15], which allows users to chat in groups, channels or directly with other users. The interface is very similar to the Slack app, with the idea that, whoever prefers to use instead the Slack app, can do it.

At the moment, I'm using a BOT to receive and send messages from several Slack Teams. This has a pitfall, which is that users cannot send messages to other users and all the messages are sent by the BOT, therefore, the experience is not completely cross-platform because a user cannot send messages from the Slack app and from the chat section of Hypatia using the same identity.

I'm in the process of refactoring this component to allow a full integration and seamless experience, but it's quite a complex process due to the security and authentication measures.

This is how the component looks like at the moment:

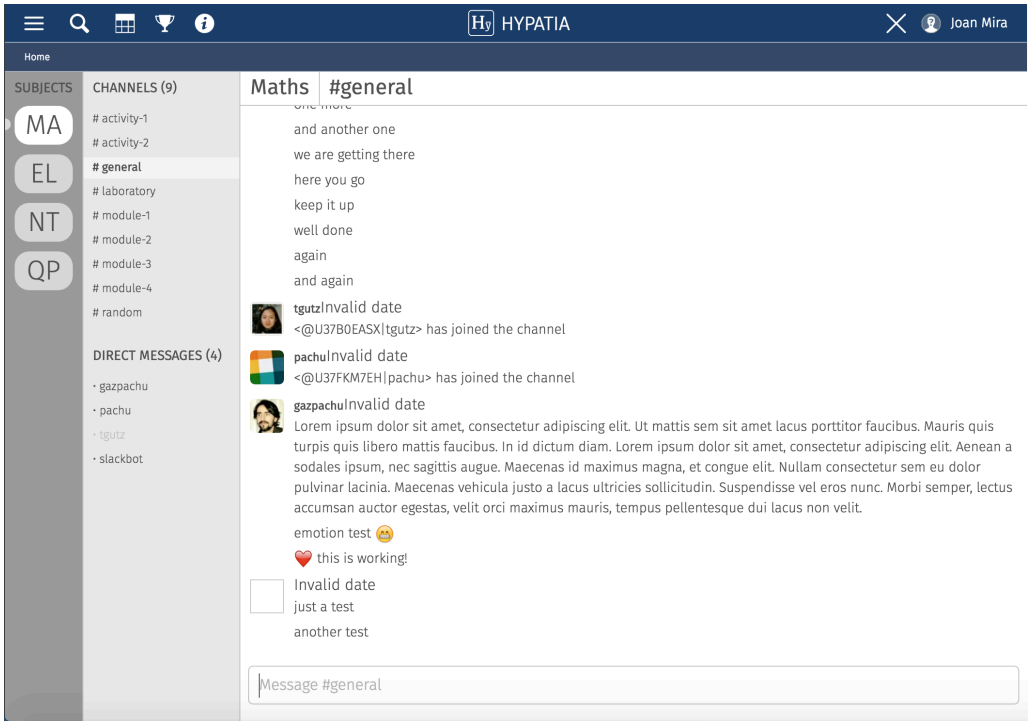

Figure 36. Screenshot of the chat module

This component has also many different sub-routines, like the ones in charge of parsing dates, emoticons or system messages, filtering bad words, handle replies, direct messages and other slack related commands.

In any case, I think this is really the most interesting part of the project. Once is completely integrated, will allow a realtime communication and much better experience than the current eLearning platforms support.

3.10 The listing component

The listing component is used in several pages, as it serves for the same purpose, which is to render a list of items. These items can be courses, subjects, modules, activities or even news.

At the moment, there is a unique layout, which creates each item as a card and displays them in responsive columns up to a maximum of four. In the future, I would like to add other types of layouts, like a single column list or a table.

The listing component also handles the differences between the types. For example, the courses items display the registration fee and the number of users registered, but the other types don't.

The DB queries need to be optimised as well, to fetch only the data required for each instance.

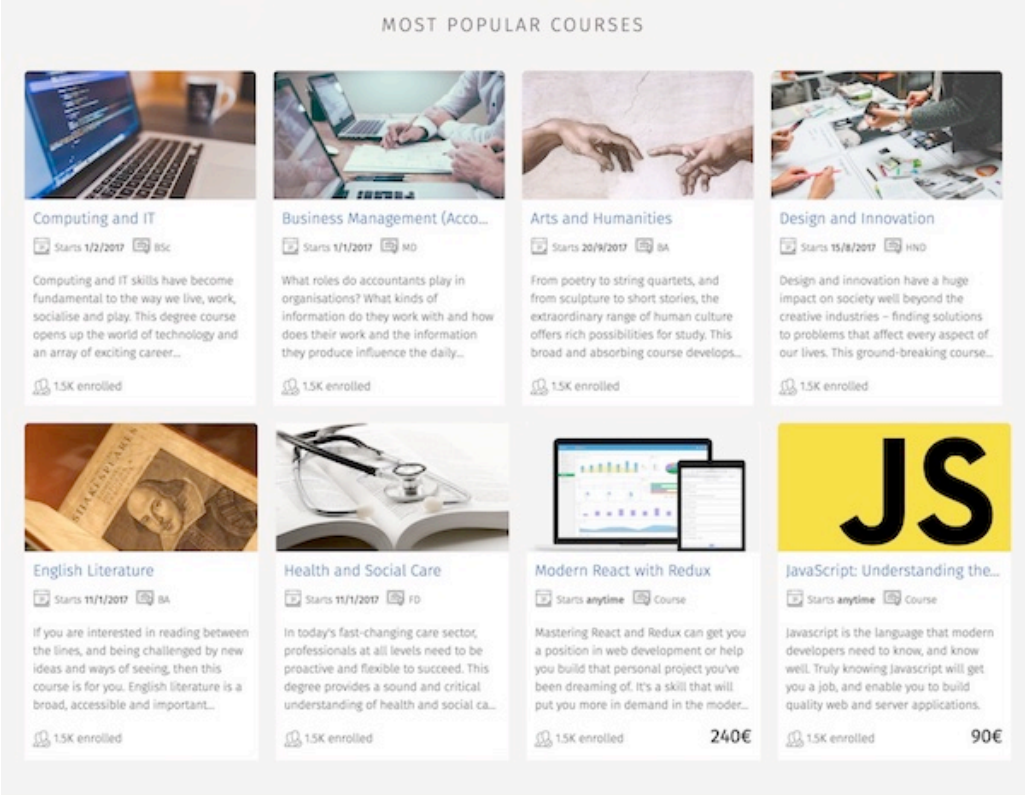

Figure 37. Screenshot of the home page course listing

3.11 The detail component

The detail component was initially conceptualised very similarly to the listing component in the fact that it's also used in several pages. During the development phase, I realised that each type of page needs a substantial number of unique elements, which made me take the decision to split the detail component into individual components, one for each type.

In any case, I will review this setup once the pages are in a more advanced state and figure out if it's better to marge them into a single component to keep things DRY and more maintainable.

Nonetheless, all types of detail pages share the same CSS styles and a very common mark-up, which at least makes the CSS more maintainable and facilitates a possible future merge.

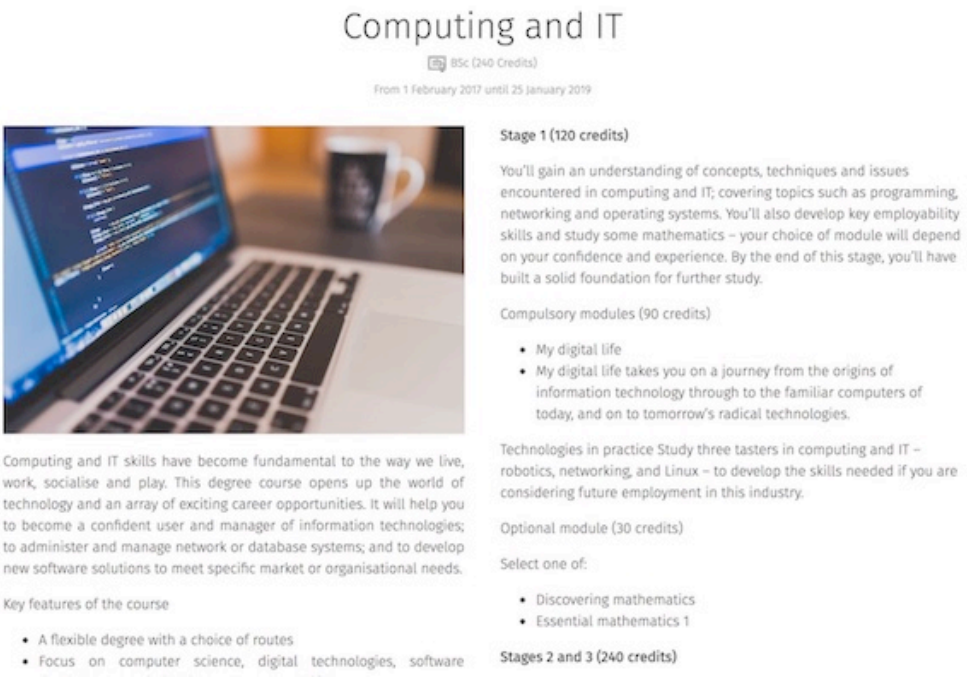

Figure 38. Screenshot of the course detail page

3.12 The user account page

The user account page and component display a couple of forms, one for the password change and another one for the rest of details. Initially, there were two more forms to update the email address and the display name. This was due to how Firebase [14] stores the user account details, which is in a separate location from the rest of the user profile info and using a dedicated API method for each operation.

During the development phase, I realised that I needed the email address and display name to be stored beside the user info details, so I decided to simplify the number of forms.

Initially, I also planned to have a right sidebar with a navigation for the account level pages, but after considering that I will need more space in the future to display information regarding the user's courses and subjects, I decided to remove it.

To update the password, we call this API method and pass the new password value:

```
this.props.user.updatePassword(password.value).then(…)
```

To update the user details, we call the set API method with the path to the user node and we pass the object that contains all the info:

```
this.props.firebase.set(`users/${ user.uid}/info`,
this.state.info).then(…)
```

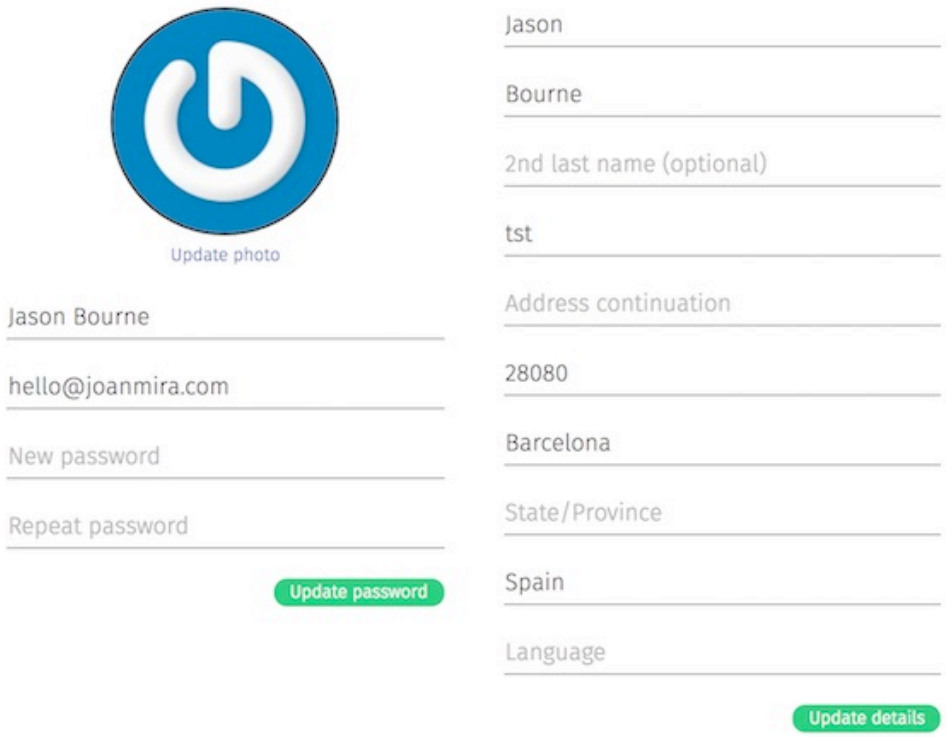

Figure 39. Screenshot of the user settings page

3.13 The dashboard page

The dashboard page purpose is to display a detailed view of the user subjects and activities, specially the current ones and also the latest messages from the chat.

To accomplish this, the component requests to the user node in Firebase, the courses key and its contents. If the user has enrolled in any subject, his node in Firebase will contain a reference to all those subjects and, by extension, to all the activities of those subjects.

Once everything is rendered, the user will be able to download or upload files, access the correct chat group for each subject, review announcements and his latest messages. It's basically a 'bird view' of what's going on at the moment for the user.

At the moment, the announcements and the latest messages are not yet linked with the DB, so the data is hardcoded. The rest of information comes directly from the DB.

The buttons underneath each activity are also not yet linked to any action, but the announcement and download ones will be relatively fast to implement. The upload action has a dependency with the activities hub module, which is still pending to be completed and the chat action could be linked very soon, as the chat rooms are already working.

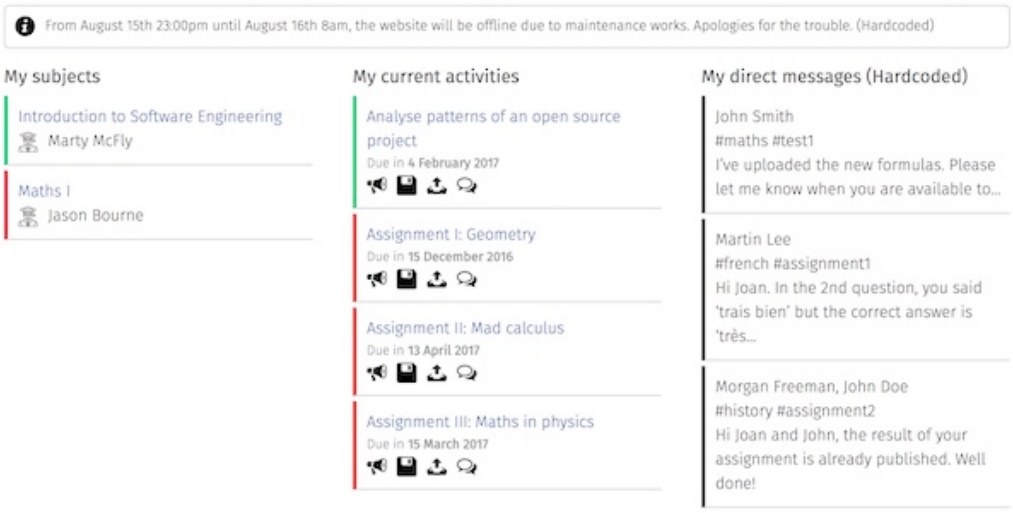

Figure 40. Screenshot from the dashboard page

3.15 The admin panel

Previously, I mentioned in the chat component that it was the most complex one from the app. Perhaps I was wrong. The admin panel, with almost 700 lines of code, is quite big. I will have to refactor it in smaller components to make it more maintainable.

The purpose of this component is to offer the admins the ability to add, edit and remove items from the different content types. The approach in the UI is very simple. In the left sidebar (*), we find dropdowns [18] for

each content type, where we can search and select the one we want to edit or click on the '+' button to create a new one.

The hierarchy of the content is very easy: **courses contain subjects and subjects contain modules and activities**.

On the right content panel, the component displays the detailed view of the item selected on the left side. It uses also the same layout when we want to create a new item.

Most of the elements in the content panel are the same for all content types, therefore I decided to tweak the render method to show & hide them depending on which content type is activated.

Another different aspect of this component, is the ability to upload files to Firebase Storage, which allows the user to have a proper asset management tool. This is done by creating an object like this:

```
this.uploadTask = this.storageRef.child('files/' +
file.name).put(file);
```

We can then track the progress and errors of the file upload with the following method:

```
this.uploadTask.on(firebase.storage.TaskEvent.STATE_CHANGE
D, function(snapshot) { …
```

The admin panel also uses a markdown editor [21] that looks like a traditional WYSIWYG editor, which makes things easier for users.

Most of the elements are straight forward to understand, like text inputs, dropdowns, text editors, date pickers [19][20] and buttons, although probably I should rethink how to display the labels for some of these form elements.

Every content type also has a 'private notes' section to save some information that will not be visible to other users. This is just to leave notes between admins. There's also an 'active' checkbox to allow admins to disable items from being rendered in the page, i.e. in case they need to be reedited or rescheduled.

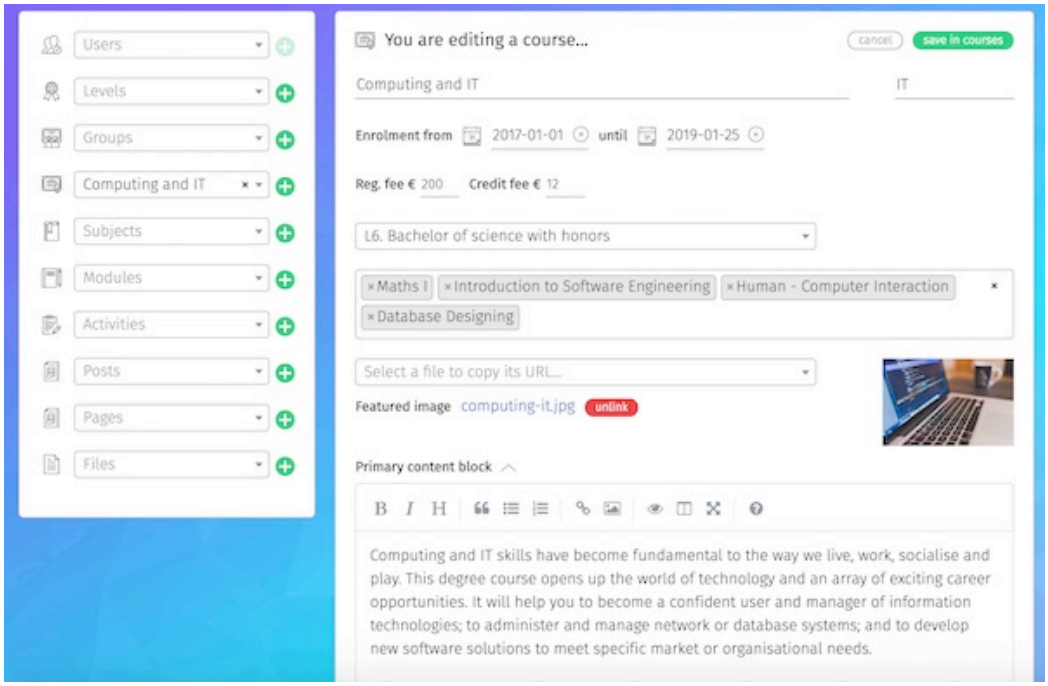

Figure 41. Screenshot of the admin panel page

4. QA and User Testing

QA was performed in the staging site following a CI and continuous refactoring approach. At this moment, there's no issue tracker or Kanban board, although in a near future, I could set it up to track better all the bugs and new features.

In order to keep the code quality high, I have followed the linting rules defined by AirBnb and set them up in the ESLint module.

In the future, a proper unit testing suite will be integrated to cover as many scenarios as possible. This will maximise the integry of the software as it becomes bigger and more complex. Two candidate libraries to accomplish this task are Mocha and Jasmine.

The User Testing focus group came from family members and UOC student groups. I sent them a message requesting for their contribution to review Hypatia and answer this survey. Although the participation was very low, the review of the responses revealed that:

- The labels for input elements in the admin panel and user account pages confuses the users, as they are not always visible
- The admin panel options might be a bit overwhelming

The rest of data obtained is not relevant for the project objectives.

5. Security, hosting and deployment

Firebase already has all its servers secured, all we have to do is adjust the settings and permissions. There are three different areas: authentication, database and storage.

The **authentication module** is separated from the database, which makes the sensitive user data, like the login details, more secured, as they cannot be altered from the database interface. To prevent anonymous users from interacting with the private pages of Hypatia, I enabled an option that forces the user to confirm his email to be able to sign in.

Firebase also requires the user to login again whenever he is going to perform a sensitive operation, like change the password or the email. The system also doesn't let sign up users with an email that is already registered in the DB.

Regarding the **database nodes** to store the business data, there are some rules set to allow write permission only to authenticated users. I also added a value called 'level' under the user info details that if equal to 5, the user gains admin rights. This value can only be changed from the Firebase interface. It's not possible to do it from any page in the app, which prevents users from hacking it and give to their account a level = 5 to access the admin panel.

The **storage settings** are perhaps the easiest to setup, as the only requirement is to give only to authenticated users the write permission. Firebase also has a system that prevents 3rd party websites from hot-linking, so we don't have to worry about other users stealing our bandwidth.

Regarding the **online repository in GitHub**, I had to implement a system to be able to store sensitive data, like API keys and at the same time make them work locally and in production. Things got a bit more complicated after I installed Travis to handle the CI.

At the moment, I'm storing the sensitive data in a file called '.env' in the root folder. This file is loaded by webpack and its content is transformed into environment variables, which means I can read them inside the app using the Node.js syntax: 'process.env.VARIABLE_NAME'. Of course, this file is not committed in the repository. Each user who downloads Hypatia will have to create the file manually and edit its content. The file is also added to .gitignore to prevent the accidental push to the repository.

When deploying to production, if we do it manually (npm run build & firebase deploy), we don't have to change anything, it will work. If we work with Travis, then we will have to enter the API keys in the project settings, in the environment variables section.

The application is currently hosted in **Firebase hosting**. It is a very convenient service, as it has a CLI tools that allow me to deploy just by typing ‘firebase deploy’ in the console. The service takes care of creating the web server and it even has HTTPS by default, which helps increase the security of the app.

The files (images, pdf, videos, etc) are hosted in Firebase storage, which is a cloud service similar to Amazon AWS or Azure. I can either upload new files through the admin panel of Hypatia or directly from the Firebase console, which is quite useful to store files that I don’t need to use in the app, like screenshots, or documents.

Recently, I added Travis CI support to make automatic builds every time the code is pushed to the master branch of the repository. This increases the development experience and makes the setup more professional and scalable. At the same time, it gives information to the public regarding the current stable release.

6. Agile development and community

My plan with Hypatia is to continue its development with an agile methodology. This means I will setup a Kanban/Scrum board, linked with the GitHub repository, to track the current and new stories/features, issues, QA process, etc.

I will also try to get other developers on board, as I am aware that a solo venture has limited possibilities of competing with more stablished companies and teams. This will enable me to organise a more serious work flow, with sprints, pull requests and a calendar of future releases.

To organise the communication of the team, I have setup a slack group. Hopefully, the community will be interested in the project and join me in this adventure! At the moment, already 12 people have showed interest in the project, specially from south-east Asia countries, which shows that there is a real need in those places for quality open source eLearning software.

7. Versions, bugs and pending features

The current release is pre-alpha, being the alpha version expected for mid-February or March 2017. Initially, I was expecting to have a more advanced version ready by this time, but I had some unexpected complications that forced me to prioritise the development of the admin panel, which I initially planned to build on phase II.
At the moment, the application has no P1 issues (as far as I am aware), but there a few details that are not working perfectly all the time, like:

- The overlay animations and on/off states in the sign up/ sign in modules don't work well always
- A few overlapping elements in the mobile view
- Breadcrumbs are not showing the real titles, but a *slugified* version of them

Regarding the pending features, here is a list of the most relevant ones:

- Add other languages (internationalisation)
- Admin module to handle translations (internationalisation)
- Add support for Facebook, Twitter and Google authentication
- Add link to send a reset password email
- Finish the slack integration and implement the channels hash tags across Hypatia
- Complete the activities hub
- Complete the calendar module
- Complete the search module
- Complete the questions and answers module
- Complete the user account pages
- Add the list and table views to the listing component
- Fix the main navigation on mobile view
- Add pagination to the listing component
- Complete the home page hero animation
- Add notification badges to the activities hub and chat icons
- Restyle the notifications popup to look better
- Add filtering and sorting to the listing component
- Add theme support: to allow users create their own visual themes for Hypatia and share them with the community
- Add dynamic navigation
- Admin module for navigation: to allow admins create their own navigation items
- Admin module to create custom layouts with hero sections: to allow admins customise their static pages to look more exciting
- Admin module to create taxonomies for posts, courses and subjects: this will help to organise the content better
- Admin module to resize images: at the moment, some images are too big, which consume precious bandwidth from the users

- Improve DB queries to make the app more scalable and load faster
- Divide the admin component into smaller subcomponents
- Allow users to create PowerPoint alike presentations from the text editor using the ImpressJS library

8. Installation and demo data

At the moment, there are two ways to test the application. The first one (more complicated) is to **clone the project** and follow the installation instructions in the readme file published in the Github project page. This process **has not been fully tested yet**, so there is a chance that the full installation cannot be completed. Test it at your own risk.

On the other hand, the easy option, is to visit the active staging environment, running in a Firebase Hosting server instance. This environment hosts **the latest stable version** and contains some demo data, including courses, subjects, modules, etc. It also has **a user with admin rights**, which is required to access the admin panel. The details of this user accounts are not publicly available at this moment. If you wish to test the admin panel, please contact me at hello@theon.io

9. Conclusions

The main conclusion from this work is that, even if it was tremendously ambitious (given the scale, complexity and timeframe), it is always better to aim as high as possible. In this case, I didn't manage to complete everything (I wonder if anybody could have done it), but the fact that I had a very big load of work, didn't intimidate me. I was truly determined.

Since the beginning, I had a very clear target and set of things I wanted to accomplish with this project. I could see the whole picture, the final result. I could see the product visually in my mind. I knew how to architect it and design it. It was just a matter of time and dedication.

This mind-set is the way of thinking of the doers, the makers, the entrepreneurs and the positive and creative people. This is the way to live life. A way I am passionate about.

I don't regret any of my decisions during this project. If I had to start again, I would probably have approached it with the same spirit and professionality. If I say that it was too ambitious and that I should have designed less modules with less functionality, then probably I would have accomplished much less. So, I can't say that being ambitious was a bad thing. It pushed me to work long hours, during holidays and even engage into an exciting intellectual challenge that ended up possessing me. For a few days, I was what some people call, in "the zone", that moment of intense creativity and illumination that feels like a drug. Everything seems to flow smoothly and the ideas come quickly and non-stop. It's a fantastic feeling that made me progress a lot.

If I look at the results from a critical point of view, I would say that, what bothered me the most, is not being able to foresee that I was going to need an admin panel, since the beginning, to handle the content. I was planning to build the admin panel in a later phase of the project, but soon it was clear that I was going to need it as soon as possible. Otherwise, the perception of completion of the project wouldn't have been the same. The admin panel is what gives to the project its CMS category. Let's not forget that.

This change of plans, forced me to stop the development of other important modules, like the calendar, the activities hub or even the chat system. Although the former is quite advanced and should be ready very soon.

Regarding the planning, it was good to have it. It gave me an idea of the sequence I had to follow and the deadlines I had to meet. In that aspect, is quite similar to how things are done in the real world, although this project gives me much more freedom to change the plans when I need to.

As I mentioned in the previous point, there are a lot of features pending to be added, which doesn't worry me. I have a long-term commitment with this project and my approach with Hypatia is to have it as play-ground, where I can explore things and have the freedom to make anything I like, as opposed to the real world strict requirements of clients and businesses.

I hope this work will help someone in a near future to make the World a better place by giving the opportunity to learn to many people.

10. Glossary

MVP: [1] Minimum Viable Product is a product with just enough features to gather validated learning about the product and its continued development.

CI: [1] In software engineering, continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day.

CMS: [1] A content management system (CMS) is a computer application that supports the creation and modification of digital content.

LMS: [1] A learning management system (LMS) is a software application for the administration, documentation, tracking, reporting and delivery of electronic educational technology (also called e-learning) courses or training programs

DB: [1] A database is an organized collection of data. It is the collection of schemas, tables, queries, reports, views, and other objects.

Slugify: [1] a slug is the part of a URL that identifies a page in human-readable keywords. Slugify is the action of formatting a normal text string into a slug.

QA: [1] Quality assurance (QA) is a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers.

P1: in Software Engineering QA, a P1 issue has the highest priority of all.
CLI: [1] A Command Line Interface is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

API: [1] In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building application software.

WYSIWYG: [1] an acronym for "what you see is what you get". In computing, a WYSIWYG editor is a system in which content (text and graphics) can be edited in a form closely resembling its appearance when printed or displayed as a finished product, such as a printed document, web page, or slide presentation.

UI: [1] The graphical user interface (GUI), mostly referred as UI, is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators. In the scenario of a website, the UI refers to all the graphic elements, links and displayed elements that the user can interact with.

UX: [1] User experience (UX) refers to a person's emotions and attitudes about using a particular product, system or service.

DRY: [1] In software engineering, don't repeat yourself (DRY) is a principle of software development, aimed at reducing repetition of information of all kinds, especially useful in multi-tier architectures.

CSS: [1] Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a mark-up language.

SASS: [1] Syntactically Awesome Style Sheets is a scripting language that is interpreted into Cascading Style Sheets (CSS). It allows to create more structured and optimised CSS code.

BOT: [1] Computer program that imitates the human behaviour.

SVG, **PNG**, **JPG**: computer image formats.

URL: [1] Uniform Resource Locator, commonly informally termed a web address (a term which is not defined identically) is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

CPU: [1] Central Processing Unit, is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

JSON: [1] JavaScript Object Notation, is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.

CRUD: [1] In computer programming, create, read, update and delete (as an acronym CRUD) are the four basic functions of persistent storage.

NPM: [1] npm is the default package manager for the JavaScript runtime environment Node.js.

AJAX: [1] Asynchronous JavaScript And XML is a set of web development techniques using many web technologies on the client-side to create asynchronous Web applications.

SEO: [1] Search Engine Optimisation is the process of affecting the visibility of a website or a web page in a web search engine's unpaid results—often referred to as "natural", "organic", or "earned" results.

SPA: [1] Single Page Application is a web application or web site that fits on a single web page with the goal of providing a user experience similar to that of a desktop application.

CTA: [1] In marketing, a call to action (CTA) is an instruction to the audience to provoke an immediate response, usually using an imperative verb such as "call now", "find out more" or "visit a store today".

MEAN: [1] MongoDB, ExpressJS, AngularJS and NodeJS is a free and open-source JavaScript software stack for building dynamic web sites and web applications.

IOS: [1] iOS (formerly iPhone OS) is a mobile operating system created and developed by Apple Inc. exclusively for its hardware.

NoSQL: [1] A NoSQL (originally referring to "non SQL", "non relational" or "not only SQL")[1] database provides a mechanism for storage and retrieval of data which is modelled in means other than the tabular relations used in relational databases.

Denormalization: [1] process of trying to improve the read performance of a database.

Flux: a coding pattern for data management usually in React apps

DOM: [1] The Document Object Model (DOM) is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.

ES5, **ES6**: latest versions of the JavaScript programming language.

Transpiling: [1] A source-to-source compiler, transcompiler or transpiler is a type of compiler that takes the source code of a program written in one programming language as its input and produces the equivalent source code in another programming language.

HTML: [1] HyperText Markup Language (HTML) is the standard mark-up language for creating web pages and web applications.

UML: [1] Unified Modelling Language (UML) is a general-purpose, developmental, modelling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

Isomorphic: a web application that can run JavaScript, both in the back-end and the front-end.

Snippet: [1] Snippet is a programming term for a small region of re-usable source code, machine code, or text.

11. Bibliography

[1] **Wikipedia glossary terms** [consulted from Nov 2016 until Jan 2017] Online at <http://www.wikipedia.com/>

[2] **React docs** [consulted from Nov 2016 until Jan 2017] Online at <https://facebook.github.io/react/docs>

[3] **Amazon vs Firebase** [consulted on Nov 2016] Online at <https://www.quora.com/Which-is-better-cloud-server-Amazon-AWS-or-Firebase>

[4] **A look at the new firebase, a powerful Google platform** [consulted on Nov 2016] Online at <https://scotch.io/bar-talk/a-look-at-the-new-firebase-a-powerful-google-platform>

[5] **Most popular React boilerplates** [consulted on Nov 2016] Online at <http://andrewhfarmer.com/starter-project/>

[6] **ESTE React boilerplate docs** [consulted on Nov 2016] Online at <https://github.com/este/este>

[7] **Express, Redux React boilerplate docs** [consulted on Nov 2016] Online at <https://github.com/DimitriMikadze/express-react-redux-starter>

[8] **Firebase UI docs** [consulted on Nov 2016] Online at <https://github.com/firebase/FirebaseUI-Web>

[9] **React router Firebase auth docs** [consulted on Nov 2016] Online at <https://github.com/tylermcginnis/react-router-firebase-auth>

[10] **Why you should never use Firebase realtime database** [consulted on Dec 2016] Online at <https://crisp.im/blog/why-you-should-never-use-firebase-realtime-database>

[11] **What different qualification levels mean** [consulted on Dec 2016] Online at <https://www.gov.uk/what-different-qualification-levels-mean/list-of-qualification-levels>

[12] **Use HTML5 to resize an imatge before upload** [consulted on Dec 2016] Online at <http://stackoverflow.com/questions/23945494/use-html5-to-resize-an-image-before-upload>

[13] **Automatic Deployment to firebase github travis** [consulted on Dec 2016] Online at <https://marlosoft.net/posts/automatic-deploy-firebase-github-travis.html>

[14] **Firebase docs** [consulted from Nov 2016 until Jan 2017] Online at <https://firebase.google.com/docs>

[15] **Slack docs** [consulted from Nov 2016 until Dec 2016] Online at <https://api.slack.com>

[16] **Redux React Firebase docs** [consulted from Nov 2016 until Jan 2017] Online at <https://github.com/tiberiuc/redux-react-firebase>

[17] **The LMS market glaciar is melting** [consulted on Jan 2017] Online at <https://techcrunch.com/2016/09/02/the-lms-market-glacier-is-melting>

[18] **Select2 docs** [consulted on Jan 2017] Online at <http://select2.github.io/>

[19] **React date picker docs** [consulted on Dec 2016] Online at <https://github.com/Hacker0x01/react-datepicker>

[20] **MomentJS docs** [consulted from Dec 2016 until Jan 2017] Online at <http://momentjs.com/>

[21] **SimpleMDE docs** [consulted from Dec 2016 until Jan 2017] Online at <https://github.com/NextStepWebs/simplemde-markdown-editor>

[22] **Showdown docs** [consulted from Dec 2016 until Jan 2017] Online at <https://github.com/showdownjs/showdown>

[23] **React router docs** [consulted from Nov 2016 until Jan 2017] Online at <https://github.com/reacttraining/react-router>

[24] **ImpressJS sample code** [consulted on Jan 2017] Online at <https://github.com/impress/impress.js>

12. Annex

The latest version of all these resources (except the Gantt diagram) is available from the project's website.

Demo: currently hosted in the Firebase server and available at: https://hypatia-8d923.firebaseapp.com/

Presentation: built in HTML5 and available at: https://theonapps.github.io/hypatia/presentation.html

Source code: including the graphic design files available at: https://github.com/theonapps/hypatia

Hypatia_UX.pdf: project sitemap and information architecture

Hypatia_architecture.pdf: software architecture proposal

Hypatia_Gantter_diagram.png: project's Ganter diagram