Budapest University of Technology and Economics
Department of Telecommunications and Media Informatics

Full-stack platform building and client-side data visualization on sport analytics data

Project Laboratory II.

Ákos Norbert Sepp
Business Informatics MSc

Advisors:
László Toka PhD and Alija Pašić PhD
Department of Telecommunications and Media Informatics

Budapest, Hungary

2019.

Table of contents

1. Importance of data analysis in football

‘Data is everywhere in the sport field. […] Gamblers regularly relied on data to know which horse or team to bet on. Coaches routinely utilized data to evaluate talent and potential for players. […] The use of data to help executives more appropriately manage their organizations can be seen in numerous examples across numerous division within any sport organization. The question often raised by those looking at data is: are we in fact examining the right issues with the correct data and making the correct correlation between the correct data points?’ (Fried and Mumcu, 2017.)

Looking for structure, patterns and relationships is a natural human behavior. Due to this basic instinct we are capable of finding in our world features, trends or anomalies. Visualization supports this by presenting the whole, complex picture in a qualitative and quantitative overview, enabling to understand our data more deeply and more effectively. (Fayyad, et al.)

According to Fried and Mumcu, data analysis became a very important factor in modern sport in the last few years. Football, as one of the most popular sports, faces extra challenges in this field. Football managers are extremely under pressure by fans and club owners who are eager to see instant and continuous results. On the other hand, players, as the most valuable assets of the game and the football business machine, are more and more focused on their health status and factors that can improve their performance.

It is clear that all sides are unbelievably motivated to boost their chances to succeed and looking for all kinds of ways to reach their goal, which has been supported by investors in infrastructural ways such as building new stadiums, training accessories or regeneration centers. Fortunately, in the last decade players and managers found a new source of support from scientists and data analysts.

In 2019, Leeds United FC manager Marcelo Bielsa revealed his method of preparation for football matches during the scandalous event called ‘Leeds Spygate’ (The Guardian, 2019). In the presentation he showed how deeply data analysis is being involved in the process, including watching more than 200 hours gameplay videos and possible team formations regarding the previous 50 matches of the opponent. His visionary method and presentation simultaneously impressed and confused the English audition, which implies that there are still plenty of ways to develop analytical methods in the best football leagues.

But why is sports analytics so important for a small country like Hungary? In recent years, Hungarian football started to develop step-by-step, due to governmental financial supports in football academies and infrastructure. Although the circumstances nowadays are far better than used to be to reach the expected results in international competitions, it is clear that there are still several fields to improve such as data analysis. As a first step, trainers recently started to measure and record specific player performance data for further analysis.

In this study I would like to perform data visualization on sport analytics data and share the results with the “end users”, the above mentioned trainers and managers who gathered the data to harness some deeper insight about their players’ performance.

To reach this goal, firstly I had to update and modify a currently working online platform where all the analytics can take place. In the following pages I would like to present both the web development and the data visualization processes and results.

2. Goals of the study

In this study I was dedicated to reach the following goals:

1) *Update and modify the original web service platform, in order to enable trainers and managers to store, access and analyze their own data on demand*
2) *Improve analytical effectiveness in the website with data visualization and translation*

After the identification of these two main goals, I was able to define subtasks in order to create a roadmap to the desired goals.

The first goal was be divided into this six steps below:

1) Analysis of initial webpage and services
2) Defining abilities to develop
3) Developing necessary front-end solutions
4) Developing front-end serving backend solutions
5) Testing and deploying updated platform
6) Evaluating web service performance

For the second goal, I defined a roadmap with the following five steps:

1) Understanding data and visualization requirements

2) Choosing visualization tool
3) Implementing data visualization
4) Testing and deploying visualization results
5) Evaluating results

In the next pages I will move along this two roadmaps parallel to each other, presenting ideas and results next to each other. I chose this approach mainly because every step in the first roadmap serves a parallel step in the second roadmap. In other words, we cannot understand one step without examining the underlying purpose in another.

3. Examining data and related platform technologies

3.1. Catapult system – the source of data

At first I would like to present the key to all the analysis, the data itself. Our sport data was recorded by a system called Catapult, one of the world's leading wearable sport science tool, currently used by the Hungarian footballers in question.

Catapult is complex movement tracker and recorder system, specifically developed to analyze, compare and evaluate sporting movements for professional athletes. As a result of the automatic data gathering provided by the Catapult system, key performance indicators became objectively measurable and calculable.

The Catapult system records and immediately calculates more than a 1,000 different attributes simultaneously with built-in sensors like accelerometer, gyroscope or magnetometer, and pre-categorizes them on-the-run. Table 1 helps to understand more deeply the huge scale of data measured by Catapult (attributes with italic style are not automatically provided).

Internal load		**External load**	
cardiovascular	metabolic	locomotorical	mechanical
pulse	*blood serum*	GPS	IMA
pulse data	*lactate*	covered distance	acceleration, deceleration
time spent in target zone	*CK*	speed	fast changes of direction

rest HRV	*IGG*	time spent is speed zones	jumps

Table 1 - KPIs measured by Catapult system (Source: http://www.cardioc.eu/catapult-arendszerfelepitese/)

As we can see in Table 1, there are several different types of measured data provided by the Catapult system. In our study, we focus on external loads, more specifically ten selected KPI, listed below:

- Total distance covered (meter)
- High-intensity distance covered (meter)
- Sprint distance covered (meter)
- Distance >30 km/h covered (meter)
- Number of high-intensity efforts taken (whole number)
- Number of sprint efforts taken (whole number)
- Maximum velocity (kilometer / hour)
- Number of high IMA events (whole number)
- Explosive distance +/- 2 m/s2 (meter)
- Total player load

Although the process of getting the ten KPIs are really interesting, in this study I will not show how they were programmatically processed into the optimal form to perform analytical tasks. We are starting our work from the preprocessed data.

3.1. Front-end development technologies used in the study

For the webpages I used the most recent and most used front-end technologies, as one of the requirements stated to do so, HTML5, JS1.7 and CSS3 (Figure 3).

Figure 1 - The front-end triangle (Source: https://www.toughlex.com/technologies/front-end)

The front-end triangle is a vastly evolving group of programming languages that provides a multi-platform opportunity for web development.

Firstly, the HTML gives the webpage a structure, basically its main function to create a space for all the content. Secondly, CSS is responsible for styling, its purpose to make the user-experience smoother, more enjoyable and understandable. Thirdly, JavaScript is responsible for all the behavioral part of the webpage. It contains the business logic, in our case it collects the provided data (storing part), sends and receives the requests and data (accessing part), processes it (analyzing part), than generates the output for the user (visualization part). For the accessing part of JavaScript, I used AJAX which is a web-developer technique for communicating with the backend.

AJAX stands for Asynchronous JavaScript and XML, and typically it helps to create a more interactive website due to the fact that the site does not need a full reload to show some information. Thanks to the asynchronous property, it optimizes the communication with the server based on the bandwidth and the downloaded data's size (e.g. load on demand). Aside from optimization it enables the developer to separate the content, the functionality and the appearance on a page, boosting the user experiment factor.

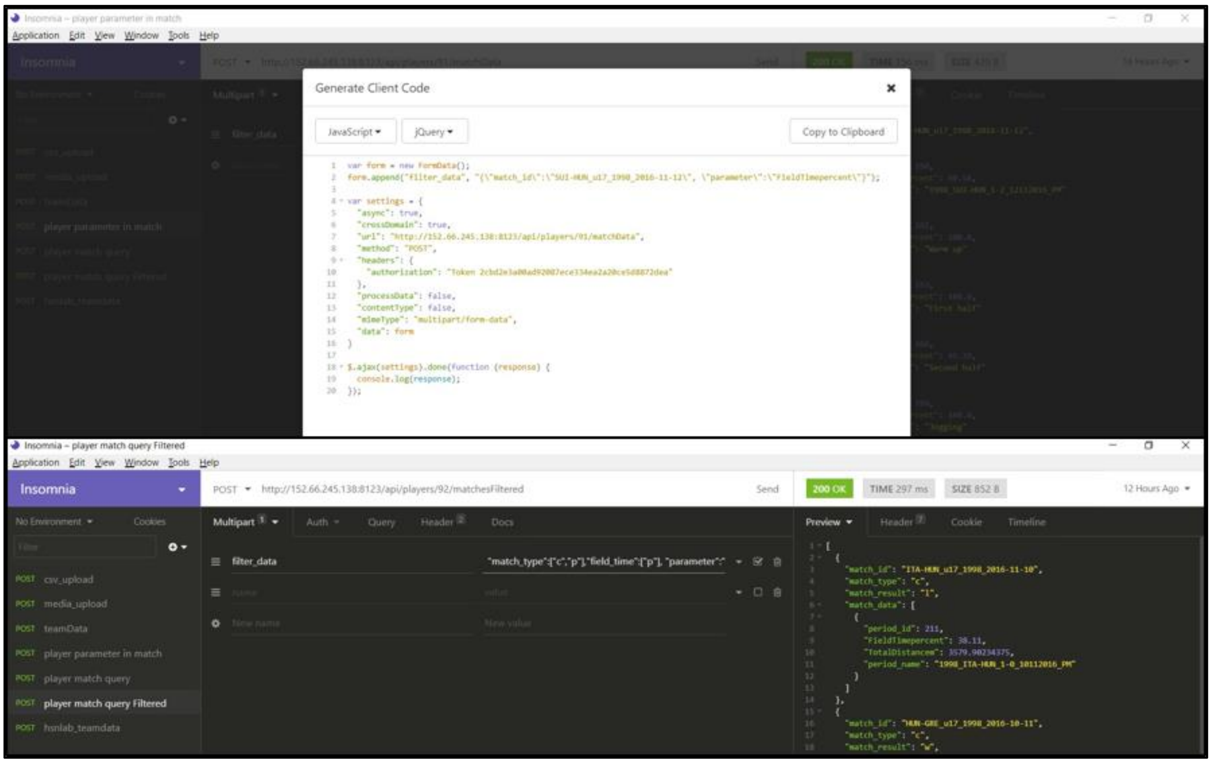
In Figure 2 a few examples are shown from the source code.

Figure 2 - Examples of developer code for the webpage (Source: own source)

There were two other important technological parts for the front-end development, third party front-end libraries and standalone programs.

From the three libraries, the first was JQuery 3.3.1, a JavaScript library for event handling, animation, DOM manipulation and document traversal, which I used through an Ajax API. The second was Bootstrap 4.1, a front-end component library, which I mostly used for building the HTML and CSS sections, through BootstrapCDN. The third was Plotly.JS, which is an open-source interactive graphing library that I also accessed through an API.

There are two standalone programs I would like to mention, Insomnia and Chrome Inspector. Insomnia is an advanced, cross-platform HTTP client with built-in helpers, templates, request chaining methods and code snippets. As Figure 3 shows, it handles requests and data pretty well, and helps to organize the API queries. In Figure 4 we can observe the Chrome Inspector, which is a necessary tool for front-end development as it provides deeper understanding and flexible testing real-time and without real code modification.



```
var form = new FormData();
form.append("filter_data", "{\"match_id\":\"SUI-HUN_u17_1998_2016-11-12\", \"parameter\":\"FieldTimepercent\"}");

var settings = {
  "async": true,
  "crossDomain": true,
  "url": "http://152.66.245.138:8123/api/players/91/matchData",
  "method": "POST",
  "headers": {
    "authorization": "Token 2cbd2e3a00ad92007ece334ea2a20ce5d8872dea"
  },
  "processData": false,
  "contentType": false,
  "mimeType": "multipart/form-data",
  "data": form
}

$.ajax(settings).done(function (response) {
  console.log(response);
});
```

```
POST http://152.66.245.138:8123/api/players/92/matchesFiltered
filter_data    "match_type":["c","p"],"field_time":["p"],"parameter":"
```

```
[
  {
    "match_id": "ITA-HUN_u17_1998_2016-11-10",
    "match_type": "c",
    "match_result": "l",
    "match_data": [
      {
        "period_id": 211,
        "FieldTimepercent": 38.11,
        "TotalDistancem": 3579.90234375,
        "period_name": "1998_ITA-HUN_1-0_10112016_PM"
      }
    ]
  },
  {
    "match_id": "HUN-GRE_u17_1998_2016-10-11",
    "match_type": "c",
    "match_result": "w",
```

Figure 3 – Insomnia user interface (Source: own source)

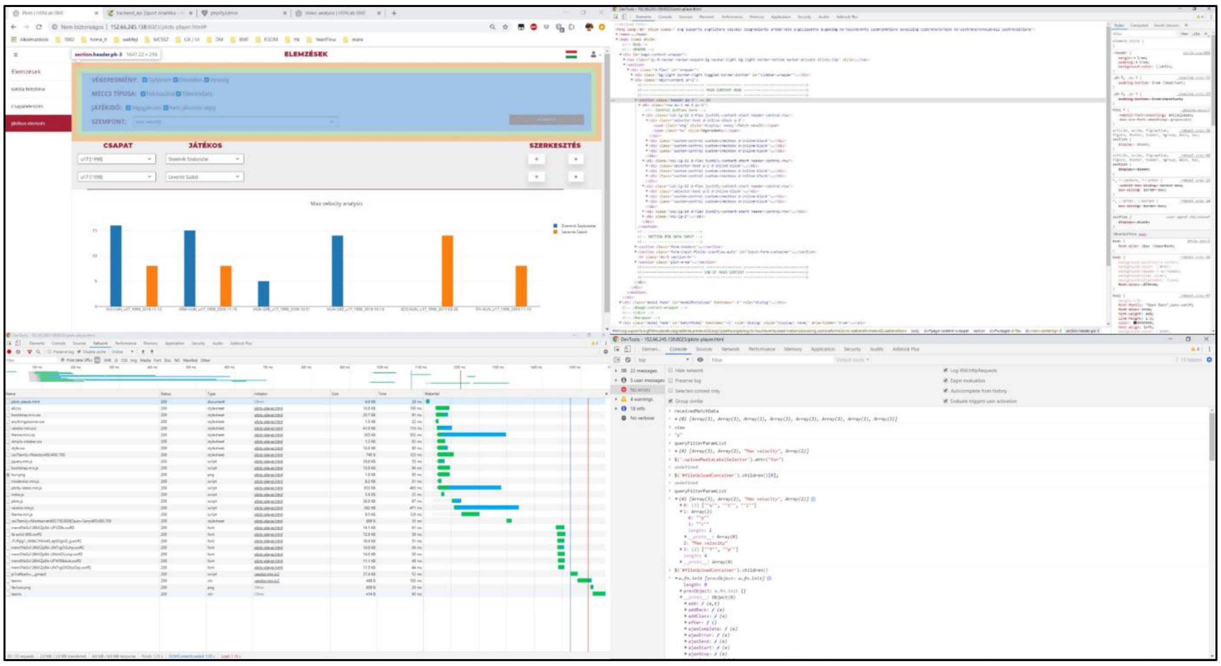


Figure 4 – Chrome Inspector in different use cases (Source: own source)

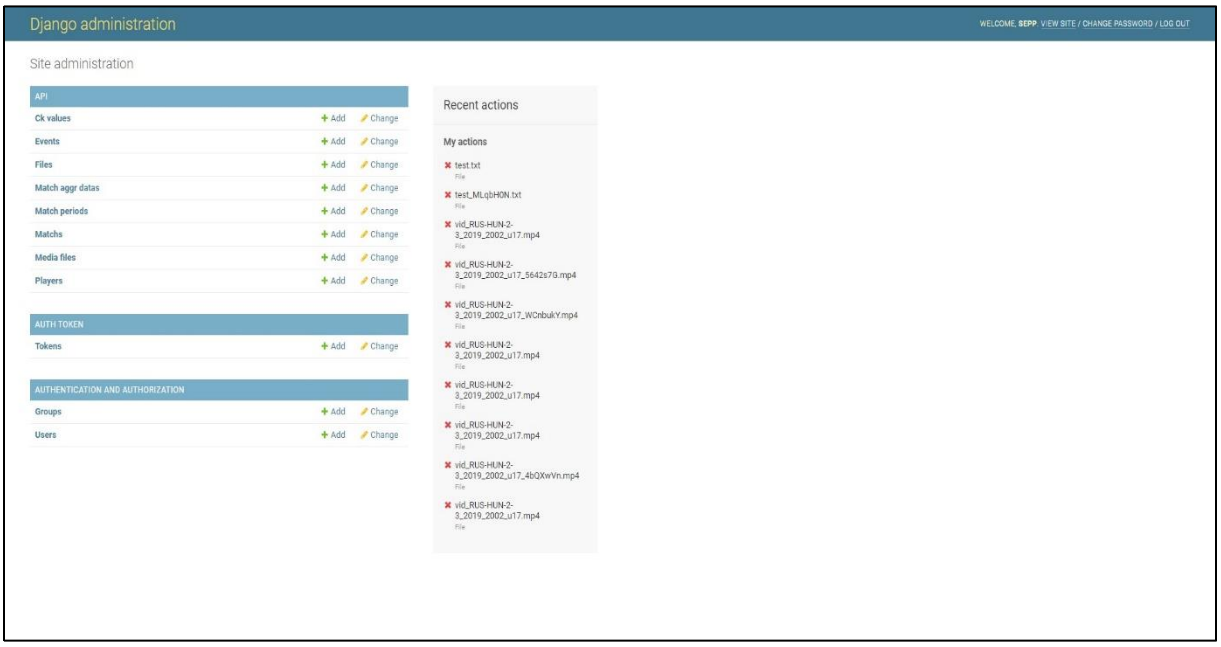
3.2. Backend development technologies used in the project

The backend was built in a high-level Python Web framework called Django. Django is an open-source and free framework with a strong developer community and an easy-to-understand and maintain inside logic, called Django MVT Structure. MVT is short for Model-View-Template. This three data structures are souls of the Django Framework.

Model is the logical data structure, which provides a definition of data formats. Views receives data via request methods like POST and GET from the user and it formats the data so it can be stored or requested to or from the database. Templates is merely a convenient way for HTML generation, and as in this project we needed specific front-end, I did not used templates at all.

Django also has strong built-in features such as authorization or administration, which enables the developer to manipulate data in a graphical interface as well (Figure 5).

The other important part of the backend is the database. While Django's built-in SQLite database could be good for a starting project, the developer does not have full control over the database to perform any operation on current or future data tables - unlike in MySQL or Oracle. In this project it was a very important factor to handle the database totally, however the project preferred a free solution. That was the reason why an open-source object-relational database system called PostgreSQL was the best choice. PostgreSQL also has a strong reputation for reliability, feature robustness and performance, and the developer can also work with in a graphical environment (Figure 6).



Django administration

WELCOME, SEPP. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

API		
Ck values	+ Add	Change
Events	+ Add	Change
Files	+ Add	Change
Match aggr datas	+ Add	Change
Match periods	+ Add	Change
Matchs	+ Add	Change
Media files	+ Add	Change
Players	+ Add	Change

AUTH TOKEN		
Tokens	+ Add	Change

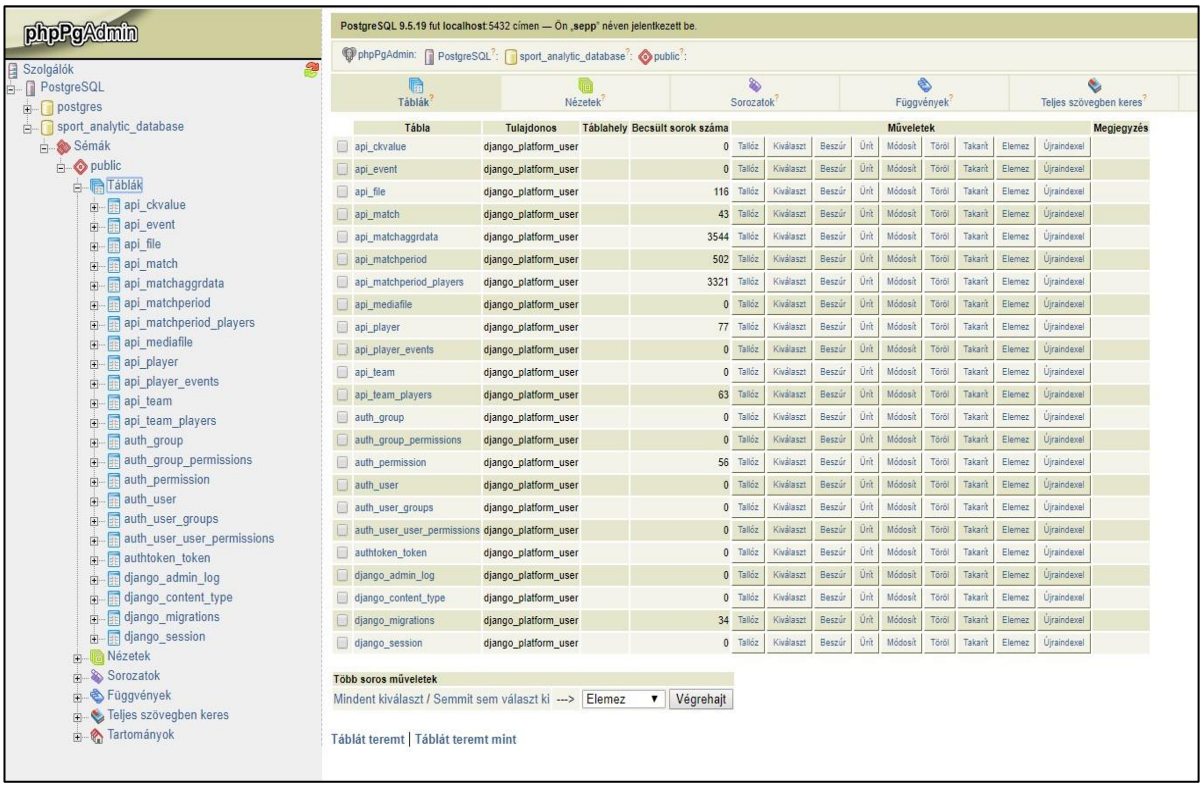
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

Recent actions

My actions

- test.txt
  File
- test_MLqbHON.txt
  File
- vid_RUS-HUN-2-3_2019_2002_u17.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17_5642s7G.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17_WCnbukY.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17_4bQXwVn.mp4
  File
- vid_RUS-HUN-2-3_2019_2002_u17.mp4
  File

Figure 5 – Django GUI (Source: own source)



phpPgAdmin

- Szolgálók
  - PostgreSQL
    - postgres
    - sport_analytic_database
      - Sémák
        - public
          - Táblák
            - api_ckvalue
            - api_event
            - api_file
            - api_match
            - api_matchaggrdata
            - api_matchperiod
            - api_matchperiod_players
            - api_mediafile
            - api_player
            - api_player_events
            - api_team
            - api_team_players
            - auth_group
            - auth_group_permissions
            - auth_permission
            - auth_user
            - auth_user_groups
            - auth_user_user_permissions
            - authtoken_token
            - django_admin_log
            - django_content_type
            - django_migrations
            - django_session
          - Nézetek
          - Sorozatok
          - Függvények
          - Teljes szövegben keres
          - Tartományok

PostgreSQL 9.5.19 fut localhost:5432 címen — Ön „sepp" néven jelentkezett be.

phpPgAdmin: PostgreSQL: sport_analytic_database: public:

Táblák | Nézetek | Sorozatok | Függvények | Teljes szövegben keres

Tábla	Tulajdonos	Táblahely	Becsült sorok száma	Műveletek									Megjegyzés
api_ckvalue	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_event	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_file	django_platform_user		116	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_match	django_platform_user		43	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_matchaggrdata	django_platform_user		3544	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_matchperiod	django_platform_user		502	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_matchperiod_players	django_platform_user		3321	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_mediafile	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_player	django_platform_user		77	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_player_events	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_team	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
api_team_players	django_platform_user		63	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
auth_group	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
auth_group_permissions	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
auth_permission	django_platform_user		56	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
auth_user	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
auth_user_groups	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
auth_user_user_permissions	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
authtoken_token	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
django_admin_log	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
django_content_type	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
django_migrations	django_platform_user		34	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	
django_session	django_platform_user		0	Tallóz	Kiválaszt	Beszúr	Ürít	Módosít	Töröl	Takarít	Elemez	Újraindexel	

Több soros műveletek

Mindent kiválaszt / Semmit sem választ ki ---> Elemez ▼ Végrehajt

Táblát teremt | Táblát teremt mint

Figure 6 – PostgreSQL graphical interface (Source: own source)

4. Planning development

4.1. Presenting initial platform

In order to thoroughly understand how and what did I need to develop, firstly I would like to present the system logic behind the web service platform (Figure 7). As one who participated in the whole development process in the past, I was familiar with the inner construction and the capabilities of the site. For deeper understanding, I briefly summarize the most important details about the platform in the next few paragraphs.

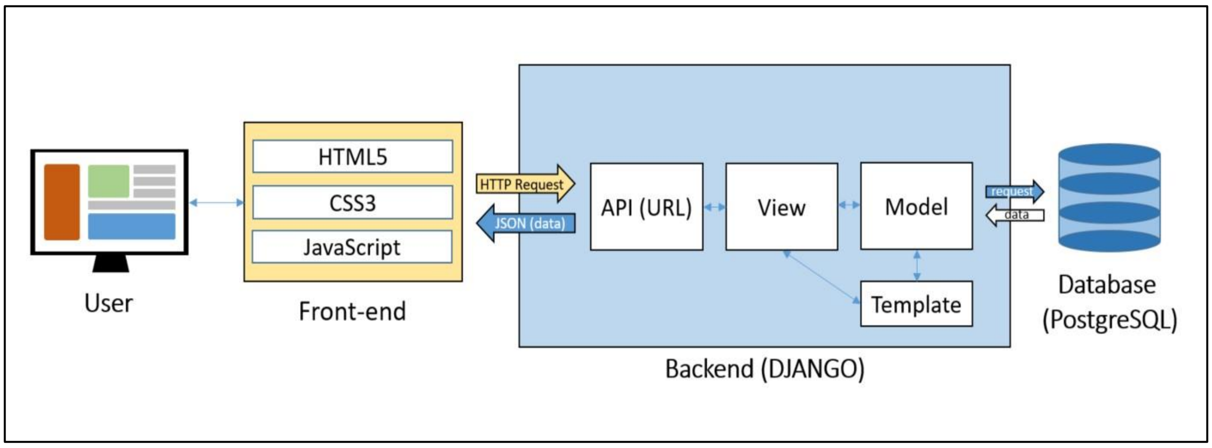


Figure 7 – Idea behind the analytic platform (Source: own source)

As Figure 7 shows, the platform basically has three important parts: the front-end, which is the part of the platform a user can reach and interact with, the backend, which practically a logical and communicational bridge as it handles and resolves requests, and the database, which contains and provides data.

The front-end sends HTTP requests to the backend, more precisely to the API application of the backend. The API sorts and forwards the requests to the corresponding Django View. The View is the central processing part of the backend, this is where the backend evaluates the requests. If needed, it sends the requests to the proper database via the according Model, then returns the data in question to the front-end, typically in JSON format.

The three parts uses three different ways to communicate: the front-end works with JavaScript for event handling, the DJANGO backend uses Python as programming and processing language, and the database handles requests with SQL queries. In order to solve all tasks, one should be familiar with all three languages.

To understand this study's results, in Figure 8 I present the current status of the front-end.

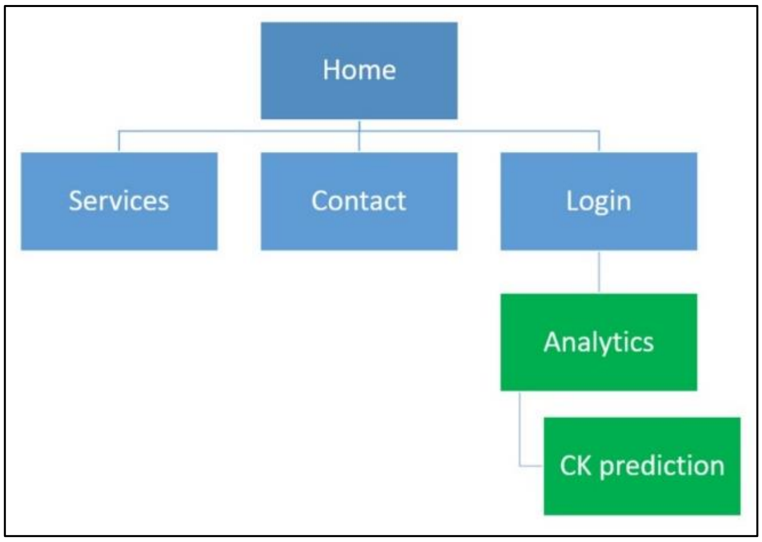


Figure 8 - Initial site arrangement (Source: own source)

The first thing to mention about Figure 8, is that pages in blue color are public sites, which means any user on the Internet can visit them. However, pages in color green are private sites, which are protected by a REST API Authorization. This site build-up is extremely convenient, because in that way I only need to add sites in the private section, and maintain security with the token provider solution, and I do not have to worry about other security issues.

The second thing is pretty obvious: the current page cannot perform any of the desired functions. However, what we cannot see on Figure 7, is the so called backend logic in the background, which makes sure about data processing, event handling and HTTP requests. Fortunately, during backend development, a few important things were considered about possible future needs, such as file upload handling. That means, that although there is not a file upload site yet, I only have to partially modify the server-side logic, and do not have to write it from scratch.

4.2. Defining necessary modifications

In Section 2 I defined the two main goals of this study. The first goal focuses on three keywords: store, access, analyze. The second goal adds two more keywords to list, visualization and translation. In order to make ends meet, I had to make updates in the front-end as well as in the backend. In this section I would go along this list of keywords and broadly describe the connecting necessary modifications and their front-end and backend backgrounds.

Storing

Without corresponding data, analysis is impossible, so as a first step, the site must receive the user’s data. In order to solve this, the front-end needed a new site, containing a form which enables the user to select and send data to the backend. The backend had to be prepared to understand (API and Model), process and store (View) the incoming data (csv of video files).

Accessing

Similarly to the previous point, I had to add another member to the private sites’ list. However, in the access process, the platform manages a two-way traffic, because it not only sends a request, it waits for the requested data as well. What is more, the received JSON file must be parsed and processed before visually appearing it to the user. The backend had to face the same three challenges, and the same three backend-part handled them, as it was explained in the storing process.

Analyzing

This step creates the connection between the two goals of this study, because without previous analytical processes, one cannot visualize data.

The analysis happened in both the backend and the front-end. Technically, the backend preprocesses the data, with functions that sort out incomprehensible or unnecessary values, and return only the related data. The front-end takes the analysis one step further, it tries to present it in a way that a human eye can interpret easily.

Visualizing

Thanks to the previous analyzation process, in this part of the development only the front-end needed additional functionality. The already processed data was provided to the special JavaScript library Plotly.JS, which made the hard work. At least, another site was added where the front-end could generate charts and plots for the user.

Translation

As the previous four steps both required modifications in the backend and the front-end, this step was an odd-one-out, as it only leaned on the front-end and was a pure JavaScript task to solve.

During my work on the webpage, I wanted to create a simple and clear website for the user, where all information are available simultaneously, and easy to handle.

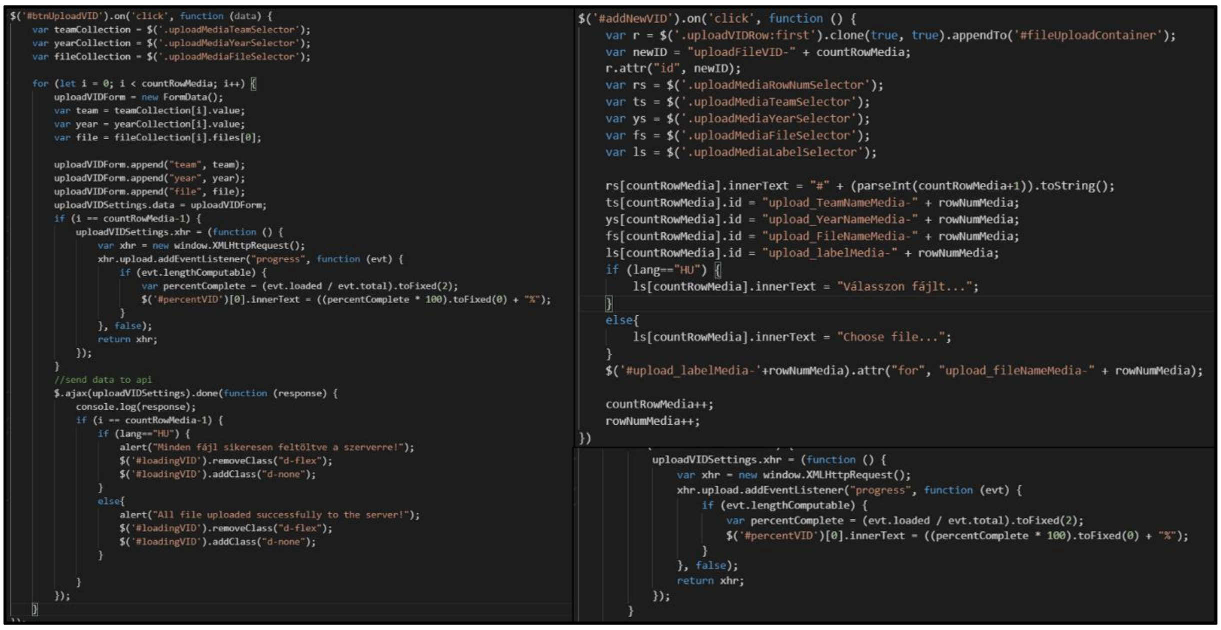
5. Platform development

5.1. Front-end modifications

Solving data storing on the front-end

The first thing to do was to create a site where users can upload their files. Therefore I created a new private site called 'Media', which had two different sections: a video and a data file uploader form. The two form's mechanism were basically the same (Figure 9):

- dynamically changeable number of form rows for file upload
- multiple AJAX requests sent to the backend with uploaded files
- monitored connection with real-time feedback about upload status



```
$('#btnUploadVID').on('click', function (data) {
    var teamCollection = $('.uploadMediaTeamSelector');
    var yearCollection = $('.uploadMediaYearSelector');
    var fileCollection = $('.uploadMediaFileSelector');

    for (let i = 0; i < countRowMedia; i++) {
        uploadVIDForm = new FormData();
        var team = teamCollection[i].value;
        var year = yearCollection[i].value;
        var file = fileCollection[i].files[0];

        uploadVIDForm.append("team", team);
        uploadVIDForm.append("year", year);
        uploadVIDForm.append("file", file);
        uploadVIDSettings.data = uploadVIDForm;
        if (i == countRowMedia-1) {
            uploadVIDSettings.xhr = (function () {
                var xhr = new window.XMLHttpRequest();
                xhr.upload.addEventListener("progress", function (evt) {
                    if (evt.lengthComputable) {
                        var percentComplete = (evt.loaded / evt.total).toFixed(2);
                        $('#percentVID')[0].innerText = ((percentComplete * 100).toFixed(0) + "%");
                    }
                }, false);
                return xhr;
            });
        }
        //send data to api
        $.ajax(uploadVIDSettings).done(function (response) {
            console.log(response);
            if (i == countRowMedia-1) {
                if (lang=="HU") {
                    alert("Minden fájl sikeresen feltöltve a szerverre!");
                    $('#loadingVID').removeClass("d-flex");
                    $('#loadingVID').addClass("d-none");
                }
                else{
                    alert("All file uploaded successfully to the server!");
                    $('#loadingVID').removeClass("d-flex");
                    $('#loadingVID').addClass("d-none");
                }

            }
        });
    }
```

```
$('#addNewVID').on('click', function () {
    var r = $('.uploadVIDRow:first').clone(true, true).appendTo('#fileUploadContainer');
    var newID = "uploadFileVID-" + countRowMedia;
    r.attr("id", newID);
    var rs = $('.uploadMediaRowNumSelector');
    var ts = $('.uploadMediaTeamSelector');
    var ys = $('.uploadMediaYearSelector');
    var fs = $('.uploadMediaFileSelector');
    var ls = $('.uploadMediaLabelSelector');

    rs[countRowMedia].innerText = "#" + (parseInt(countRowMedia+1)).toString();
    ts[countRowMedia].id = "upload_TeamNameMedia-" + rowNumMedia;
    ys[countRowMedia].id = "upload_YearNameMedia-" + rowNumMedia;
    fs[countRowMedia].id = "upload_FileNameMedia-" + rowNumMedia;
    ls[countRowMedia].id = "upload_labelMedia-" + rowNumMedia;
    if (lang=="HU") {
        ls[countRowMedia].innerText = "Válasszon fájlt...";
    }
    else{
        ls[countRowMedia].innerText = "Choose file...";
    }
    $('#upload_labelMedia-'+rowNumMedia).attr("for", "upload_fileNameMedia-" + rowNumMedia);

    countRowMedia++;
    rowNumMedia++;
})
```

```
            uploadVIDSettings.xhr = (function () {
                var xhr = new window.XMLHttpRequest();
                xhr.upload.addEventListener("progress", function (evt) {
                    if (evt.lengthComputable) {
                        var percentComplete = (evt.loaded / evt.total).toFixed(2);
                        $('#percentVID')[0].innerText = ((percentComplete * 100).toFixed(0) + "%");
                    }
                }, false);
                return xhr;
            });
        }
```

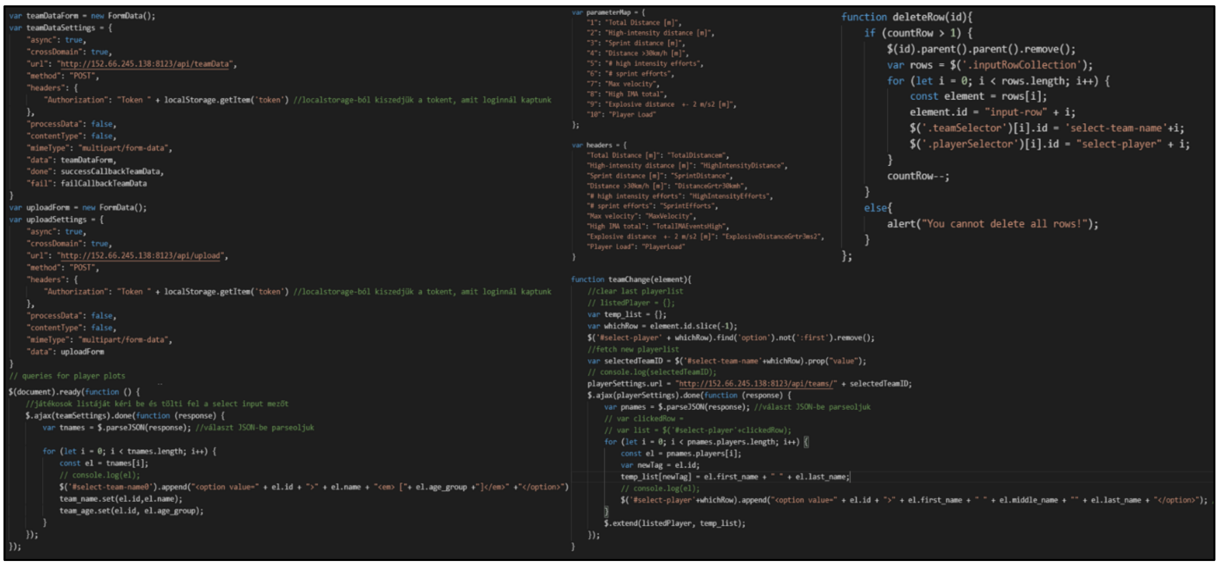
Figure 9 – Form handling logic in Model site (Source: own source)

Solving data access, analyzation and visualization on the front-end

The second thing was to make sure users can benefit from the upload. For this, I created two new private sites, one for player-specific and one for team-specific content. The main idea was the same for both sites: enabling to analyze and compare requested entity's specific data and be able to dynamically change the content of the screen. This front-end development phase involved the previously presented accessing, analyzing and visualizing steps as well.

The most important developing steps were (Figure 10 and Figure 11):

- setting up necessary AJAX communication parameters
- requesting possible entities and show them in form
- handling user interactions in forms
- testing and adjusting UX functions e.g. error messages
- processing received data in comprehensible format
- programmatically generate graphical results for user



```
var teamDataForm = new FormData();
var teamDataSettings = {
    "async": true,
    "crossDomain": true,
    "url": "http://152.66.245.138:8123/api/teamData",
    "method": "POST",
    "headers": {
        "Authorization": "Token " + localStorage.getItem('token') //localstorage-ból kiszedjük a tokent, amit loginnál kaptunk
    },
    "processData": false,
    "contentType": false,
    "mimeType": "multipart/form-data",
    "data": teamDataForm,
    "done": successCallbackTeamData,
    "fail": failCallbackTeamData
}
var uploadForm = new FormData();
var uploadSettings = {
    "async": true,
    "crossDomain": true,
    "url": "http://152.66.245.138:8123/api/upload",
    "method": "POST",
    "headers": {
        "Authorization": "Token " + localStorage.getItem('token') //localstorage-ból kiszedjük a tokent, amit loginnál kaptunk
    },
    "processData": false,
    "contentType": false,
    "mimeType": "multipart/form-data",
    "data": uploadForm
}
// queries for player plots
$(document).ready(function () {
    //játékosok listáját kéri be és tölti fel a select input mezőt
    $.ajax(teamSettings).done(function (response) {
        var tnames = $.parseJSON(response); //választ JSON-be parseoljuk

        for (let i = 0; i < tnames.length; i++) {
            const el = tnames[i];
            // console.log(el);
            $('#select-team-name0').append("<option value=" + el.id + ">" + el.name + "<em> ["+ el.age_group +"]</em>" +"</option>")
            team_name.set(el.id,el.name);
            team_age.set(el.id, el.age_group);
        }
    });
});
```

```
var parameterMap = {
    "1": "Total Distance [m]",
    "2": "High-intensity distance [m]",
    "3": "Sprint distance [m]",
    "4": "Distance >30km/h [m]",
    "5": "# high intensity efforts",
    "6": "# sprint efforts",
    "7": "Max velocity",
    "8": "High IMA total",
    "9": "Explosive distance  +- 2 m/s2 [m]",
    "10": "Player Load"
};

var headers = {
    "Total Distance [m]": "TotalDistancem",
    "High-intensity distance [m]": "HighIntensityDistance",
    "Sprint distance [m]": "SprintDistance",
    "Distance >30km/h [m]": "DistanceGrtr30kmh",
    "# high intensity efforts": "HighIntensityEfforts",
    "# sprint efforts": "SprintEfforts",
    "Max velocity": "MaxVelocity",
    "High IMA total": "TotalIMAEventsHigh",
    "Explosive distance  +- 2 m/s2 [m]": "ExplosiveDistanceGrtr3ms2",
    "Player Load": "PlayerLoad"
}

function teamChange(element){
    //clear last playerlist
    // listedPlayer = {};
    var temp_list = {};
    var whichRow = element.id.slice(-1);
    $('#select-player' + whichRow).find('option').not(':first').remove();
    //fetch new playerlist
    var selectedTeamID = $('#select-team-name'+whichRow).prop("value");
    // console.log(selectedTeamID);
    playerSettings.url = "http://152.66.245.138:8123/api/teams/" + selectedTeamID;
    $.ajax(playerSettings).done(function (response) {
        var pnames = $.parseJSON(response); //választ JSON-be parseoljuk
        // var clickedRow =
        // var list = $('#select-player'+clickedRow);
        for (let i = 0; i < pnames.players.length; i++) {
            const el = pnames.players[i];
            var newTag = el.id;
            temp_list[newTag] = el.first_name + " " + el.last_name;
            // console.log(el);
            $('#select-player'+whichRow).append("<option value=" + el.id + ">" + el.first_name + " " + el.middle_name + "" + el.last_name + "</option>");
        }
        $.extend(listedPlayer, temp_list);
    });
}
```

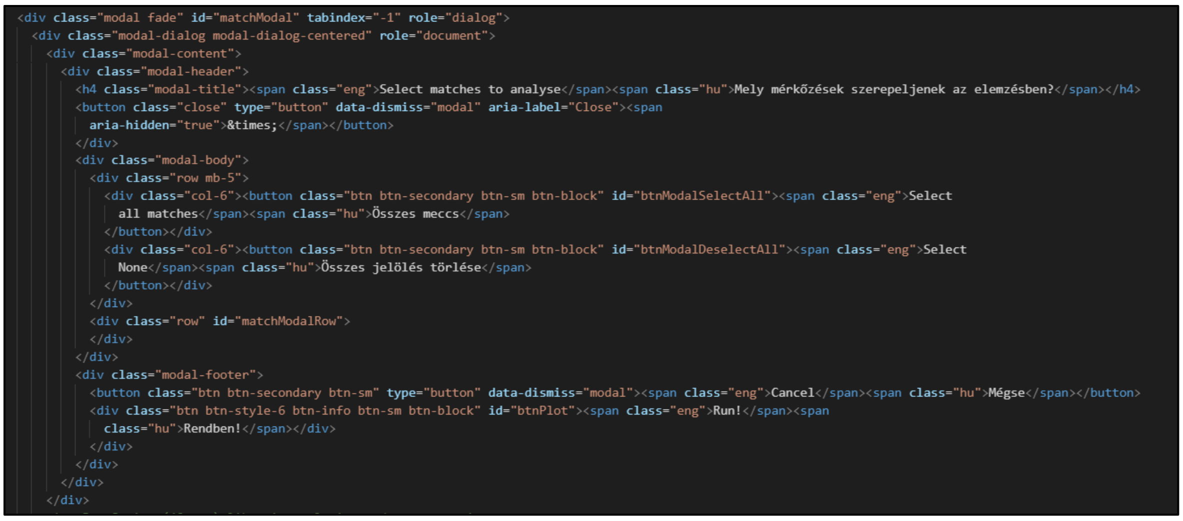
```
function deleteRow(id){
    if (countRow > 1) {
        $(id).parent().parent().remove();
        var rows = $('.inputRowCollection');
        for (let i = 0; i < rows.length; i++) {
            const element = rows[i];
            element.id = "input-row" + i;
            $('.teamSelector')[i].id = 'select-team-name'+i;
            $('.playerSelector')[i].id = "select-player" + i;
        }
        countRow--;
    }
    else{
        alert("You cannot delete all rows!");
    }
};
```

Figure 10 - Request and UI handling logic in Player and Team sites (Source: own source)

Solving translation on the front-end

The third thing to do was the translation. The request was to enable the user to easily switch between English and Hungarian languages. To solve this, there were several possible options for the implementations, from relying only on JavaScript, or involving HTML all the way to a mixed version with CSS.

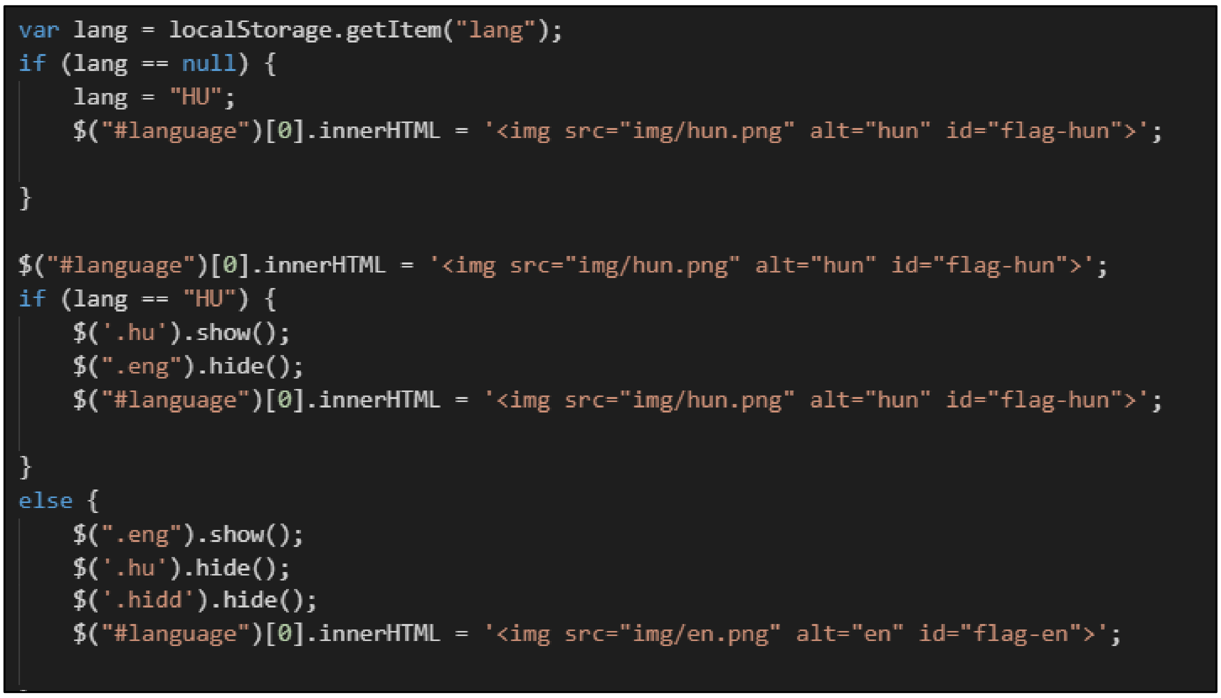
Finally I decided to go with a mixed solution, which practically meant that I had to write the JavaScript logic, but also has to modify the HTML construct of the page, hiding both the English and Hungarian version of each element's label in the code. Then I assigned the HTML element with different CSS classes to logically separate the languages (Figure 11)



```html
<div class="modal fade" id="matchModal" tabindex="-1" role="dialog">
  <div class="modal-dialog modal-dialog-centered" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h4 class="modal-title"><span class="eng">Select matches to analyse</span><span class="hu">Mely mérkőzések szerepeljenek az elemzésben?</span></h4>
        <button class="close" type="button" data-dismiss="modal" aria-label="Close"><span
          aria-hidden="true">&times;</span></button>
        </div>
        <div class="modal-body">
          <div class="row mb-5">
            <div class="col-6"><button class="btn btn-secondary btn-sm btn-block" id="btnModalSelectAll"><span class="eng">Select
              all matches</span><span class="hu">Összes meccs</span>
            </button></div>
            <div class="col-6"><button class="btn btn-secondary btn-sm btn-block" id="btnModalDeselectAll"><span class="eng">Select
              None</span><span class="hu">Összes jelölés törlése</span>
            </button></div>
          </div>
          <div class="row" id="matchModalRow">
          </div>
        </div>
        <div class="modal-footer">
          <button class="btn btn-secondary btn-sm" type="button" data-dismiss="modal"><span class="eng">Cancel</span><span class="hu">Mégse</span></button>
          <div class="btn btn-style-6 btn-info btn-sm btn-block" id="btnPlot"><span class="eng">Run!</span><span
            class="hu">Rendben!</span></div>
          </div>
        </div>
      </div>
    </div>
```

Figure 11 – Mixed solution for translation in code (Source: own source)

I chose this solution mainly because its UI transformation relies much less on users' connection speed, it only downloads bigger data (in HTML code) once, then reuses it as many times as needed. The JS code only switches between CSS styles, which does not require any additional communication or reload on the client-side (Figure 12).



```javascript
var lang = localStorage.getItem("lang");
if (lang == null) {
    lang = "HU";
    $("#language")[0].innerHTML = '<img src="img/hun.png" alt="hun" id="flag-hun">';

}

$("#language")[0].innerHTML = '<img src="img/hun.png" alt="hun" id="flag-hun">';
if (lang == "HU") {
    $('.hu').show();
    $(".eng").hide();
    $("#language")[0].innerHTML = '<img src="img/hun.png" alt="hun" id="flag-hun">';

}
else {
    $(".eng").show();
    $('.hu').hide();
    $('.hidd').hide();
    $("#language")[0].innerHTML = '<img src="img/en.png" alt="en" id="flag-en">';
```

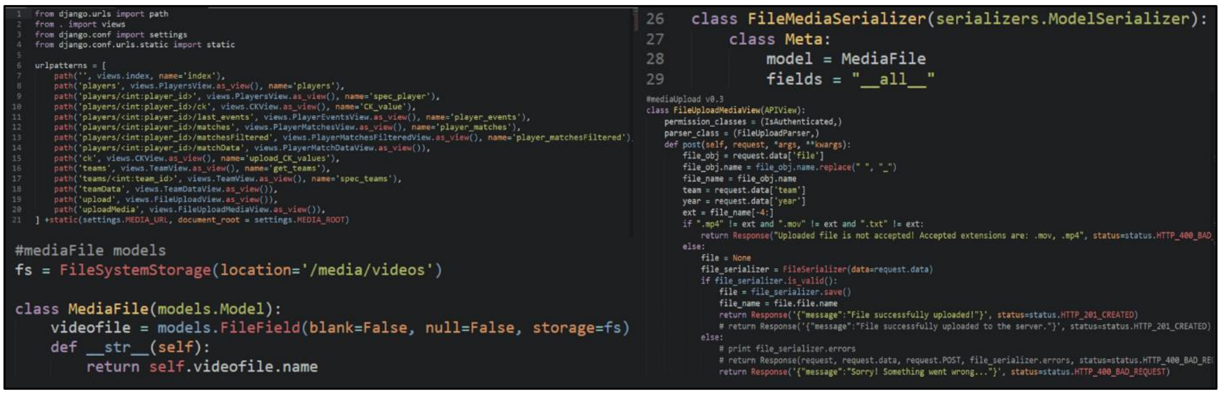
Figure 12 – Solving translation with dynamic CSS modification (Source: own source)

5.2. Backend modifications

In Section 4.1 I illustrated how communication works inside the Django framework. The requests from the user arrive to the API, which is technically a file with the proper URLs.

Then the API passes the request to the View, which handles it, based on the Models. If the request needs data, the request goes along the Model all the way to the Database, then returns with the specific data. That means, in order to achieve all necessary modifications in the backend, I had to modify these files as well. In the next paragraphs figures contain code junks from all three files, and from a helping fourth file called *serializers.py*, which basically a description for returning data structures.

Solving data storing on the backend



```
from django.urls import path
from . import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('', views.index, name='index'),
    path('players', views.PlayersView.as_view(), name='players'),
    path('players/<int:player_id>', views.PlayersView.as_view(), name='spec_player'),
    path('players/<int:player_id>/ck', views.CKView.as_view(), name='CK_value'),
    path('players/<int:player_id>/last_events', views.PlayerEventsView.as_view(), name='player_events'),
    path('players/<int:player_id>/matches', views.PlayerMatchesView.as_view(), name='player_matches'),
    path('players/<int:player_id>/matchesFiltered', views.PlayerMatchesFilteredView.as_view(), name='player_matchesFiltered'),
    path('players/<int:player_id>/matchData', views.PlayerMatchDataView.as_view()),
    path('ck', views.CKView.as_view(), name='upload_CK_values'),
    path('teams', views.TeamView.as_view(), name='get_teams'),
    path('teams/<int:team_id>', views.TeamView.as_view(), name='spec_teams'),
    path('teamData', views.TeamDataView.as_view()),
    path('upload', views.FileUploadView.as_view()),
    path('uploadMedia', views.FileUploadMediaView.as_view()),
] +static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

```
#mediaFile models
fs = FileSystemStorage(location='/media/videos')

class MediaFile(models.Model):
    videofile = models.FileField(blank=False, null=False, storage=fs)
    def __str__(self):
        return self.videofile.name
```

```
class FileMediaSerializer(serializers.ModelSerializer):
    class Meta:
        model = MediaFile
        fields = "__all__"
```

```
#mediaUpload v0.3
class FileUploadMediaView(APIView):
    permission_classes = (IsAuthenticated,)
    parser_class = (FileUploadParser,)
    def post(self, request, *args, **kwargs):
        file_obj = request.data['file']
        file_obj.name = file_obj.name.replace(" ", "_")
        file_name = file_obj.name
        team = request.data['team']
        year = request.data['year']
        ext = file_name[-4:]
        if ".mp4" != ext and ".mov" != ext and ".txt" != ext:
            return Response("Uploaded file is not accepted! Accepted extensions are: .mov, .mp4", status=status.HTTP_400_BAD_
        else:
            file = None
            file_serializer = FileSerializer(data=request.data)
            if file_serializer.is_valid():
                file = file_serializer.save()
                file_name = file.file.name
                return Response('{"message":"File successfully uploaded!"}', status=status.HTTP_201_CREATED)
                # return Response('{"message":"File successfully uploaded to the server."}', status=status.HTTP_201_CREATED)
            else:
                # print file_serializer.errors
                # return Response(request, request.data, request.POST, file_serializer.errors, status=status.HTTP_400_BAD_RE
                return Response('{"message":"Sorry! Something went wrong..."}', status=status.HTTP_400_BAD_REQUEST)
```

Figure 13 – Solving storing on the backend (Source: own source)

First of all, I had to create the API end-point in the URL list, contained by the *urls.py* file (top left corner on Figure 13), to enable the front-end to properly communicate with the Django. In the URL declaration, one must define which View should be returned on call. Therefore as a second step, I created a new view for the media files in *views.py* (bottom left corner on Figure 13), which described what should the backend do when a request is sent to it.

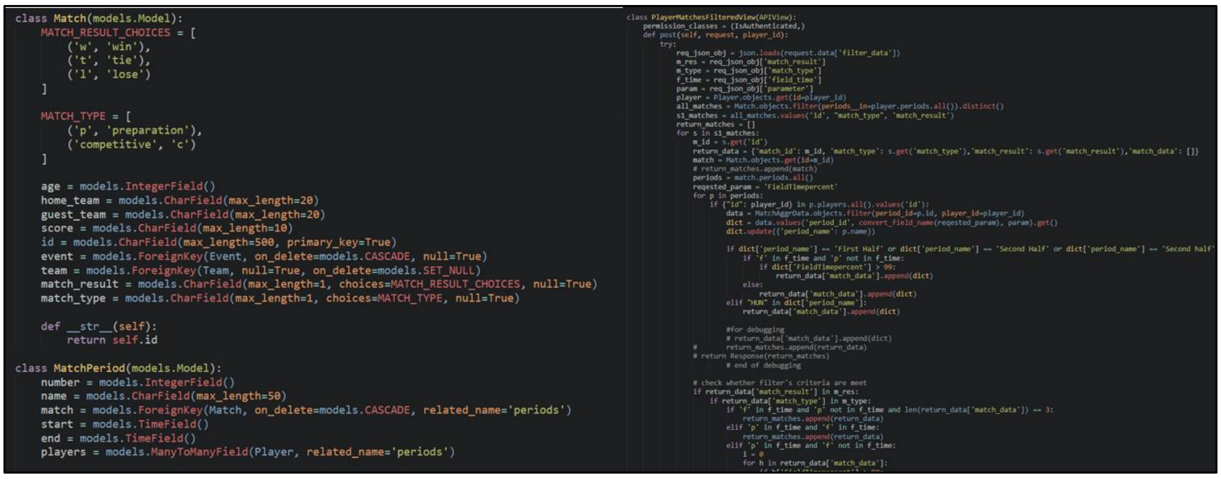
The idea behind the codes of *views.py* was this:

- define a new media file based on the received data from HTTP request
- store received meta data about the file
- call the proper serializer for uploaded file
- save the serialized file with stored metadata

In the part of file handling, I referred several times to the model, for which I invoked a model serializer, to properly interpret the returned data from the database, previously created in the *serializers.py* (top right corner on Figure 13).

As a last step I updated the model in the entire database to migrate all changes across the backend.

Solving data access, analyzation and visualization on the backend



```
class Match(models.Model):
    MATCH_RESULT_CHOICES = [
        ('w', 'win'),
        ('t', 'tie'),
        ('l', 'lose')
    ]

    MATCH_TYPE = [
        ('p', 'preparation'),
        ('competitive', 'c')
    ]

    age = models.IntegerField()
    home_team = models.CharField(max_length=20)
    guest_team = models.CharField(max_length=20)
    score = models.CharField(max_length=10)
    id = models.CharField(max_length=500, primary_key=True)
    event = models.ForeignKey(Event, on_delete=models.CASCADE, null=True)
    team = models.ForeignKey(Team, null=True, on_delete=models.SET_NULL)
    match_result = models.CharField(max_length=1, choices=MATCH_RESULT_CHOICES, null=True)
    match_type = models.CharField(max_length=1, choices=MATCH_TYPE, null=True)

    def __str__(self):
        return self.id

class MatchPeriod(models.Model):
    number = models.IntegerField()
    name = models.CharField(max_length=50)
    match = models.ForeignKey(Match, on_delete=models.CASCADE, related_name='periods')
    start = models.TimeField()
    end = models.TimeField()
    players = models.ManyToManyField(Player, related_name='periods')
```

```
class PlayerMatchesFilteredView(APIView):
    permission_classes = (IsAuthenticated,)
    def post(self, request, player_id):
        try:
            req_json_obj = json.loads(request.data['filter_data'])
            m_res = req_json_obj['match_result']
            m_type = req_json_obj['match_type']
            f_time = req_json_obj['field_time']
            param = req_json_obj['parameter']
            player = Player.objects.get(id=player_id)
            all_matches = Match.objects.filter(periods__in=player.periods.all()).distinct()
            s1_matches = all_matches.values('id', "match_type", 'match_result')
            return_matches = []
            for s in s1_matches:
                m_id = s.get('id')
                return_data = {'match_id': m_id, 'match_type': s.get('match_type'),'match_result': s.get('match_result'),'match_data': []}
                match = Match.objects.get(id=m_id)
                # return_matches.append(match)
                periods = match.periods.all()
                reqested_param = 'FieldTimepercent'
                for p in periods:
                    if {"id": player_id} in p.players.all().values('id'):
                        data = MatchAggrData.objects.filter(period_id=p.id, player_id=player_id)
                        dict = data.values('period_id', convert_field_name(reqested_param), param).get()
                        dict.update({'period_name': p.name})

                        if dict['period_name'] == 'First Half' or dict['period_name'] == 'Second Half' or dict['period_name'] == 'Second half'
                            if 'f' in f_time and 'p' not in f_time:
                                if dict['FieldTimepercent'] > 99:
                                    return_data['match_data'].append(dict)
                            else:
                                return_data['match_data'].append(dict)
                        elif "HUN" in dict['period_name']:
                            return_data['match_data'].append(dict)

                        #for debugging
                        # return_data['match_data'].append(dict)
                #       return_matches.append(return_data)
                # return Response(return_matches)
                        # end of debugging

                # check whether filter's criteria are meet
                if return_data['match_result'] in m_res:
                    if return_data['match_type'] in m_type:
                        if 'f' in f_time and 'p' not in f_time and len(return_data['match_data']) == 3:
                            return_matches.append(return_data)
                        elif 'p' in f_time and 'f' in f_time:
                            return_matches.append(return_data)
                        elif 'p' in f_time and 'f' not in f_time:
                            i = 0
                            for h in return_data['match_data']:
```

Figure 14 – Solving data access, analyzation and visualization on the backend (Source: own source)

Most of the steps were almost the same during the development process, I had to make the same modifications from the URLs all the way to the model migration. The crucial difference was the way, how the backend handled the data which was described by the *views.py* file (right side of Figure 14) and the type of models that the backend worked with (left side of Figure 14)

The logic behind the data access, analyzation and visualization contained four parts:

- the first part stores the basic file data, sent by the HTTP request from the front-end
- the second part sends a request to the database for the related data
- the third part loops through the received data, filtering and selecting matching the stored values in the first part
- the fourth step formats and returns the selected data to the front-end

The models this time were bigger entities, with several more attributes (e.g. Players with first name, last name, date of birth, preferred foot, position and events), all connected to each other.

All Players plays in one Team, in all Matches at least one side was a Team recorded in the database. Matches have several MatchPeriod, and all MatchPeriod has several attributes, recorded in MatchAggrData, typically KPIs like TotalDistance.

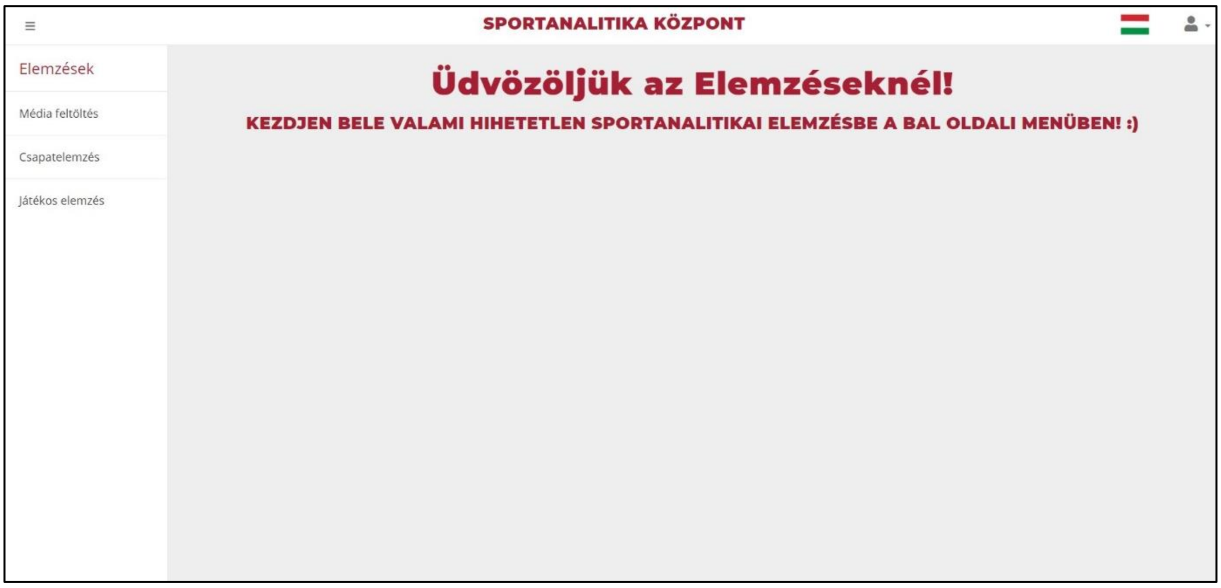
An example of the related data tables:

- *Player*: (first name: “Bob”, last name: “Bobek”, born: “2000.01.01”, foot: “left”, position: “GK”, id: 10)
- *Team*: (age: u17, year: 2000, players: “Bob Bobek”, “John Johnny”, “Mike Mikkelson”. etc.)

- *Match*: (type: preparation, result: win, home_team: “u17_2000”, guest_team: “Canada_u17_2000”, score: “2-1”)
- *MatchPeriod* (number: 13, name: “Warming up”, match: “HUN-CAN-2018-11-01…”, start: 2018-11-01-10-00-00, end: 2018-11-01-11-59-00, players: “Bob Bobek”, “John Johnny”, “Mike Mikkelson”, *etc*…)
- *MatchAggrData* (player: 10, period: 13, TotalDistance: 321.100, IMATotal: 11, *etc*...)

6. Deployment and presenting the new platform

6.1. Analytics page



Üdvözöljük az Elemzéseknél!

KEZDJEN BELE VALAMI HIHETETLEN SPORTANALITIKAI ELEMZÉSBE A BAL OLDALI MENÜBEN! :)

Figure 15 – Main page UI (Source: own source)

The main page, called *Analytics* functions as the inner landing page of the site. The left sidebar collects and lists the main functionalities of the page, the media upload, the team and the player analytics.

However *analytics.html* was not created during this study, one development was done to the site, which is translation. In Figure 15, we can see the Hungarian version of the site. On the top right corner of the picture there is the Hungarian flag, showing the language of the opened site. In section 5.1, Figure 12 we can observe as the *localStorage* changes value and the flag icon shifts from one flag to another.

6.2. Media upload

This page is the implementation of the *storage* functionality.

In Figure 16, we can see the user interface of the subpage.

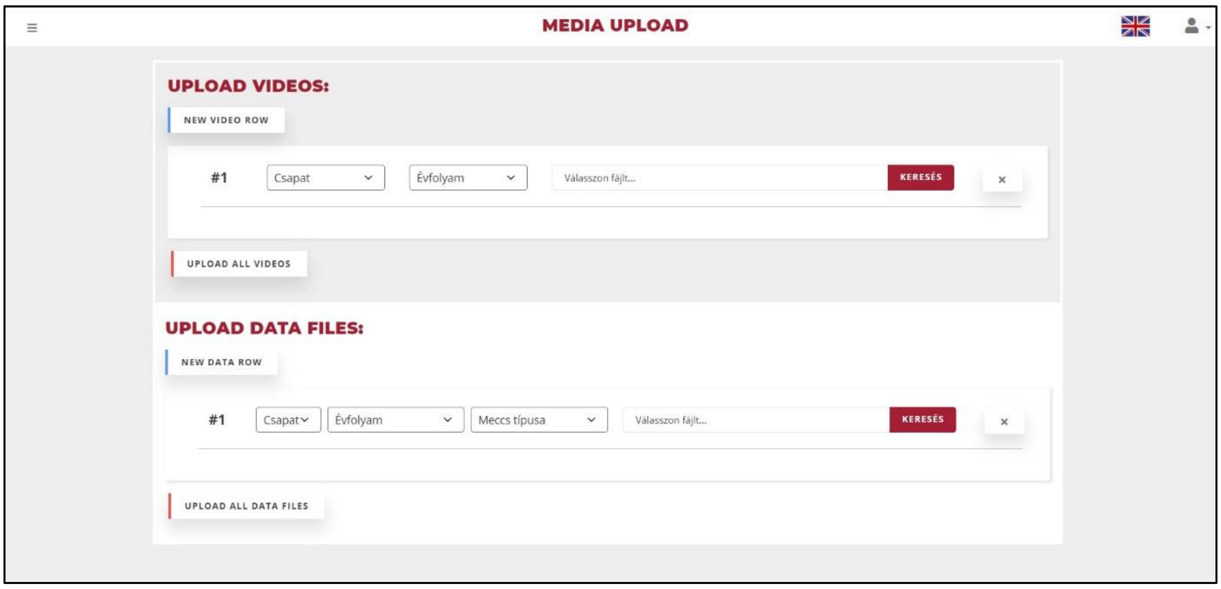


Figure 16 – Media upload UI (Source: own source)

In the interface, there are two modules, a video and a data file upload form.

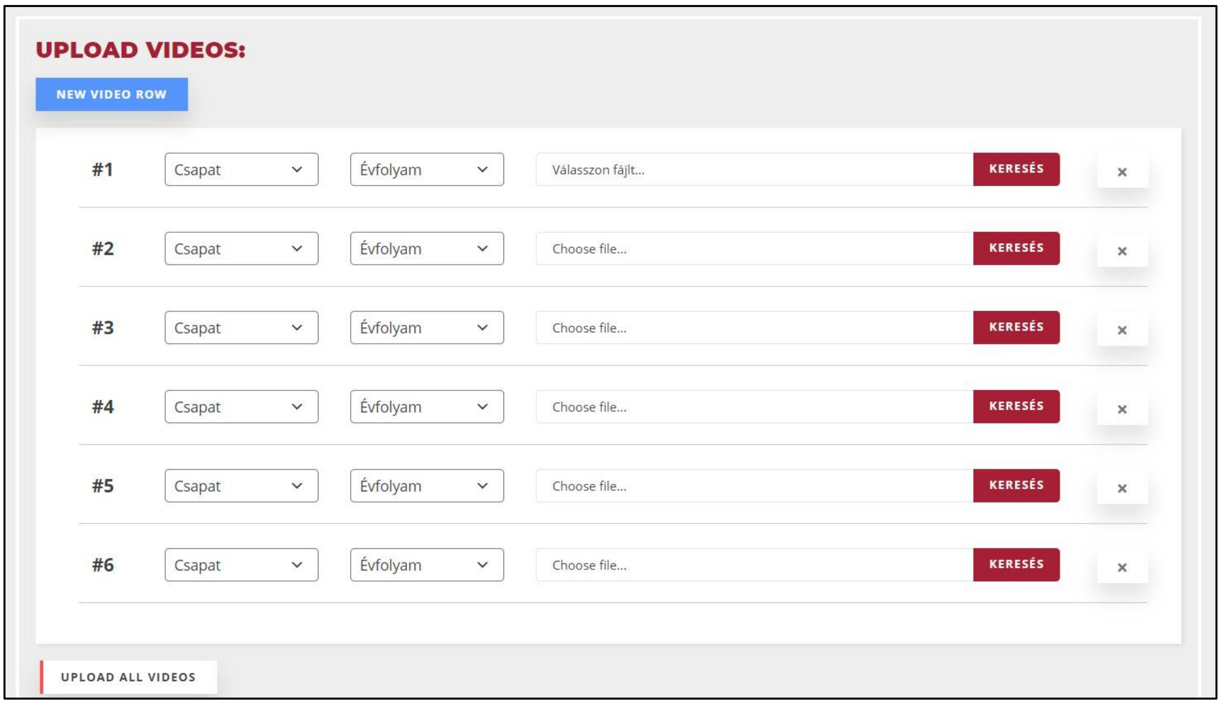


Figure 17 – Dynamical form handling (Source: own source)

Both module contain three buttons and a form, with the following possible actions:

- The button on top adds a new row to the form, enabling the user to upload another video if needed (Figure 17)
- The button at the bottom starts the uploading to the server, based on all the properly filled-in forms. During the upload, a real-time loader shows the percent of transmitted data for the user (Figure 18)
- the form itself contains the information that has to be send to the server: the age, the team and the file (and the match type, in the second form)

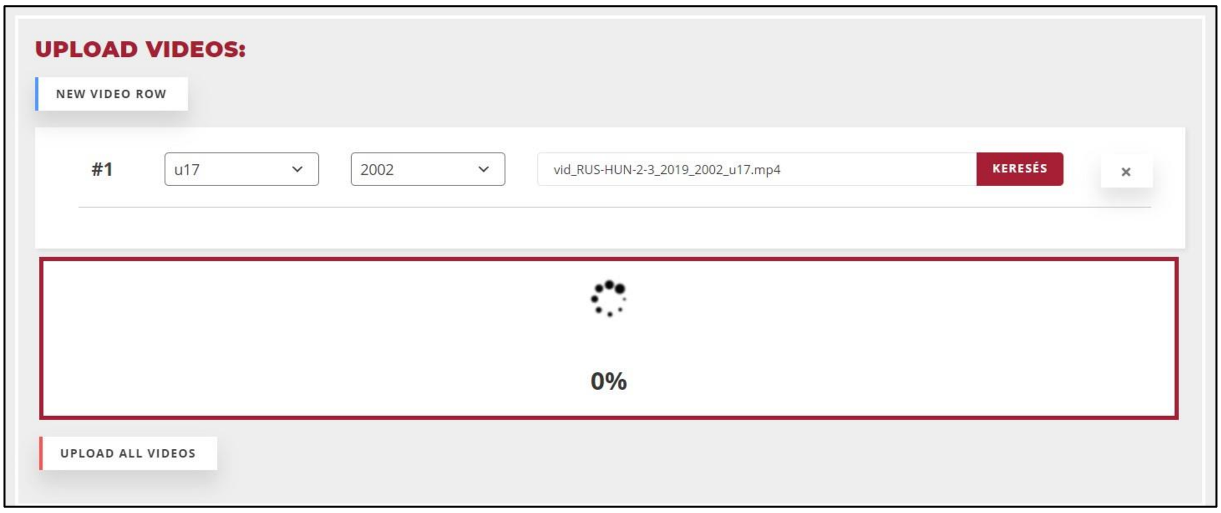


Figure 18 – Video uploading process with real-time status (Source: own source)

6.3. Team and player analytics

Team (Figure 19) and player (Figure 21) analytics platforms are the implementation of the access, analyzation and visualization functions.

Both site has the same structure:

- module for setting the parameters for the query (top part with white background)
- dynamically expandable and reducible form for selecting team or player in question
- plot are, where the front-end generated SVG appears after successful requests and answers (Figure 20 and 23)

The only difference between the two pages, that team analytics summarizes all players' data for one match, while player analytics concentrates only on certain players. Therefore in player analytics, there is another supplementary parameter to set, which is field time (showing whether the player was substituted in or out during the match).

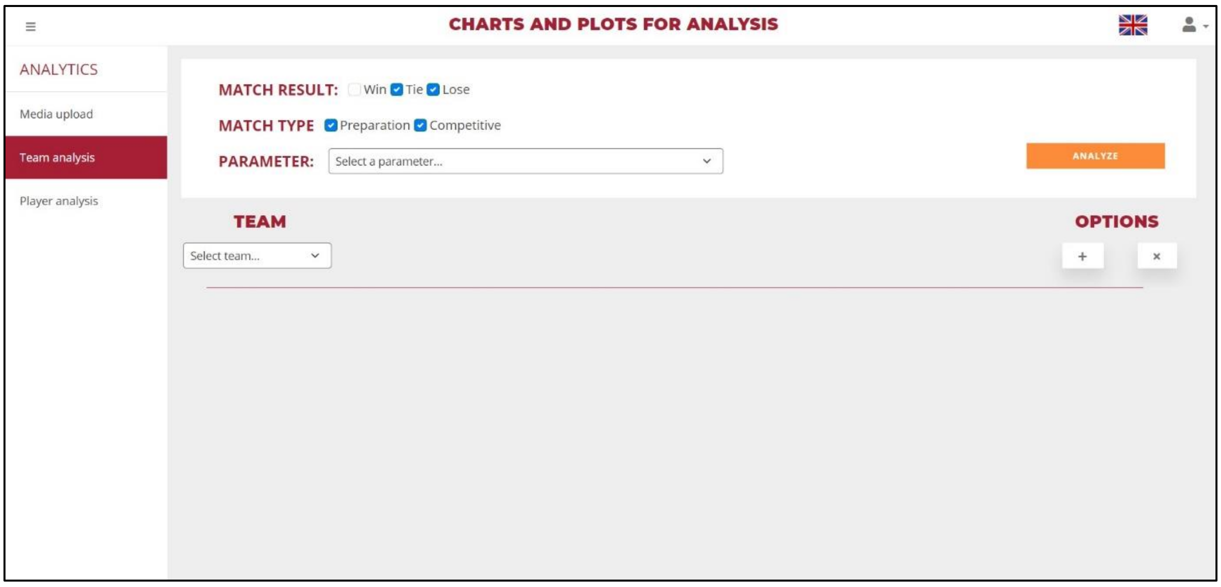

≡
CHARTS AND PLOTS FOR ANALYSIS
ANALYTICS
Media upload
Team analysis
Player analysis
MATCH RESULT: Win Tie Lose
MATCH TYPE Preparation Competitive
PARAMETER: Select a parameter...
ANALYZE
TEAM
Select team...
OPTIONS
+
×

Figure 19 – Team analysis UI (Source: own source)

As enhancing user experience was critical during my work, in front-end development I tried to create user-friendly, easy-to-use sites. For example to fill the parameter module in easily, I used simple form controls, such as checkboxes and dropdown-lists. There were no place for free text as it could entangle the HTTP request's filtering data, later not meeting the formal requirements on the backend.

Moreover, these modules have a second line of correction checker, as alert messages pop up in faulty or missed fulfillment, pointing to the exact place of missing values, in order to help the user correct the form without any problem.

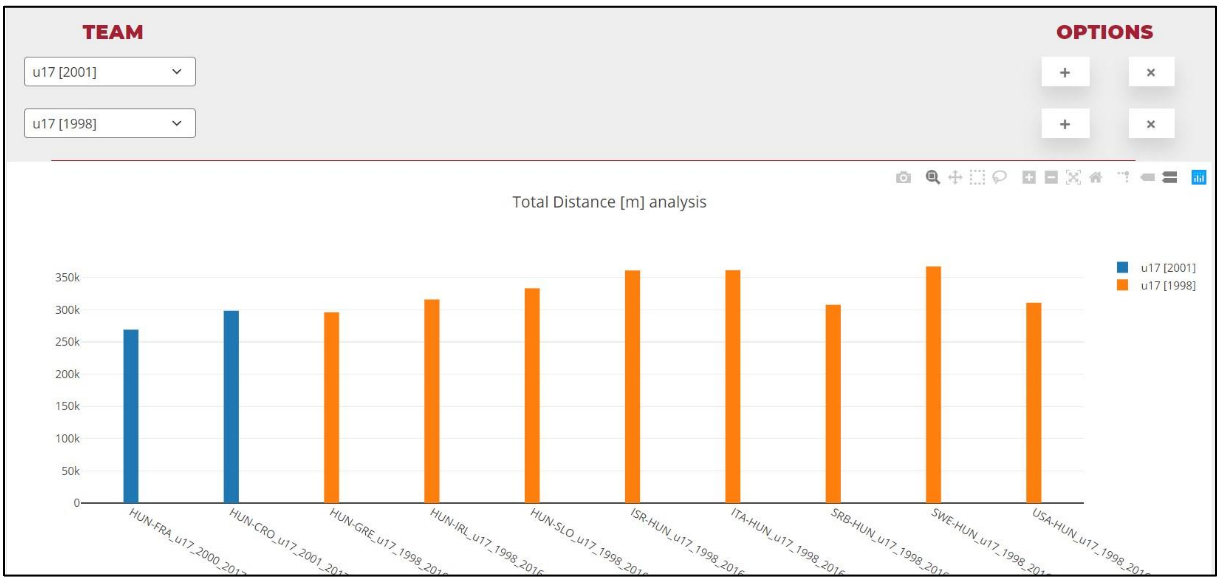


Figure 20 – Generated plot on team analysis page (Source: own source)

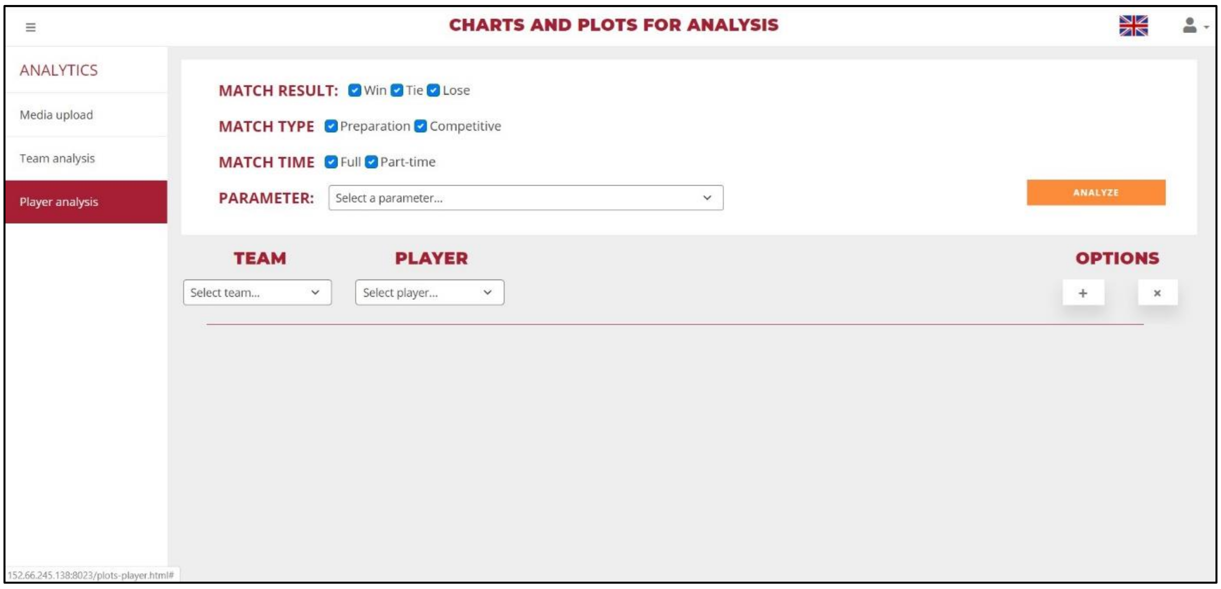
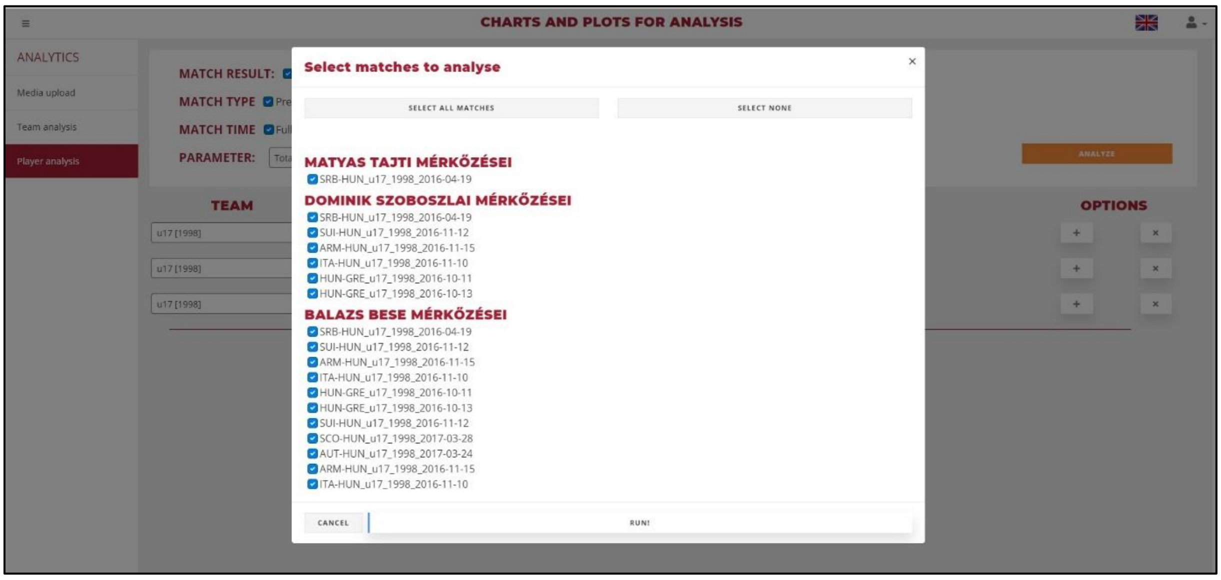

CHARTS AND PLOTS FOR ANALYSIS
ANALYTICS
Media upload
Team analysis
Player analysis
MATCH RESULT: Win Tie Lose
MATCH TYPE Preparation Competitive
MATCH TIME Full Part-time
PARAMETER: Select a parameter...
ANALYZE
TEAM
PLAYER
OPTIONS
Select team...
Select player...
+
×
152.66.245.138:8023/plots-player.html#

Figure 21 – Player analysis page UI (Source: own source)

Another functionality for enhancing user experience was adding an appearing window to filter the matches before plotting (Figure 22). It was a necessary step due to the great number of matches to display, as too many entities on one plot would disturb the understanding, and was desirable in order satisfy the user requirements to analyze specific matches on demand.



Select matches to analyse

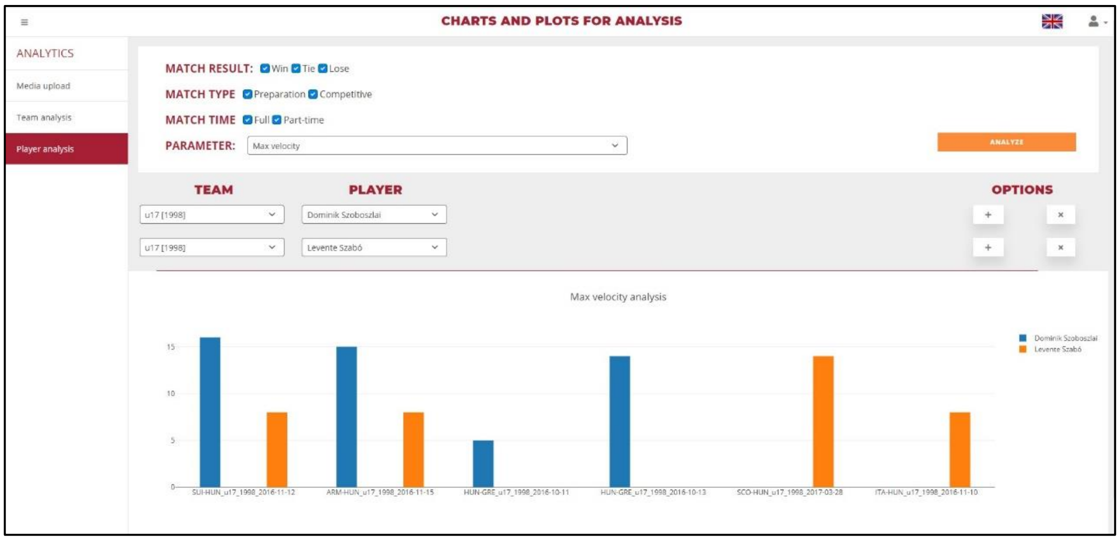
MATYAS TAJTI MÉRKŐZÉSEI
- ✓ SRB-HUN_u17_1998_2016-04-19

DOMINIK SZOBOSZLAI MÉRKŐZÉSEI
- ✓ SRB-HUN_u17_1998_2016-04-19
- ✓ SUI-HUN_u17_1998_2016-11-12
- ✓ ARM-HUN_u17_1998_2016-11-15
- ✓ ITA-HUN_u17_1998_2016-11-10
- ✓ HUN-GRE_u17_1998_2016-10-11
- ✓ HUN-GRE_u17_1998_2016-10-13

BALAZS BESE MÉRKŐZÉSEI
- ✓ SRB-HUN_u17_1998_2016-04-19
- ✓ SUI-HUN_u17_1998_2016-11-12
- ✓ ARM-HUN_u17_1998_2016-11-15
- ✓ ITA-HUN_u17_1998_2016-11-10
- ✓ HUN-GRE_u17_1998_2016-10-11
- ✓ HUN-GRE_u17_1998_2016-10-13
- ✓ SUI-HUN_u17_1998_2016-11-12
- ✓ SCO-HUN_u17_1998_2017-03-28
- ✓ AUT-HUN_u17_1998_2017-03-24
- ✓ ARM-HUN_u17_1998_2016-11-15
- ✓ ITA-HUN_u17_1998_2016-11-10

Figure 22 - Match selection before creating plot on Team analysis (Source: own source)

However the most important UX enhancing function was the visualization itself. Data on plots are categorized by the top modules filtering and are separated by colors. What is more, the added functionality of the JavaScript library, Plotly.JS itself: it lets the user to crop, select, zoom in or out as much as possible in the plot, or even export the chart into raster picture format.


CHARTS AND PLOTS FOR ANALYSIS

MATCH RESULT: ✓ Win ✓ Tie ✓ Lose

MATCH TYPE ✓ Preparation ✓ Competitive

MATCH TIME ✓ Full ✓ Part-time

PARAMETER: Max velocity

TEAM	PLAYER
u17 [1998]	Dominik Szoboszlai
u17 [1998]	Levente Szabó

≡
ANALYTICS
Media upload
Team analysis
Player analysis
ANALYZE
OPTIONS
+
×
Max velocity analysis
15
10
5
0
SUI-HUN_u17_1998_2016-11-12
ARM-HUN_u17_1998_2016-11-15
HUN-GRE_u17_1998_2016-10-11
HUN-GRE_u17_1998_2016-10-13
SCO-HUN_u17_1998_2017-03-28
ITA-HUN_u17_1998_2016-11-10
Dominik Szoboszlai
Levente Szabó

Figure 23 – Generated data comparing plot on player analysis (Source: own source)

7. Evaluating solution

7.1. Evaluating development

In Section 4.1, Figure 8 showed the initial state of the platform's structure. In Figure 24, I present the new platform, containing all the updates and modifications that were described in the previous sections.

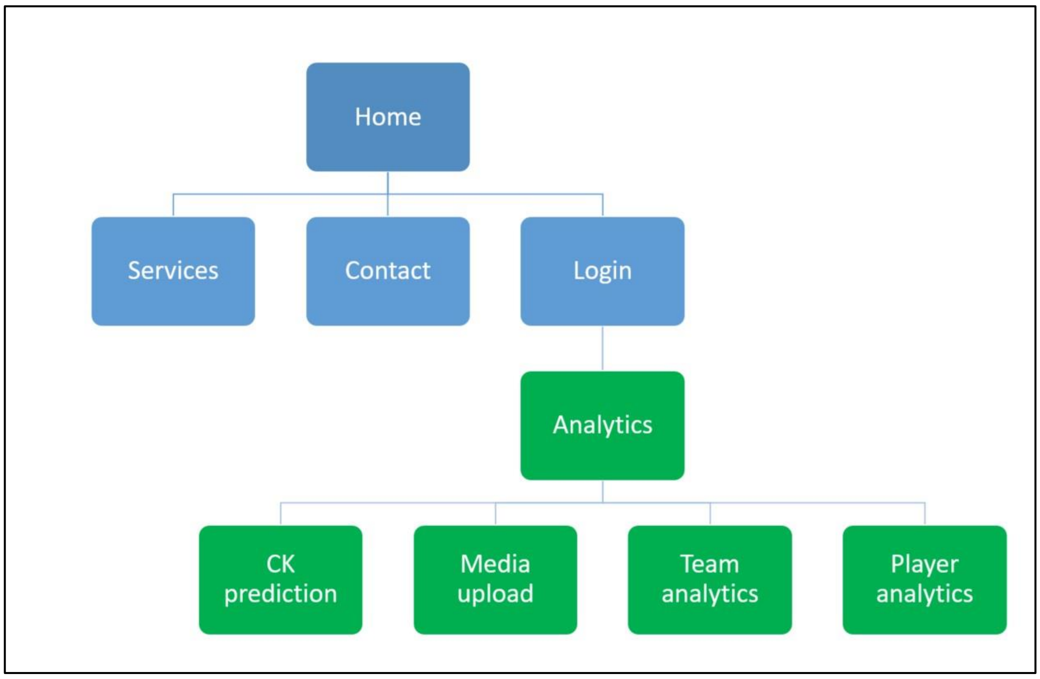


Figure 24 – Final platform construction (Source: own source)

In spite of the fact that feedbacks from users and coworkers found the website useful and practical, it is always an indispensable step to review and evaluate the final result at the end of the process. Therefore I compared the final website to the original goals (declared in section 2) in Table 2.

Goals	Solution	Evaluation result
users can store their data on the website	Created new site for file upload. Users can upload multiple files simultaneously.	Accomplished
users are able to access their uploaded data	Both team and player data is accessible for listing. Video player is not implemented yet.	Mostly accomplished
users can analyze uploaded data on demand	Team and player plots are generated on demand, any time	Accomplished
English-Hungarian translation	Most important texts like headers, parameters and titles are translated. A few text needed JS solution, there are few missing translations.	Mostly accomplished
data visualization for analytics	Third-party JS library implementation, with several user-helping functions	Accomplished

Table 2 – Evaluating website through original requirements (Source: own source)

As described, the webpage meets the most important given criteria almost in every area, therefore we can accept the solution as *mostly accomplished* for deployment. However there are obvious room for further development, as I point it out in the next section.

8. Further possibilities of the study

I would like to emphasize that there are plenty of options to develop this platform further. Although it meets the most important criteria for every-day functioning as well as it is already in use, yet it is clear that improvement can be done both in technological and UX (user experience) ways, both on front-end and backend.

On the front-end, translation and video player solutions are not fully accomplished, I would start with them first. Then I would move on to deeper and more sophisticated ways of data visualizations, broadening the spectrum of plots (currently the only option is the bar chart), adding more analytical and statistical options like minimum-maximum, standard deviation or weighted average.

On the backend, it is important to note that the current website only uses basic functionality, regarding data only. There are several missing solutions for UX that were necessary for a fully functioning page, like search engine, admin and profile settings, and advanced ways for authorization and site access.

All things considered, the website already works and completes the current user demands, however there are several ways to make the solution more complex and useful in the future.

9. Summary

Importance of data analytics

I presented why data analytics is crucial in the field of football, what are the data analytical trends in professional football and why all this topic is important in Hungarian football.

Goals of the study

I summarized the expectations coming from the business side in two main goals:

1) *Update and modify the original web service platform, in order to enable trainers and managers to store, access and analyze their own data real-time*
2) *Improve analytical effectiveness in the website with data visualization and translation*

Then I created a roadmap for both goals to indicate what logical steps were meant to be taken to reach these goals.

Presenting data and used technologies

In Section 3, I discussed the source of data (Catapult) and the two part of used technologies in the study: front-end and backend technologies. I also described how this two side of a platform connect to each other.

Planning development

For planning development, firstly I presented the initial structure of the website. Secondly I determined the points where modifications and updates needed both from front-end and backend related viewpoints.

Platform development

With related code pieces in Section 5 I described the development steps I took. Firstly presented the front-end modifications than the connecting backend updates.

Implementation

I walked through all the sites that were created during this study and the functionalities which were implemented, illustrated by screenshots from the original website.

Evaluating results

I considered in section 7 all the original goals in the light of the results, than evaluated the solution from bullet-point to bullet-point. As a summary, I evaluated the solution as *mostly accomplished*, because there were a few minor things that were not be achieved a hundred percent.

Further possibilities

I observed the possibilities of further improvements. I found that both the front-end and the backend can be improved with further work, however the website already meets the most important criteria for real-life usability.

Results of the study

The two main goals of this research were:

3) *Update and modify the original web service platform, in order to enable trainers and managers to store, access and analyze their own data on demand*
4) *Improve analytical effectiveness in the website with data visualization and translation*

Given the above study, I concluded the following results:

1. The recent version of the platform is able to store different kind of user data (e.g. videos and csv files), and it is possible to access and analyze those data on demand
2. With Hungarian and English translation, and basic data visualization the analytical effectiveness of the website greatly improved, resulting in a usable website with great potential.

10. Acknowledgements

I would like thank to Alija Pašić PhD, László Toka PhD, Gergely Dobreff and Márk Szalay for their various help during my Project Laboratory 2 work.

11. References and figures

11.1. References

Django Software Foundation (2015-2019) *Django 2.2 Documentation. https://docs.djangoproject.com/en/2.2/ [Downloaded 10 November 2019]*

Fried. G. – Mumcu. C (2017). *Sport Analytics. A data-driven approach to sport business and management.* Routledge. New York. 2017.

Fayyad, U. - Grinstein, G. G. – Ward, O. M. (2002) *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers 2002.

The PostgreSQL Global Development Group (1996-2019). *PostgreSQL Documentation https://www.postgresql.org/docs/ [Downloaded 10 November 2019]*

Plotly (2019) *Plotly JavaScript Open Source Graphing Library Documentation https://github.com/plotly/documentation/tree/source-design-merge/_posts/plotly_js/ [Downloaded 10 November 2019]*

The Guardian (2019). *Marcelo Bielsa admits Leeds have spied on every opponent this season. https://www.theguardian.com/football/2019/jan/16/marcelo-bielsa-leeds-spied-every-opponent [Downloaded: 9 May 2019]*

The jQuery Foundation (2019) *jQuery API Documentation https://api.jquery.com/ [Downloaded 10 November 2019]*

11.2. List of figures and tables

Figures

Tables