Joel Vainikka

Full-stack web development using Django REST framework and React

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Thesis

16 May 2018

Author Title Number of Pages Date	Joel Vainikka Full-stack web development using Django REST framework and React 34 pages + 3 appendices 16 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Software Engineering
Instructor	Peter Hjort, Senior Lecturer

The purpose of this thesis was to investigate Finnish Esports League's (FEL's) web service and how it could be split into two separate services. FEL's old service is on WordPress, which consists of two different sides, a news site and a tournament platform for Esports players. The goal of the split is to ease further developing of the service and maintenance. Secondly, this study investigates full-stack development and frameworks chosen for the new project.

A full-stack developer is an expert who understands every aspect of the stack (toolset). Full-stack developing has been gaining popularity and stacks such as MEAN (MongoDB, Express.js, AngularJS, Node.js) have been favoured for their uniform programming language.

A new service was built with Django REST framework. Building a complete service can be achieved with Django only, but the development team wanted to separate server and client side. Therefore, REST was used for back-end, and the front-end application was created with React, which connects to back-end using REST API.

The project did not end while writing this thesis, but the development continues. As an experience, working with Django REST was smooth. Splitting up server and front-end development was a good decision and it made developing easier and helped to manage the project. Because frameworks were new for the developers, getting to know them was time consuming.

Keywords	Django, Full-stack, Python, React, RESTful, WordPress

Tekijä Otsikko Sivumäärä Aika	Joel Vainikka Full-stack-verkkosovelluskehitys Django REST -sovelluskehyksen ja React-JavaScript-kirjaston avulla 34 sivua + 3 liitettä 16.5.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	Software Engineering
Ohjaaja	Lehtori Peter Hjort

Insinöörityön tarkoitus oli tutkia Suomen Elektronisen Urheilun Liigan (FEL) verkkopalvelua ja sitä, miten se toteutettaisiin uudelleen jakamalla palvelu kahteen erilliseen osioon. FEL:n verkkopalvelun vanha järjestelmä toimi WordPress-kehyksellä ja sivusto koostui uutissivustosta ja pelaajille tarkoitetusta turnauspalvelusta. Jaon tarkoituksena oli hakea helpotusta jatkokehittämiseen ja ylläpitämiseen. Samalla insinöörityössä tutkittiin, mitä on full-stack-sovelluskehittäminen ja mitkä sovelluskehykset valittiin uuteen projektiin.

Full-stack-sovelluskehittämisellä tarkoitetaan, että sovelluskehittäjä hallitsee ohjelmistokehitysprojektin jokaisen osa-alueen ja kehitystyökalun. Full-stack-kehittämisestä on tullut suosittua, ja valmiita kehyspaketteja, kuten MEAN (MongoDB, Express.js, AngularJS, Node.js), suositaan niiden yhtenäisen ohjelmointikielen vuoksi.

Uuden järjestelmän palvelinpuolta alettiin rakentaa Django REST -kehyksellä. Django-kehyksellä verkkosivujen luonti onnistuisi yksistään eikä vaatisi muita kehyksiä, mutta kehittäjät halusivat erottaa palvelinpuolen kokonaan käyttöliittymästä. Siksi selainpuolen näkymien hallintaan otettiin käyttöön React-kehys, jolla on helppo luoda itsenäinen ohjelmisto vain käyttöliittymää varten. Se ottaa yhteyden palvelimeen REST-rajapinnan kautta.

Lopullinen järjestelmä oli insinöörityöraporttia kirjoitettaessa vielä kesken ja sen työstäminen jatkui. Kokemuksena Django REST- ja React-kehyksillä työskentely oli sujuvaa. Palvelin- ja selainpuolen ohjelmiston jakaminen erilleen oli hyvä ratkaisu, ja se helpotti kehittämistä ja ylläpitämistä. Koska kehitysalustat olivat kehittäjille uusia, ongelmia esiintyi lähinnä vain uusien kehyksien kanssa työskentelyssä. Uusiin kehyksiin tutustuminen vei kehityksessä aikaa.

Avainsanat	Django, Full-stack, Python, React, RESTful, WordPress

Metropolia

Contents

List of Abbreviations

API	Application programming interface
FEL	Finnish Esports League
CMS	Content Management System
CSS	Cascading Style Sheet
CS: GO	Counter-Strike: Global Offensive
HS	Hearthstone
HTML	Hyper Text Markup Language
JSON	JavaScript Object Notation
JSX	JavaScript XML
MySQL	My Structured Query Language
npm	Node packet manager
pip	Python packet manager
REST	Representational state transfer
SPA	Single page application
UDP	User Datagram Protocol
URL	Uniform Resource Locator

1 Introduction

Starting a new software project is always exciting, and there are currently many ways of creating content for the web. To choose the right frameworks can be a challenge. Although it is tempting to try out the newest and trendiest frameworks, it might not be the smartest decision. Comparing different frameworks is time consuming and experts have different views on what features a good framework should have.

As frameworks in web development evolve, front-end and back-end frameworks tend to get tied to one another creating stacks of tools. As a result, there is a need for new kind of developers called full-stack developers. A full-stack developer is a developer who understands every aspect of the full-stack and has a wide understanding of the different parts of the project without having deep knowledge of every aspect of the project.

This study investigates the structure of the web-service provided by Finnish Esports League (FEL). The developer team decided to split the current website into two separate services to make it easier to manage the services in the future.

This thesis investigates how frameworks are chosen and how a project gets started by using certain frameworks. This study also focuses on what it is like to work with the chosen frameworks. The project is carried out by using Python's Django REST framework and React JavaScript library.

This project was commissioned by Finnish Esports League, when they decided to make a change to a new system. The study covers the choices that were made by the development team at the beginning of the project and how working with the chosen frameworks turned out. Finally, this project investigates how problems caused by heavy back-end, messy front-end and lack of flexibility in system design were solved.

2 Finnish Esports League

This section investigates what Finnish Esports League is and what kind of services they provide. Also, problems in the current system are listed and finally, a proposal for replacement will be presented.

2.1 Service Introduction

Finnish Esports League (FEL) is a small company founded in 2015, which aims at promoting Esports in Finland by organizing seasonal leagues and small tournaments for computer and console games. Two main games that the company organizes are Counter-Strike: Global Offensive (CS: GO) and Hearthstone (HS) but they also have tournaments for games such as FIFA and NHL. FEL writes its own news about ongoing leagues and things that are currently going on in the Finnish competitive Esports scene.

This study investigates the structure of the web-service provided by Finnish Esports League. The developer team decided to split the current website into two separate services to make it easier to manage the services in the future. The purpose of the web-service is to provide news for fans. Also, players login and register their teams to tournaments through this website.

Just as in any sports there are two types of competitions on the FEL’s web-service, individual and team-based sports. Esports games such as CS: GO and HS are good examples of both types of competitions, because they have special characteristics that need to be taken into consideration when making a service that tries to generalise a service design.

Counter-Strike: Global Offensive

Counter-Strike: Global Offensive (CS: GO) is a first-person shooter (FPS) game developed by Hidden Path Entertainment and Valve Corporation. The game is fourth in the series and was released in August 2012 [1.] and has since grown to be the one of the most viewed Esports game to date according to Newzoo. [2.]

In the game two teams, the Terrorists and the Counter-Terrorists play against each other. Both teams have their own objectives such as defusing or plating a bomb, but the winner of one round can be decided when one team eliminates the other one before the objective is completed or time runs out.

CS: GO Tournament Rules

In tournament matches, the basic rules of Electronic Sports League's (ESLs) are applied. One tournament can, for example, be ran in a single-elimination bracket with each match playing out as a best-of-three series.

In the game there are seven maps in the official map pool. Of these seven maps, the number of maps played in one match is decided by series. A best-of-three series map veto is illustrated in figure 1.

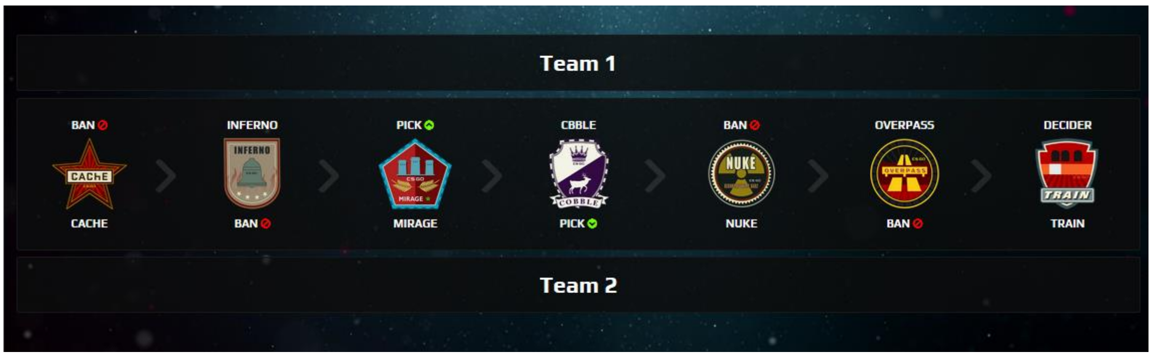


Figure 1. Illustration of a best-of-three match's map veto process.

Rules of map veto vary, but the above demonstrated method is the most common way to veto maps. One played map consists of 30 rounds and teams switch sides after 15 rounds. The first team to get 16 rounds wins. In case of draw, usually six rounds overtime is played until winner is decided.

Although the rules stay unchanged, tournament structure and map veto vary in each tournament. Complexity of the CS: GO's tournament rules are one critical factor to be noted in the match automation. Designing a static service for creating one type of tournament makes it challenging to make changes or organizing other types of tournaments in future.

Hearthstone

Hearthstone is digital card collection game developed by Blizzard Entertainment. The goal of the game is to defeat the opponent’s hero character by using cards. The player has a deck of 30 cards the content of which is based on selected hero. There are nine heroes in the game, which are unlocked by playing the game. Each hero also has some special skill in their use. [3.]

At the beginning of the game both players have 30 health points and three to four cards, depending who starts first. Using cards and hero’s special skill requires mana points. Each round player’s mana points reset, and player receives one card from deck. At the beginning both players have one mana point to use, which raises every round by one up to maximum of ten. [3.]

Hearthstone Tournament Rules

Hearthstone uses last hero standing format, which is essentially a best-of-five series. Both players choose four heroes (card decks), after which they both ban one hero? for each other. After that, the players choose the starting deck. The winning player continues with the winning deck and the loser picks one deck from remaining decks available.

2.2 Service Structure

The current FEL service runs on WordPress. It connects with a third-party CS: GO server rent service called Dathost and the eBot match manager. These services are used to automate matches.

2.2.1 Frameworks

This section investigates which frameworks were used to build the FEL service. It also discusses the reasons for choosing them.

WordPress

WordPress is an open source content management system (CMS) which was released in 2003. It is written on PHP programming language and uses My Structured Query Language (MySQL) database for data storing. WordPress is widely used to create blogs and news sites. According to WordPress website, it is also the platform of choice for over 30% of all sites across the web. [4.] A content management system (CMS) is a software for managing content. It usually includes features such as content creation and editing. [5.]

The community around WordPress has grown since its initial release and now there are tens of thousands of plugins and thousands of themes available. [6; 7] This makes it an excellent choice for people who have limited tech experience to get a webpage up and running. [4.]

For Finnish Esports League (FEL), applying WordPress was an excellent choice of platform to start running the business. When a team of professional writers started publishing news about Esports, they started to get visibility.

AngularJS

AngularJS in an open source JavaScript framework developed by Google. [8, 3] AngularJS uses Model–view–controller (MVC) architecture for building web applications. [9, 2.] Because of these characteristics, AngularJS is an excellent choice for single-page applications (SPAs). SPA webpages are generated by user iteration and do not require the whole page to load again, but to render the page again based on the data.

AngularJS was added to FEL service during the development to help showing data in real time. It is used to fetch real-time data for frontpage widget of ongoing league matches and a league match-page to show map veto in real time. This removes a need for user to refresh the page manually to see if the opponent team has made a choice.

Bulma CSS framework

Bulma is a free and open source Cascading Style Sheet (CSS) framework based on Flexbox. [10.] Bulma offers a variety of building blocks for creating flexible designs.

CSS framework is a compilation of pre-set rules for Hyper Text Markup Language (HTML) styling. Commonly CSS frameworks come packed with a grid system, which allows easily to build a webpage out of boxes that will scale on desktop and mobile devices. Larger frameworks include certain styling so keeping up a consistent look for website is made effortless.

2.2.2 Structure

As mentioned earlier, the FEL service operates on WordPress, which is heavily modified for FEL’s use. The service connects with some third-party services such as a CS: GO server rent service called Dathost and a match manager called eBot. This section investigates the structure trough out an example of how FEL service handles one league match for CS: GO.

eBot

eBot is a tool to help in CS: GO server automation and make it easier to create matches with given configurations. eBot requires its own server and SQL-database to operate. This eBot server is then linked with the game server host through a control panel. The eBot server should be located close to the game server, because eBot uses User Datagram Protocol (UDP) connection. Because this is an unreliable protocol, packet loss can cause problems with the matches.

FEL league consists of a set number of matches, where each team competes against one another during the league season. FEL creates all these league matches at the beginning of league season. Each match has specified starting date by default on Sundays at 6 PM. This date can be changed of both teams agree on this.

On the match day, once captain from both teams has checked in 30 minutes prior to starting time, back-end calls Dathost’s Application programming interface (API) to start up the server. The next back-end waits until map veto between two teams is done. After map veto, FEL service adds match with configurations to eBot's database through SQL insert query. After the match is created, it is started with another SQL query and eBot establishes UDP connection to the game server. The structure of the service as well as the connections to third-party services is shown in figure 2 below.

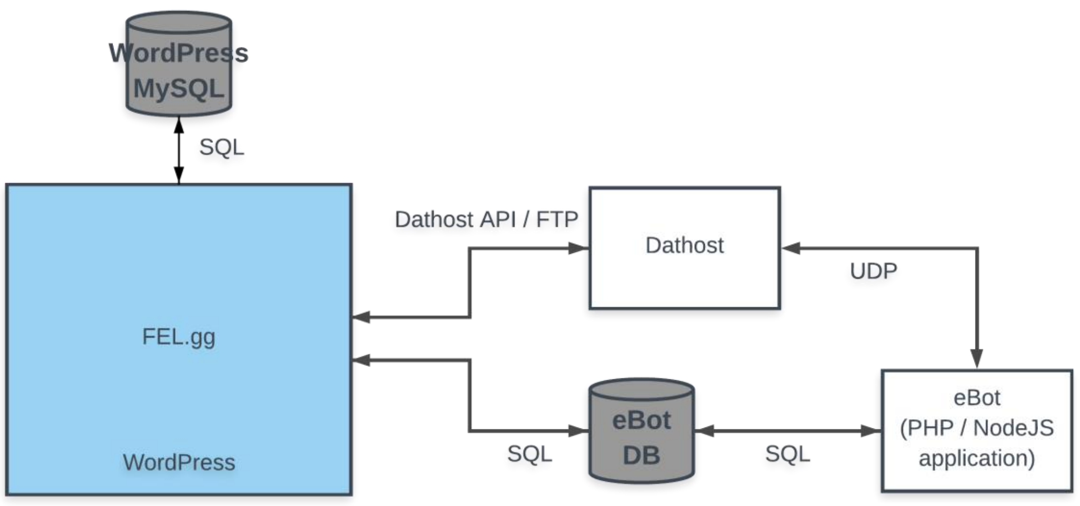


Figure 2. The service structure and its connections to third-party services.

When one match is over, eBot stores the results of the match in its own database. This information can later be fetched if needed. Servers shut down automatically after one hour of idle time.

Without CS: GO, the service could run alone without any connections to other third-party services, because all the other games rely on users setting up the match with given rules and informing service about the result. This is as far as automation can go with games such as HS, FIFA and NHL. Since Valve has given the rights for customers to host their own servers to Valve's games, this has allowed the business to grow on server rental.

2.3 Problems

Pointing out a single problem can be difficult, because several small problems tend to accumulate. This chapter discusses why these problems exist and how they together create larger problems. The current service is a well-functioning system but comes with some drawbacks such as customization and flexibility of the service.

CSS Conflicts

CSS conflicts are not a rare thing in front-end development. There are ways to avoid them and some good practices to keep the CSS reusable and consistent. In FEL’s case everything that should not be done, was done and everything that can conflict, did.

FEL uses Vantage theme by SiteOrigin, which was later discovered in development to conflict with Bulma CSS framework as well as other CSS frameworks. While Bulma’s grid components worked just fine, other components of the framework conflicted with the Vantage theme. The problem laid in generic class names that happened to be same in Bulma and Vantage theme. This made Bulma mostly useless.

All this had caused overwriting CSS stylings inline all over the project files, which caused an enormous search operation when making minor changes to styles. The theme also conflicted with other frameworks, and the options were to write own styles or fix conflicts.

Request Overflow

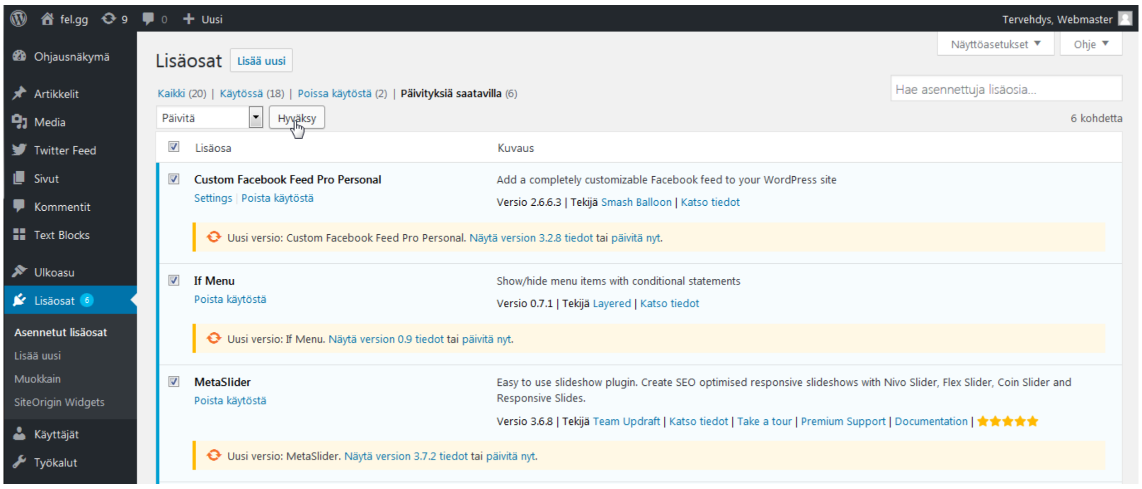
Development team realized that WordPress was not the best choice of platform for this type of automated service. There was a need to bring more tools to help the creating process which complicated things. One of those complications was AngularJS, which was used to refresh data in real time.

There was a need to refresh data in real time on match pages where participants can see who has checked in to upcoming match. Also, map veto needs to refresh data in real time, and the frontpage needed a widget to show ongoing league matches in real time. This caused a vast number of requests to database.

Because default time for league matches was set on Sunday evening, most of the teams accepted that the match will be played at that time. This caused a rush hour once a week on Sunday evening. Requests made frequently by frontpage and match page caused the server to crash on multiple occasions.

Maintenance

The service relies on number of different third-party widgets and plugins that are used to show the social media feed and configure WordPress settings. Updating these should be easy through the admin panel and in a normal situation this is how it will be done, as shown in figure 3 below.



Lisäosat Lisää uusi

Kaikki (20) | Käytössä (18) | Poissa käytöstä (2) | Päivityksiä saatavilla (6)

6 kohdetta

Lisäosa	Kuvaus
Custom Facebook Feed Pro Personal Settings | Poista käytöstä	Add a completely customizable Facebook feed to your WordPress site Versio 2.6.6.3 | Tekijä Smash Balloon | Katso tiedot
Uusi versio: Custom Facebook Feed Pro Personal. Näytä version 3.2.8 tiedot tai päivitä nyt.	
If Menu Poista käytöstä	Show/hide menu items with conditional statements Versio 0.7.1 | Tekijä Layered | Katso tiedot
Uusi versio: If Menu. Näytä version 0.9 tiedot tai päivitä nyt.	
MetaSlider Poista käytöstä	Easy to use slideshow plugin. Create SEO optimised responsive slideshows with Nivo Slider, Flex Slider, Coin Slider and Responsive Slides. Versio 3.6.8 | Tekijä Team Updraft | Katso tiedot | Take a tour | Premium Support | Documentation | ★★★★★
Uusi versio: MetaSlider. Näytä version 3.7.2 tiedot tai päivitä nyt.	

Figure 3. The WordPress admin panel plugin manager.

In the initial stages of the development a decision was made that the whole WordPress project would be in Git version control. This means that every time there is an update to any third-party plugin, it must be first updated on local development environment, which is then pushed to version control master branch and later moved to the server where the service is hosted. After all that the plugins had to be restarted manually to get them running right again. This was probably the easiest problem to avoid in the first place.

Service Flexibility

The service was built in a hurry; thus, it was not as flexible as developers wanted. Building the service was started with minor changes such as chancing CS: GO map pool or adding a new game for users to compete in. These two cases are discussed below.

Changing CS: GO’s map pool required developers to edit the maps directly to the source code. Even though chance such as this is never made in a hurry, it is something that anyone should be able to change if they have the rights. Data which can change over time, should never be hardcoded.

FEL wants to keep adding new games for the player to compete. Adding a new game to FEL service was quite tricky as well. Knowing that each game will be handled almost in the same way resulted in repeated code, which was then modified slightly to different games.

3 Development Frameworks

This section investigates full-stack development and what development environment virtualization means. Also, the frameworks, which were chosen for the FEL's new project will be discussed in this section.

3.1 Full-Stack Development

Full-stack development is rarely talked discussed as such; the different kinds of developers will also be dealt with. A full-stack developer is someone who understands every layer of the full-stack. This includes not only front- and back-end but also project management, system infrastructure and databases. In summary a full-stack developer might have a wide knowledge of all the layers, instead of deep knowledge of a particular layer. [11.] Below is a quotation from an article in Udacity's, which summarises full-stack developer well:

> Ultimately, what distinguishes a successful Full Stack Web Developer is not a superhuman ability to do everything, but rather, to understand everything. [12.]

One of the most popular older stacks is LAMP, which includes Linux operating system, Apache HTTP Server, MySQL database and PHP programming language. The same stack for Windows is called WAMP and on macOS MAMP. [13.] When LAMP was released, the stack size was small, and at that time it was enough to build dynamic webpages. Since then stacks have grown to be bigger and more demanding for developers.

A good example of ready stacks is MEAN (MongoDB, ExpressJS, AngularJS, NodeJS), which was first introduced by MongoDB developer Valeri Karpov in 2013. The idea behind this stack was to have a uniform language through the whole stack, which in this case was JavaScript. This makes it easier for the developer to work on both ends of the project. [14.]

Defining usual stacks is difficult. Seeing some new-comers like Python Django on server side and React trying to compete against Angular, it becomes increasingly a matter of opinion of the developers to choose his or her own tools to build a stack instead of choosing a ready stack. LAMP and MEAN are still in use and they are the first ones to come across when trying to find out which stack to use. Since Django is written with Python, which is one of the most popular programming languages in the world, it has been growing popularity in web development. [15.]

3.2 Virtual Development Environment

There are now more tools available than ever before and installing development environment has become a new issue since every project might have a completely different toolset. Stacks now rely on packet managers and virtual environments which can reproduce a project specific environment effortlessly. Being able to reproduce development environments easily helps developers significantly; they know that the development environment works on any system.

Tools such as Docker and Python's virtualenv are important parts of full-stack, as well as packet managers like Python packet manager (pip) and Node packet manager (npm) that allow installing dependencies effortlessly from dependency file. Example of Python's virtualenv, where two versions of Python run inside virtual environments is illustrated in figure 4.

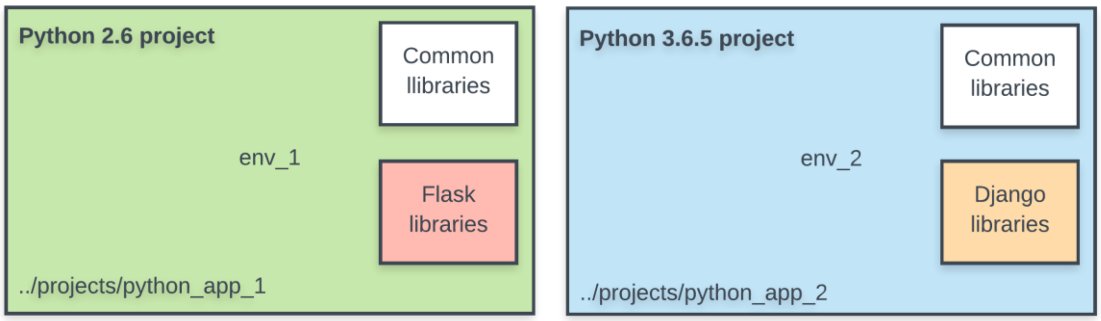


Figure 4. Two separate Python projects running in virtual environments with different Python versions and included libraries.

Although libraries could be installed globally to the system and imported, this may cause problems if some older projects require an older version of some library. Environment virtualization solves that problem.

Docker

Docker is a platform originally developed for Linux but is also available for other operating systems. It serves people who need to run Linux and Windows based applications with containers. Containers are a way to virtualize applications, however the mechanism works differently compared to virtual machines. Containers run natively on Linux and are more lightweight compared to a normal virtual machine, because it shares the host's kernel and does not require a guest operating system to be installed. [16.] The difference between a container and a virtual machine is illustrated in figure 5.

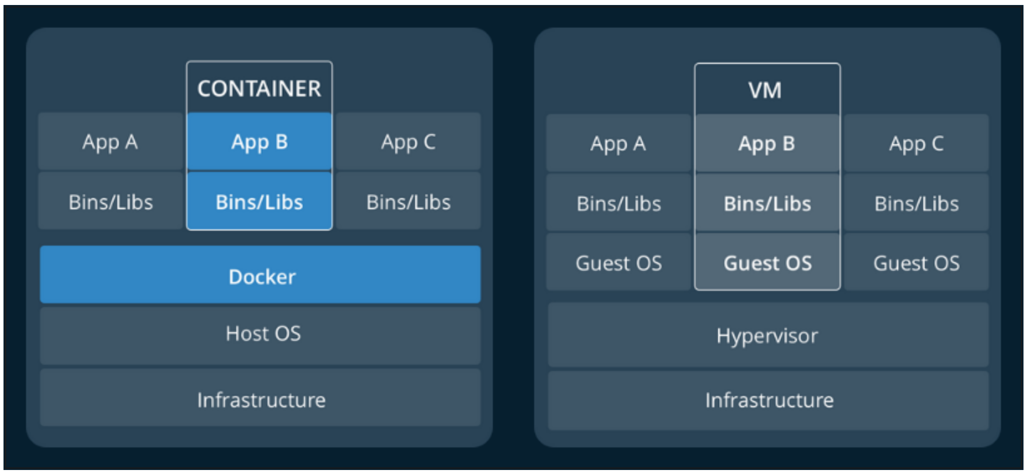


Figure 5. Illustration of the difference between a container and a virtual machine. [16.]

Docker isolates the program from the operating system; it is possible to install only the necessary programs in the container, thus saving resources. Docker uses an image called Dockerfile, which is used to configure the container. [17.]

Other solutions, such as for example Vagrant, configure operating systems completely; Vagrant takes care of installing and configuring applications, as well as setting up the system configuration. Vagrant can thus also be used to configure virtual machines. [18.] This solution was used to create a development environment for the FEL's WordPress service.

This approach is similar and a subset of Docker's functionality: Docker also configures an operating system (the container) using the Dockerfile and the chosen image. Additionally, it takes cares of managing the containers. [19.]

Each member of the development team uses a different operating system; thus, Docker is used to ensure the consistency of the development environments across different environments. The project was created from a Cookiecutter template, which ships with a pre-configured Dockerfile and Django template, which was then modified for custom use. This will be further discussed in chapter 4.

3.3 Project Tools

More programming languages are supporting web development, and this opens new possibilities for the developers. This also complicates things, because there are now more tools available than ever before and choosing the tools for the project might be harder to decide.

This section goes through the tools (stack), which were chosen for the FEL's new project. The tools for this project were chosen around the idea of having a Representational state transfer (RESTful) API server with separate client-side application.

3.3.1 Django Framework

Python is a cross-platform, general-purpose, object-oriented programming language, increasing popular in sectors of mathematics, biophysics, machine learning and web development. Python programs are usually interpreted instead of compiled to platform-specific binaries which allows a fast development cycle.

Django is a high-level open-source Python Web framework that encourages rapid development and clean, pragmatic design. [20.] Django was open-sourced in summer 2005. Project was supported by other open-source projects such as Apache, Python and PostgreSQL. [21.]

Django is not a content-management-system (CMS) like WordPress or Drupal, which are both big CMSs written in PHP. Quoting Django's own website:

> For example, it doesn't make much sense to compare Django to something like Drupal, because Django is something you use to create things like Drupal. [22.]

Although components of Django share CMS-like functionality, like the admin panel module, it does not qualify to be a CMS.

Working with Django reminds of working with PHP. Django does not require any front-end frameworks for creating dynamic web-services because Django uses own templating language called Django template language (DTL), which is mixed with HTML. Django can be combined with other frameworks but is not necessary. [20.] This basic Django way to create web-services uses Model Template View (MTV) paradigm, which is essentially Model View Controller (MVC) with different terms [22.] as shown in figure 6.

Typical MVC	Django MTV (file)
Model	Model (models.py)
View	Template (template.html)
Controller	View (views.py)

Figure 6. Compares typical MVC and Django's representation MVC.

In the MVC paradigm, the model represents the data, the view shows it and the controller controls what is done to the data. The user sees the view and uses the controller to modify the model, which is then updated in the view again. The model acts as the data storage. This is illustrated in figure 7.

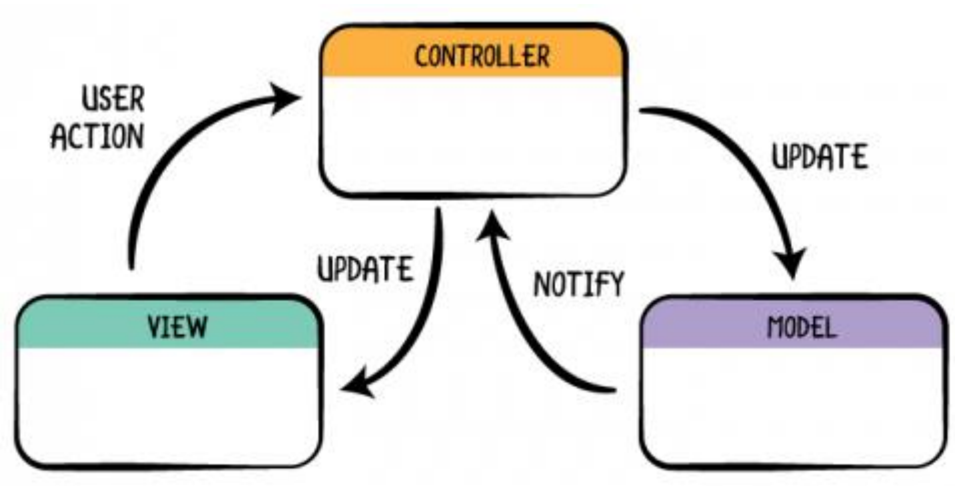


Figure 7. Illustration of MVC concept. The client sees the view and interacts with the model. [24.]

In a regular Django application, requests are directed by the urls.py config file. Django parses the Uniform Resource Locator (URL) and redirects the data to different modules. These modules include the view (controller), data models and templates. When creating a new Django module (app) it creates a folder with app’s name and the file structure. Template files are not created because the view.py can be used to render HTML as well. Making the app easier to manage, it is good practise to create separate templates folders and include the HTML templates in to the views.py.

Django REST Framework

By default, Django is not ideal for building RESTful APIs, because it does not ship with a straightforward way to build APIs. Instead, there is a collaboratively funded side project called Django REST framework. Django REST framework is a powerful and flexible toolkit for building Web APIs. Although free to use, it is encouraged to fund the project when creating commercial projects. [25.]

By using Django REST framework, the MVC logic is moved to the front-end side. This framework transforms Django's role in the stack to that of a server; it does not display any content but is only responsible for handling requests from the client for providing and modifying data. RESTful API is illustrated in figure 8.

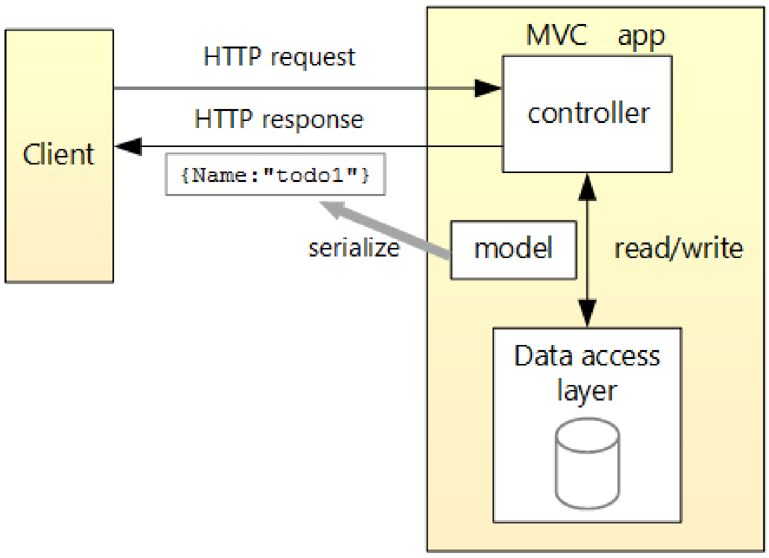


Figure 8. Shows data flow of a RESTful API. [26.]

The view (the controller) no longer renders the data (the model) into the view, instead the data is serialized and retrieved by the view itself. The view takes care of rendering this serialized data (the model), it becomes clear that the view is now responsible for a small part of the controller's job.

When choosing a framework to work with, Django was chosen for its easy setup and easy approach. Django encourages developers to separate different concerns of the system in to small parts, and thus generalizes everything in such a way that the user only needs to create models with the desired structure and functionality.

Another option on Python’s side was Flask, which allows the user much more freedom to structure the project in their own way. Django expects its users to follow its philosophy closely. For example, Flask allows the whole project to run from a single file, whereas Django expects the user to follow the project folder structure.

The development team had previous experience with Python. Django felt much more approachable than Flask: there was no need to reinvent the wheel for many common functionalities that were required for this project. PostgreSQL database was chosen alongside with the Django as data storage.

3.3.2 PostgreSQL Database

PostgreSQL is an open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. PostgreSQL originates back to 1986 as part of the POSTGRES project at the University of California at Berkley and has been under active development since. [27.]

PostgreSQL has earned a reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open source community behind the software to consistently deliver performant and innovative solutions. PostgreSQL supports all major operating systems. [27.]

Django uses SQLite as default database engine. This is not recommended even in development if creating a real product. It is good option for quick development and prototyping, because it is less strict compared to PostgreSQL.

PostgreSQL is more powerful and faster for larger systems than SQLite or even MySQL. While SQLite is good for prototyping and small local application, so using it was not even an option. PostgreSQL was recommended in Django's documentation and by the authors of the Two Scoops of Django. [28, 13 – 15.]

3.3.3 React Library

React (also called as ReactJS) is an open-source JavaScript library for building dynamic user interfaces. It is developed by Facebook, Instagram and community. React is considered as the view from MVC model. [29.]

Developing with React drives to break the user interface to small components, which might include one or more components inside them. [30.] React components are written with JavaScript XML (JSX) pre-processor to format the user interface. Syntax of React is a combination of JavaScript and HTML-like code. This code is then compiled to JavaScript. [30; 31.]

The decision of the platform for the client-side application was narrowed down to two options. The options were Angular and React. React felt more interesting option for the developers as well as something new to learn. Bootstrap CSS framework was chosen to help in creating clean looking user interface for the client-side application.

3.3.4 Bootstrap CSS

Bootstrap is a CSS framework developed by Twitter, it has been a popular CSS framework for years. First version released 2011 and got latest major update to version 4 in 18 January 2018. [32; 33.]

CSS framework is a compilation of pre-set rules for HTML styling. Commonly CSS frameworks come packed with a grid system, which allows easily to build webpage out of boxes that will scale on desktop and mobile devices automatically. Larger frameworks include certain styling so keeping up a consistent look for website is made effortless.

Choosing Bootstrap for CSS framework was straight forward. Bootstrap has well written documentation, dedicated support by community and it can be customized. Other options were to keep using Bulma or creating everything from scratch using the CSS Grid, but Bootstrap gives more head start with its ready blocks and is fast for creating prototypes.

3.3.5 Git Version Control

Version control is a system that keeps track of all the changes in file or set of files and allows client to return in time to specific version if needed. This can be achieved manually locally by creating copy of the file to another folder each time changes are made. Local version's downside is that it is highly error prone. It is easy to make human error and copy something to wrong folder thus losing files. A better way to do this is to use Centralized Version Control (VCS), which stores files in one centralized server or even better and more reliable way is to use advanced system like Git. [34.]

Git is a Distributed Version Control System (DVCS), meaning that every client has full copy of the repository including its full history, unlike in VCS where data history is stored only in the server. Git makes it possible for every project participant to have the same files at dispose allowing them to make changes and committing changes to repository where others can reach it. [34.]

Code storage is an important part of every project. The development team continued using Git, because it was already used in previous projects.

4 Project

This chapter investigates the project by first looking into the proposition of a new replacing system. Also, it looks how the development environment is set up and demonstrates Django REST framework with an example from the project.

4.1 Proposition

To build a more robust service, it was important to take all the problems concerning the service and which have already been discussed in this study into account. This section discusses what kind of structure the new proposed system should have. The author of this thesis proposed an idea of splitting the current website into two separate services to make it easier to manage and develop.

The first service would be a news site, where all the news and results could be found. This side of the FEL's web services would no longer have user authentication, because it would be only used to comment the news articles. Commenting on the articles has been scarce and it was mostly done through social media such as Twitter and Facebook, so there was no need for it anymore. This side will have its own database for the news articles and only fetches other data such as match results from the FEL API.

The other side would be strictly for the players. This player service would have authentication service and possibility to buy a seasonal license to the service. Because this is a custom service it is easier to build from scratch rather than using something as of a base. Unlike the news site, this part can fetch and update the database. The whole service structure is illustrated in figure 9.

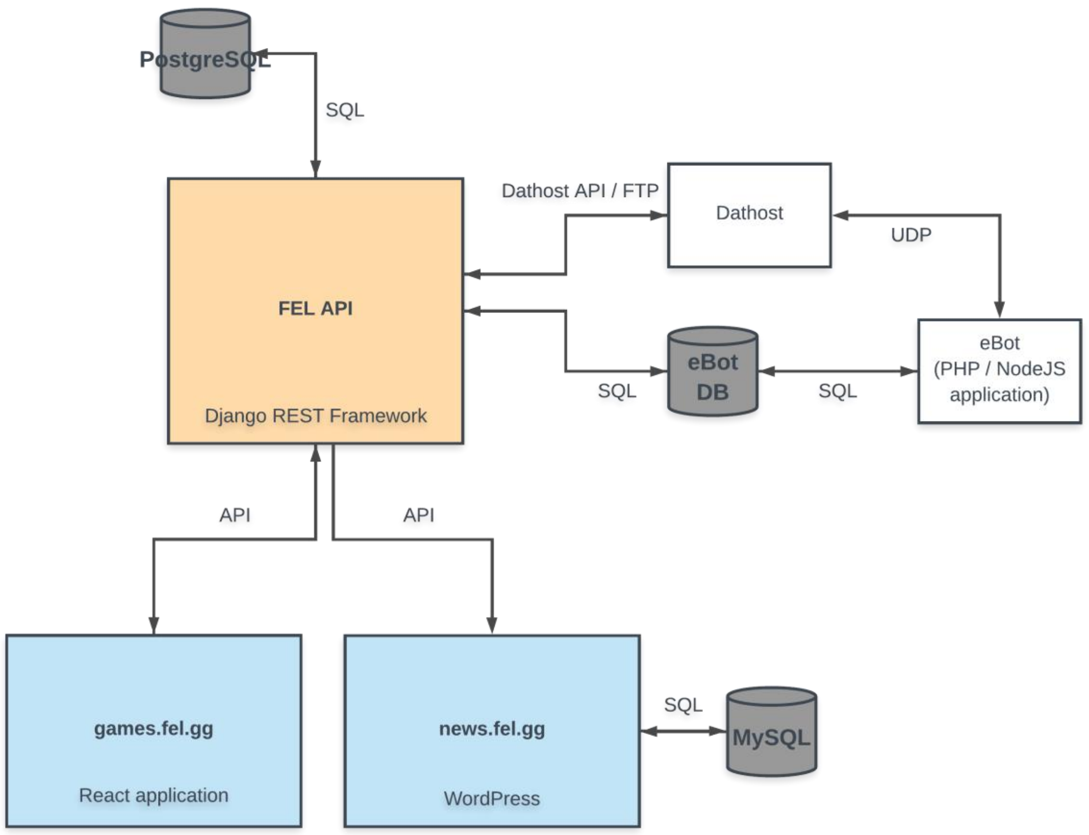


Figure 9. New proposed system structure is more complicated but allows more flexibility.

As can be seen above, the new FEL service structure consists of two separate platforms. Although more complex compared to the old system, it is hoped to be more maintainable, because possible problems can be pointed out much more accurately.

With new system developers have planned that some parts of the FEL API would be later opened for public, so that other third-parties can have access to the match logs and use that data for their own use. Separation between back-end and front-end was decided so they can each run stand-alone and developed that way.

4.2 Project Goal

The goal of this project was to create a more flexible and scalable system with RESTful API, which can be used to create tournaments dynamically and a client-side web application which connects to this API and can operate on desktop web browsers and mobile devices. In this report the goal is to demonstrate a part of the system and investigate how is it to work with these frameworks.

4.3 Setting Up Development Environments

Setting up a development environment can be achieved several ways. Methods demonstrated in this section are the way this project got its development environment set up. Although some differences were done to the configurations, which this thesis will not cover. Setting up a similar development environment and requires the following programs pre-installed:

- Docker-Compose
- NodeJS 8.11.1 with npm 5.6.0 or newer
- Python 3.6.5 with pip

Other installations are included in the coming up sections.

4.3.1 Front-end Environment

Starting to work with React has been done effortless with create-react-app command line utility. [35.] The installation has three steps and the first step is to create a new app by running following command:

```
$ npx create-react-app my-app
```

Instead of installing the create-react-app command line utility globally with npm, npx installs the utility locally and once it is done using it, the utility is then removed. The installation takes a moment, but after it is finished, a folder with given app name has been created. The development server can be activated by moving inside the app folder and running the following command:

```
$ yarn start
```

The command automatically opens a browser window to http://localhost:3000 where React project template is shown. This is already a working environment, but to complete the installation, Bootstrap and Reactstrap need to be installed.

Reactstrap includes the stateless React components for Bootstrap. This makes using Bootstrap with React project much easier. Reactstrap does not include Bootstrap CSS and that is why it must be installed separately. Both can be installed with npm like shown on Listing 1. below. [36.]

```
$ npm install --save bootstrap@4.0.0
$ npm install --save reactstrap react@^16.0.0 react-dom@^16.0.0
```

Listing 1. Commands for installing Bootstrap to a React project.

Using yarn install after installing other packages might solve problems caused by possible packet conflicts. Finally adding an import for the Bootstrap CSS into the src/index.js file finalizes the installation.

```
import 'bootstrap/dist/css/bootstrap.css';
```

4.3.2 Back-end Environment

While not being as effortless as setting up a React environment, Django has a few options for setting up its own development environments. One option is to setup a basic project with instructions from Django’s website. However, setting up the PostgreSQL database is a bit more involved, thus making the development environment reproducibility harder. An easier approach to creating the development environment, is to use a templating tool. React’s create-react-app’s equivalent in the Python world, is called Cookiecutter: a command-line utility tool for project templates. [37.] Using this template and modifying it for the need, takes less effort than doing everything from scratch.

Getting started is as easy as following the tutorial on Cookiecutter’s GitHub page. [37.] The first step is to install Cookiecutter using pip with the following command.

```
$ pip install "cookiecutter>=1.4.0"
```

Once Cookiecutter is installed, it can be used to install any template available. In this case, the template used is “cookiecutter-django”, which creates a Django project with pre-installed Django REST framework. When using Cookiecutter, some configuration can be done during the installation process. Some of the most important configurations are demonstrated below in Listing 2.

```
$ cookiecutter http://github.com/pydanny/cookiecutter-django
project_name [My Awesome Project]: demo_project
...
timezone [UTC]: Europe/Helsinki
windows [n]: y
...
use_docker [n]: y
Select postgresql_version:
1 - 10.3
2 - 10.2
3 - 10.1
4 - 9.6
5 - 9.5
6 - 9.4
7 - 9.3
Choose from 1, 2, 3, 4, 5, 6, 7 [1]: 1
...
[SUCCESS]: Project initialized, keep up the good work!
```

Listing 2. Cookiecutter template installation, demonstrates template configuration.

This saves time on the PostgreSQL configuration. The options are chosen in such a way, that the project can be developed on Windows operating systems. In case Docker is required, this can also be chosen during the configuration process, as shown in listing 2.

If the operating system used for developing is Windows, the “psycopg2” PostgreSQL adapter for Python must be added to the requirements/base.txt. After that, the tool stack can be built with the first command and started with the second, as shown below. [38.]

```
$ docker-compose -f local.yml build
$ docker-compose -f local.yml up
```

The second command takes care of starting up the PostgreSQL and the Django instance inside the Docker container. Running any shell commands that need to be run inside the container, can be achieved by adding “`docker-compose -f local.yml run --rm`“ in front of the command. Creating a super user is done this way and is demonstrated below [38.]

```
$ docker-compose -f local.yml run --rm django python manage.py cre-
atesuperuser
```

This prompts for a username and password, for the main admin account for the Django project. This concludes the installation process of the back-end environment.

4.4 Project Example

This section investigates the project and looks some of the code as an example of Django models are made, serialized and how to then access that data from a demo React application. In the example goal is to create a team Django app. It was chosen as an example, because it has more complexity and provides a good example of the possibilities that Django provides. Next a serializer is created for this model and finally print the data from the API to a React prototype application.

4.4.1 Django Apps

Django's functionality is based on small apps and their interaction with each other's. Each app has own models and URL configurations. This section discusses how the team app was done.

The first thing to do is to create a new app. Creating a new app to the project can be done, inside the project folder, where manage.py called and given parameters "startapp" and name of the app as shown below.

```
$ docker-compose -f local.yml run --rm django python manage.py startapp
team
```

This command creates a new app called "team" with a folder structure, which includes all the necessary files, such as models.py and views.py. Although this step does not add any functionality yet, this helps the developer by cutting down manual folder creation and keeps the folder structure unified.

After creation, the app must be added to the project config. Inside the base.py config, there is a list of installed Django apps, which are in the project. Adding apps to the list activates the for the use. (See Listing 3. below)

```
INSTALLED_APPS = (
    ...
    'authentication',
    'player',
    'license',
    'game',
    'team',
)
```

Listing 3. Listing of installed apps in the base.py config file.

Creating a model on the basic level only requires fields that are desired. Django provides several options for the model creation and customization, thus making model creation easy. Looking at the Team class for example.

The Team class has defined fields such as id, name, game and members. Each field has properties set here as well, such as maximum length and if the value on the field must be unique. (See Listing 4. below)

```
import uuid
...
from invite.models import Invitation


class Team(TimeStampedModel):
    id = models.UUIDField(primary_key=True,
                          default=uuid.uuid4, editable=False)
    game = models.ForeignKey(Game, on_delete=models.CASCADE)
    name = models.CharField(_('name'), max_length=200, unique=True,
        help_text=_('Required. 500 characters or fewer. Unique.'),
        error_messages={'unique': _("This name already exists."),},
    )

    members = models.ManyToManyField(Player, through='Membership',
                                     through_fields=('team', 'player'),
                                     related_name='teams')
    invitations = GenericRelation(Invitation,
                                  content_type_field='target_content_type',
                                  object_id_field='target_id')
    objects = TeamManager()

    def join_player(self, player):
        """ Adds a membership for a player """
        from .utils import can_player_join_team, create_membership
        if can_player_join_team(team=self, player=player):
            create_membership(self, player)
            return True
        else:
            return False
```

Listing 4. Team class inside team/models.py.

A special note is that the Team class takes in an abstract base class TimeStamped-Model. TimeStampedModel adds self-updating “created” and “modified” fields to the model. This is part of a third-party help utility project called model_utils, which is a collection of Django model mixings and utilities. [39.]

Unlike in some other frameworks, in Django not only the data type is defined, but things such as relation to other data models and restrictions of the data. This means that Django models contain some of the validation process already inside the model declaration. Validators can be added as properties e.g. in the code field from the team class, which takes in a MinLengthValidator(2), which raises an error if the given field is less than two. (See Listing 5. below)

```
code = models.CharField(max_length=5, null=False,
                            blank=False, unique=True,
                            validators=[MinLengthValidator(2)])
```

Listing 5.   Field “code” from team/models.py with a validator property.

It is recommended to override the __str__ function in every model, which makes identifying the objects in the admin panel easier. Instead of “<class name> object”, the actual name property of the object will be displayed. (See Listing 6. below)

```
def __str__(self):
    return self.name
```

Listing 6.   String override method returns the name field from the object.

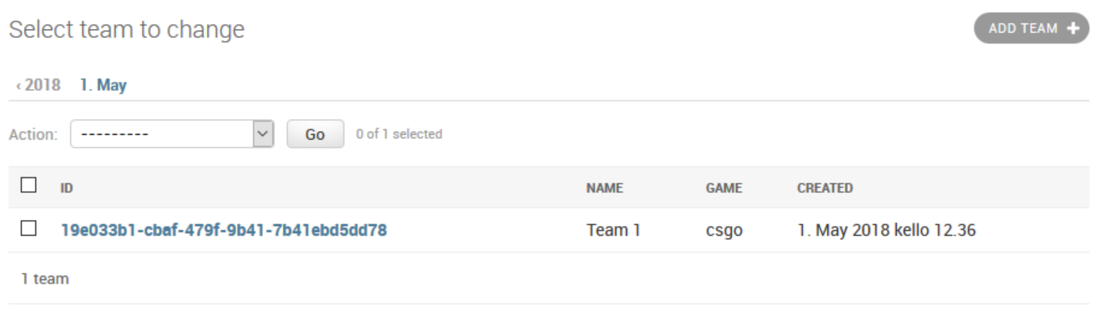
Inside the team/models.py there is a second class called Membership. The Team is the main class and uses Membership class as bridge to the Player model. (See Listing 7. below)

```
class Membership(TimeStampedModel):
    team = models.ForeignKey('Team', on_delete=models.CASCADE)
    player = models.ForeignKey(Player, on_delete=models.CASCADE)
    TYPE_DEFAULT = 0
    TYPE_ADMIN = 1
    TYPE_CHOICES = (
        (TYPE_DEFAULT, 'default'),
        (TYPE_ADMIN, 'admin'),
    )
    type = models.SmallIntegerField(choices=TYPE_CHOICES, null=False,
                                    default=TYPE_DEFAULT)
    unique_together = ('player', 'team')
    objects = MembershipManager()
```

Listing 7.   Membership class inside team/models.py.

This membership class links a player and a team together through a membership. A player has position in the team, and this data is stored in the membership. The Membership class operates only under the Team class and cannot be created separately.

Created models can be then registered to the admin panel. This makes it possible to modify the data and shows the history of it. A team created from the Team model inside the admin panel is shown in figure 10.


Select team to change

‹ 2018 1. May

Action: --------- Go 0 of 1 selected

ID	NAME	GAME	CREATED
19e033b1-cbaf-479f-9b41-7b41ebd5dd78	Team 1	csgo	1. May 2018 kello 12.36

1 team

Figure 10. Team app inside Django admin panel.

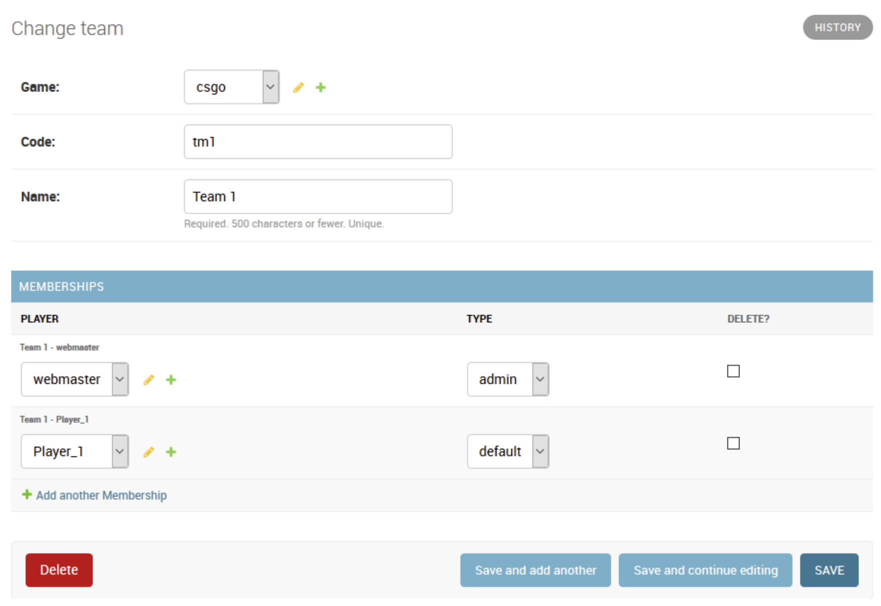
By default, only the object is shown or the overriding __str__ value. But in this case the id, name, game and created are shown because they are configured in the admin settings.

Each Django app has its own admin settings file (admin.py), which contains configurations of the admin view. Registering a model to the admin panel is optional but gives a view of the data for the administrator, thus is recommended. Admin settings of the Team admin can be seen in Listing 8. below.

```
from django.contrib import admin
from .models import Team, Membership


class MembershipInline(admin.TabularInline):
    model = Membership
    extra = 0


@admin.register(Team)
class TeamAdmin(admin.ModelAdmin):
    exclude = []

    date_hierarchy = 'created'
    list_filter = ('game', )
    list_display = ('id', 'name', 'game', 'created', )
    inlines = [MembershipInline]
```

Listing 8. Admin panel configurations of the team/admin.py.

Registration does not need other information except the model that is being registered. Customizations to the admin view are added here, such as what fields are shown and options for the data filtering in the admin panel. The Membership is registered as separate class and added as TabularInline under the Team model. This shows the Membership in admin panel as illustrated in figure 11.



Game:	csgo
Code:	tm1
Name:	Team 1

Required. 500 characters or fewer. Unique.

MEMBERSHIPS

PLAYER	TYPE	DELETE?
Team 1 - webmaster webmaster	admin	☐
Team 1 - Player_1 Player_1	default	☐

Figure 11. Editing a team in Django admin panel.

As shown above the Membership class works under Team class. Because the Game field is a reference to another model admin panel allows to pick data straight from a list or add new one.

Django is flexible with its data models. Data migration allows database to be updated as the development moves on. With migrations developer does not have to worry about resetting the database each time the models change, because the data is updated based on the model changes.

The migration is done in two parts. The first command prepares the migration with possible conflicts and in case there is a new data field it prepares a default value, if that is not defined.

```
$ docker-compose -f local.yml run --rm python manage.py makemigrations
```

The second command executes the migration to the database.

```
$ docker-compose -f local.yml run --rm python manage.py migrate
```

Migration is recommended even after minor changes that have been made to the models. Purpose of this is to ensure that everything continues working correctly.

4.4.2 Django Serializer

A serializer makes it possible to turn complex Django data into JavaScript Object Notation (JSON) format and pass that data to the client side. When data is returned trough serializer, it transforms the data back to complicated Django format, which is then stored to database by the view (controller).

The main urls.py file contains all the routings of the system. This file reads the URL and redirects the data to corresponding place. Django uses a router to simplify API routing, which has the URL prefix and the corresponding viewset. Router generates automatically URL patterns and this list URLs form the router is then passed for the baseURL api/v1/. This is demonstrated in Listing 9. below.

```
from rest_framework.routers import DefaultRouter
...
from team.views import TeamViewSet
from game.views import GameViewSet

router = DefaultRouter()

...
router.register(r'player', PlayerViewSet)
router.register(r'team', TeamViewSet)
router.register(r'game', GameViewSet)

urlpatterns = [
    ...
    url(r'^api/v1/', include(router.urls)),
    url(r'^$', schema_view),
]
```

Listing 9. Main URLs configurations file.

Each viewset has own serializer inside the app folder. At the same time the serializer filters what data is passed, it also acts as a validator. This can be seen on the fields that have “required=True”, which raises an error if these fields are missing when passing data through the serializer. (See Listing 10. next page)

```
from rest_framework import serializers
from .models import Team

class CreateTeamSerializer(serializers.ModelSerializer):
    name = serializers.CharField(required=True)
    code = serializers.CharField(required=True)
    game_id = serializers.IntegerField(required=True)

    class Meta:
        model = Team
        fields = ('name', 'code', 'game_id',)

class TeamSerializer(serializers.ModelSerializer):
    class Meta:
        model = Team
        fields = ('id', 'name')
```

Listing 10. Model based serializer modeling team class.

Inside TeamSerializer there is a class called Meta, which has two properties. First is the model that is used for this serializer and another is a list of the fields, which are in this case excluded from the dataset. Include could be used instead, but in this case, there are less to exclude.

The view controls the data after it has passed the serializer. A view usually includes the basic functions create() and update(). Something to take a note of is the Retrieve-ModelMixin, which is passed inside the TeamViewSet. It is used for data fetching when doing using GET method and automates the process. [Appendix 1.] API's endpoints could be tested with tool called Swagger, which allows to perform API calls from a simple UI on the browser. [Appendix 2.]

4.4.3 React Connection to API

Since the development did not get far enough and concentrated heavily to the RESTful API, the client-side application was only prototyped, and since the final layout was still under design nothing permanent was created. In this demonstration goal is to list all the teams from the database.

It is recommended that the folder structure contains a components folder, which has folders for different URLs. In this case a new component is created to ./components/team/TeamList.js. The purpose of this component is to build the list of teams and render it to the view. This TeamList component is imported to the app.js and placed inside a Bootstrap element col. (See Listing 11 on the next page)

```
import React, { Component } from 'react';
import {Container, Row, Col} from 'reactstrap';
import TeamList from './components/team/TeamList';

export default class App extends React.Component {
  ...
  render() {
    return (
      <div>
        ...
        <Container>
          <Row>
            <Col>
              <TeamList />
            </Col>
          </Row>
        </Container>
      </div>
    );
  }
}
```

Listing 11. TeamList imported to the App.js.

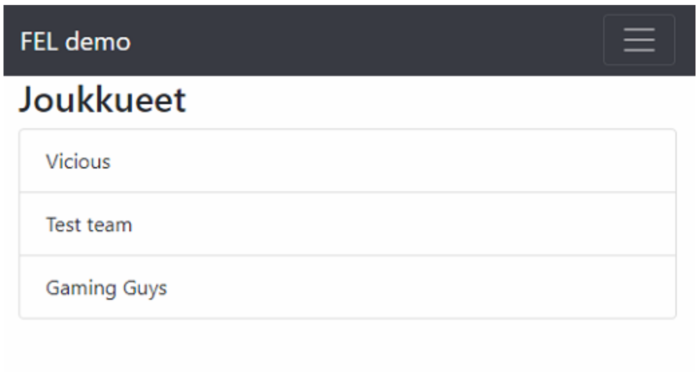
The TeamList component consists of couple of methods. First is the constructor, which contains the state of the component. The other methods include renderTeams(), render() and the componentDidMount(). The whole component can be seen in Appendix 3.

Once the component is mounted it instantly invokes the componentDidMount() lifecycle function. In this case the function uses fetch() method to fetch data from the URL. A promise is given for the fetch method, which will complete after the fetch method has finished. After the promise is fulfilled the data is added to the state. (See Listing 11.)

```
componentDidMount() { // Fetches the data, once page has loaded
    fetch('<API_LINK>').then(response => {
        return response.json();
    }).then(data => {
        let teams = data.results;
        this.setState(teams);
    })
}
renderTeams(team) { // Renders single item from the dataset
    return (
        <ListGroupItem key={team.id}>{team.name}</ListGroupItem>
    )
}
render() { // Renders the whole component using Bootstrap’s ListGroup
    return (
        <div>
           <ListGroup>{this.state.teams.map(this.renderTeams)}</ListGroup>
        </div>
    )
}
```

Listing 12. TeamList component’s data fetching and data rendering methods.

Above shown render functions work together to build a list component. The render() method calls the list of teams and loops them with the map() method. The renderTeams() method returns a ListGroupItem with the data to the ListGroup and once the loop ends, the whole ListGroup is returned to the App.js. The main component displays the list of teams as illustrated in figure 12 below.


FEL demo

Joukkueet

Vicious

Test team

Gaming Guys

Figure 12. A mobile view of the listing, which lists the teams from the API.

5 Experiences and Thoughts

This section investigates the decision of splitting the current website into two separate services and developing a RESTful API with Django REST framework. It also discusses the success of the project and what could have been done differently.

For the author of this thesis it was the first time working in a bigger project with Python and getting to know React. One member in the development team had previous experience working with Django and thus was mostly in charge of the back-end coding. Being part of the decision process, which might have long term effects was interesting and below are some experiences and thoughts.

5.1 Problems and Considerations

The project itself did not get far enough for the possible conflicts to appear or being able to say if there are some structural problems in the system. For this reason, the problems mainly were about time and project management.

The development team was small and could only dedicate a limited amount of time to this project every week. Although the project did not have a scheduled deadline, the development team would have desired faster results. This added to the fact that there was a lot of new to learn, which slowed the progress more.

To outline a possible problem, the development team realised that the project ended up being much bigger than initially thought, because the idea was to create a service better in every way possible; last-minute ideas started to flow in fast and made the project even bigger. Keeping the focus on the purpose of creating a platform with at least the same functionality as the old system, was getting out of hand. As a solution for this, a backlog was created, where the development team could post their ideas. This was an important thing which could have been avoided in the first place by having a solid project plan.

The above-mentioned lack of solid project plan mirrored into the project management as well. Planning the back-end to such an extent, that the front-end was designed would have made the back-end development much easier. Some branches were added to version control but keeping track of the actual changes was hard, since the plan for the back-end was only defined in vague terms. This made it difficult to say which parts were still missing and which still needed improvement. As for the future, the development team decided to start writing all the features and progress to a project management platform called Taiga. [40.]

5.2 Final Thoughts

Our team took a leap and tried something that was new for almost everyone, and from the learning perspective it was a good thing. The project enforced the fact that the same result can be achieved in many different ways.

When compared to Django, some previously created RESTful APIs with MEAN were more work from a coding perspective, since with MEAN everything had to be done from the ground up. By default, Django ships with a development server and an admin panel, which gives a good starting point for a project. Setting up a RESTful API was effortless, and the created Django model transformed well into an API. Being able to serialize and use inheritance for the models cut down the amount of code.

Working in an environment which the developer knows, makes working enjoyable. Looking into new frameworks and languages gives a new perspective and is something that should be encouraged. This project was a learning experience for the whole development team.

Based on this project, I would recommend Django as a base for a web service. Django encourages the developer to do things in a certain way but does not stunt Python's flexibility: this makes easy things simple and hard things possible. It is flexible enough to create regular web applications as well as RESTful APIs.

The next step would be to learn more about React and take the project forward on the client-side. Time will tell how the React application will work out. The advantage of making RESTful APIs is that the client-side application can be switched to any other framework if necessary, which gives much more room to change things in the future.

6 Conclusion

The purpose of this study was to investigate full-stack development and the structure of the web-service provided by FEL. Also, this thesis focuses on the decision to split the current website into two separate services.

Full-stack development usually refers to the developers who understand every layer of the full-stack. This includes not only front- and back-end development environments, but also project management, system infrastructure and databases. In summary, a full-stack developer might have a wide knowledge of all the layers, but not necessarily a deeper understanding of a single layer.

FEL approved of the proposition of splitting the service in half. Also, the developer team concluded that splitting up the old service was a good decision. This project began after a decision about the tools was made.

Learning the technologies was time consuming and slowed down the development. Also, the development team was able to dedicate a limited time for the project which slowed down the progress.

Although the project did not get completed during this thesis study, the progress looks promising and further development continues. In the future, a good addition would be to leave out the eBot completely and replace it with a custom solution.

References

1 Counter-Strike: Global Offensive - Steam Store Page. 2018. Valve. <http://store.steampowered.com/app/730/CounterStrike_Global_Offensive/>. Accessed 10 April 2018.

2 Most Watched Games on Twitch & YouTube Gaming - March 2018. 2018. Newzoo. Web Document. <https://newzoo.com/insights/rankings/top-games-twitch-youtube/>. Accessed 19 April 2018.

3 Pelaajan opas: Hearthstone. 2018. Finnish Esports League. Web Document. <https://fel.gg/fel-pelit-hs/>. Accessed 21 April 2018.

4 WordPress About. 2018. WordPress. Web Document. <https://wordpress.org/about/>. Accessed 4 April 2018.

5 CMS eli Content Management System. 2018. Omni Partners. Web Document. <https://omnipartners.fi/sanakirja/cms-eli-content-management-system/>. Accessed 20 April 2018.

6 WordPress Plugins. 2018. WordPress. Web Document. <https://wordpress.org/plugins/>. Accessed 4 April 2018.

7 WordPress Themes. 2018. WordPress. Web Document. <https://wordpress.org/themes/>. Accessed 4 April 2018.

8 Freeman, Adam. 2014. Pro AngularJS (Expert's Voice in Web Development) 1st ed. Edition. Berkely. Apress.

9 Seshadri, Shyam & Green, Brad. 2014. AngularJS Up & Running. Sebastopol. O’Reilly Media, Inc.

10 Bulma. 2018. Bulma. Web Document. <https://bulma.io/>. Accessed 11 April 2018.

11 What does the term "full-stack programmer" mean? 2016. Begleiter, Josh. Web Document. <https://www.quora.com/What-does-the-term-full-stack-programmer-mean-What-are-the-defining-traits-of-a-full-stack-programmer>. Accessed 16 April 2018.

12 What Is A Full Stack Web Developer, And Why Should I Become One? 2017. Udacity. Web Document. <https://medium.com/udacity/what-is-a-full-stack-web-developer-and-why-should-i-become-one-6e93d0c774b6>. Accessed 11 April 2018.

13 LAMP (Linux, Apache, MySQL, PHP). 2008. Rouse, Margaret. Web Document. <https://whatis.techtarget.com/definition/LAMP-Linux-Apache-MySQL-PHP>. Accessed 16 April 2018.

14 The MEAN Stack: MongoDB, ExpressJS, AngularJS and Node.js. 2013. Karpov, Valeri. Web Document. <https://www.mongodb.com/blog/post/the-mean-stack-mongodb-expressjs-angularjs-and>. Accessed 16 April 2018.

15 6 Web Development Stacks to Try in 2017. 2017. Lozinsky, Yuriy. Web Document. <https://webinerds.com/6-web-development-stacks-try-2017/>. Accessed 16 April 2018.

16 Docker Get Started. 2018. Docker. Web Document. <https://docs.docker.com/get-started/>. Accessed 25 April 2018.

17 What is a container. 2018. Docker. Web Document. <https://www.docker.com/what-container>. Accessed 12 May 2018.

18 Introduction to Vagrant. 2018. Vagrant. Web Document. <https://www.vagrantup.com/intro/index.html>. Accessed 12 May 2018.

19 Vagrant vs. Docker. 2018. Vagrant. Web Document. <https://www.vagrantup.com/intro/vs/docker.html>. Accessed 12 May 2018.

20 Meet Django. 2018. Django. Web Document. <https://www.djangoproject.com/>. Accessed 15 April 2018.

21 Django Documentation. Version 2.0. 2018. Web Document. <https://docs.djangoproject.com/en/2.0/misc/design-philosophies/>. Accessed 15 April 2018.

22 Django Documentation – FAQ: General. 2018. Django. Web Document. <https://docs.djangoproject.com/en/2.0/faq/general/>. Accessed 15 April 2018.

23 Django Documentation – Templates. 2018. Django. Web Document. <https://docs.djangoproject.com/en/2.0/topics/templates/>. Accessed 5 May 2018.

24 Model-View-Controller (MVC) in iOS: A Modern Approach. 2016. Peres, Rui. Web Document. <https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach>. Accessed 25 April 2018.

25 Django REST framework. 2018. Web Document. <http://www.django-rest-framework.org/>. Accessed 20 April 2018.

26 Create a Web API with ASP.NET Core and Visual Studio for Windows. 2018. Microsoft. Web Document. <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.1>. Accessed 24 April 2018.

27 PostgreSQL - About. 2018. The PostgreSQL Global Development Group. Web Document. <https://www.postgresql.org/about/>. Accessed 24 April 2018.

28 Greenfeld, Daniel Roy & Greenfeld, Audrey Roy. 2017. Two Scoops of Django 1.11: Best Practices for the Django Web Framework. Los Angeles: Two Scoops Press.

29 Facebook's React JavaScript User Interfaces Library Receives Mixed Reviews. 2018. Hemel, Zef. Web Document. <https://www.infoq.com/news/2013/06/face-book-react>. Accessed 24 April 2018.

30 What is React? 2017. Lerner, Ari. Web Document. <https://www.fullstackre-act.com/30-days-of-react/day-1/>. Accessed 25 April 2018.

31 Tutorial: JSX. 2018. Shen, Paul. Web Document. <http://buildwithreact.com/tuto-rial/jsx>. Accessed 5 May 2018.

32 Bootstrap - About. 2018. Bootstrap. Web Document. <https://getboot-strap.com/docs/4.0/about/overview/>. Accessed 11 April 2018.

33 Bootstrap 4. 2018. Bootstrap. Web Document. <http://blog.getboot-strap.com/2018/01/18/bootstrap-4/>. Accessed 11 April 2018.

34 Chacon, Scott and Straub, Ben. 2014. Pro Git Second Edition. Web Document. Git. <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>. Ac-cessed 11 April 2018.

35 Create React App GitHub. 2018. Facebook. Web Document. <https://github.com/facebook/create-react-app>. Accessed 4 April 2018.

36 reactstrap GitHub. 2018. reactstrap. Web Document. <https://github.com/re-actstrap/reactstrap>. Accessed 4 February 2018.

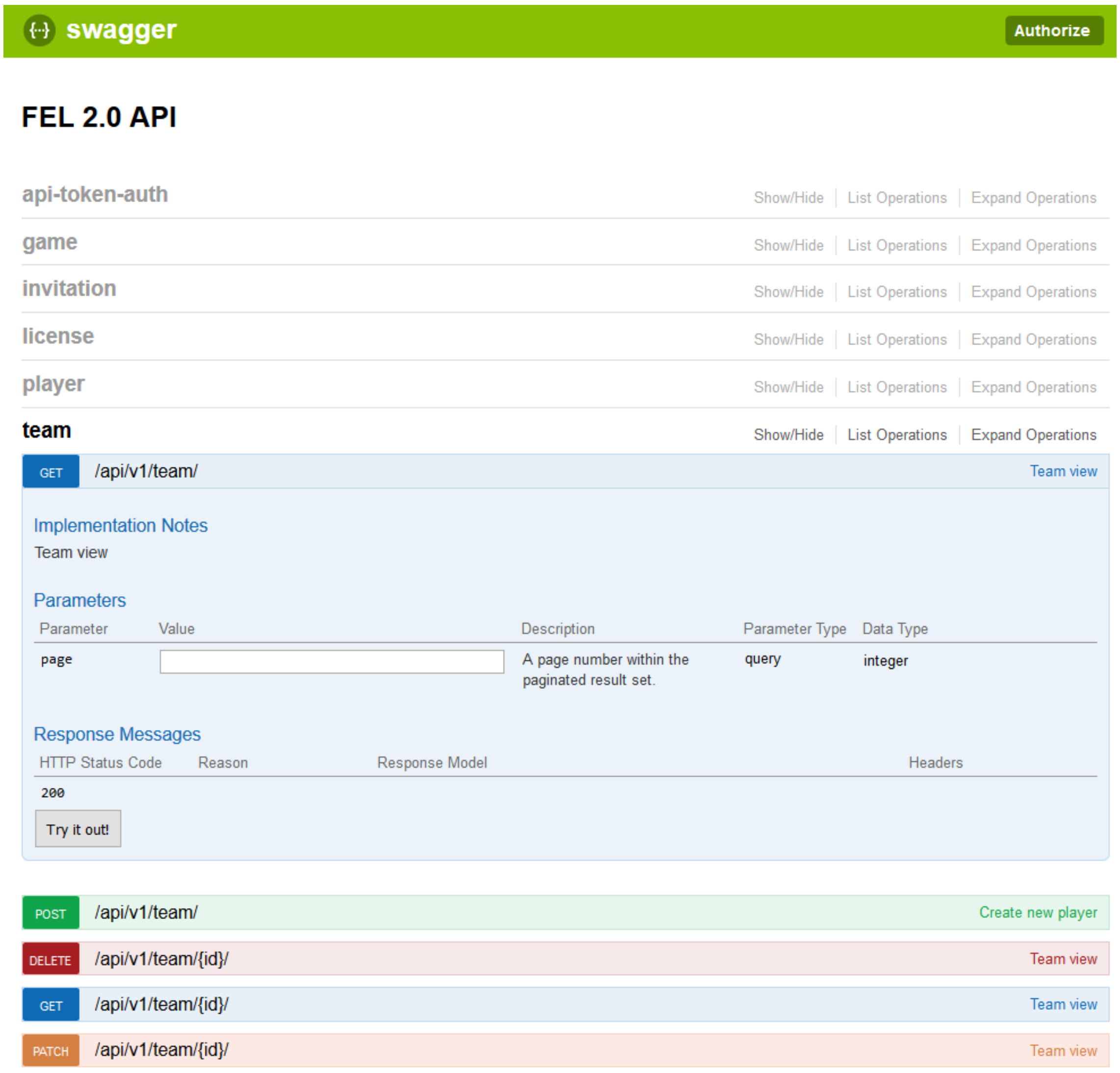
37 Cookiecutter GitHub. 2018. audreyr. Web Document. <https://github.com/audreyr/cookiecutter>. Accessed 2 February 2018.

38 Getting Up and Running Locally with Docker. 2018. Cookiecutter Django. Web Document. <http://cookiecutter-django.readthedocs.io/en/latest/developing-lo-cally-docker.html#optionally-designate-your-docker-development-server-ip>. Ac-cessed 3 May 2018.

39 django-models-utils GitHub. 2018. Jazzband. Web Document. <https://github.com/jazzband/django-model-utils>. Accessed 3 May 2018.

40 Taiga - Love Your Project. 2018. Taiga. Web Document. <https://taiga.io/>. Ac-cessed 3 May 2018.

```
...
class TeamViewSet(viewsets.ModelViewSet,
                  mixins.CreateModelMixin,
                  mixins.RetrieveModelMixin,
                  mixins.UpdateModelMixin,
                  viewsets.GenericViewSet, ):
    """ Team view """

    queryset = Team.objects.all()
    permission_classes = (IsAdminOrIsSafe,)

    def get_serializer_class(self):
        if self.action == 'create':
            return CreateTeamSerializer
        return TeamSerializer

    def create(self, request, *args, **kwargs):
        """ Create new player """
        self.permission_classes = (IsAuthenticated,)

        serializer = CreateTeamSerializer(data=request.data,
                                          many=False)
        if not serializer.is_valid():
            return Response(status=status.HTTP_406_NOT_ACCEPTABLE,
                            data=serializer.errors)

        data = serializer.data

        game = get_object_or_404(Game, id=data['game_id'])

        if request.user.teams.filter(game=game):
            return Response(status=status.HTTP_406_NOT_ACCEPTABLE,
                            data='already in a team')

        team = Team.objects.create_team(name=data['name'],
                                        code=data['code'],
                                        game=game)
        team.save()
        team.join_player(request.user)
        return Response(status=status.HTTP_201_CREATED, data=data)

    @detail_route(methods=['get'], )
    def invitations(self, request, pk=None):
        """ Get player invitations """
        serializer_context = {
            'request': request,
        }
        player = self.get_object()
        invitations = player.invitations.all()
        serializer = TeamInvitationsSerializer(invitations,
                                               many=True,
                                               context=serializer_context)
        data = {'data': serializer.data}
        return Response(status=status.HTTP_200_OK, data=data)
```



FEL 2.0 API

api-token-auth	Show/Hide	List Operations	Expand Operations
game	Show/Hide	List Operations	Expand Operations
invitation	Show/Hide	List Operations	Expand Operations
license	Show/Hide	List Operations	Expand Operations
player	Show/Hide	List Operations	Expand Operations
team	Show/Hide	List Operations	Expand Operations

GET /api/v1/team/ Team view

Implementation Notes

Team view

Parameters

Parameter	Value	Description	Parameter Type	Data Type
page		A page number within the paginated result set.	query	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200			

Try it out!

POST	/api/v1/team/	Create new player
DELETE	/api/v1/team/{id}/	Team view
GET	/api/v1/team/{id}/	Team view
PATCH	/api/v1/team/{id}/	Team view

```
import React, { Component } from 'react';
import { ListGroup, ListGroupItem } from 'reactstrap';

class TeamList extends Component {

    constructor() {
        super();
        this.state = {
            teams: []
        }
    }

    // Fetches the data, once page has loaded
    componentDidMount() {
        fetch('<API_LINK>').then(response => {
            return response.json();
        }).then(data => {
            let teams = data.results;
            this.setState(teams);
        })
    }

    // Renders single item from the dataset
    renderTeams(team) {
        return (
            <ListGroupItem key={team.id}>{team.name}</ListGroupItem>
        )
    }

    // Renders the whole component using Bootstrap’s ListGroup
    render() {
        return (
            <div>
                <ListGroup>
                    {this.state.teams.map(this.renderTeams)}
                </ListGroup>
            </div>
        )
    }
}

export default TeamList;
```