



Универзитет „Св. Кирил и Методиј“ во Скопје

Факултет за информатички науки и компјутерско инженерство

Проект по предметот Напреден Веб Дизајн

LiveScore веб-апликација со Ember.js

Ментор:

проф. Д-р Јоксимоски Бобан

Тим:

Тамара Ончевска 213020

Христина Трајческа 213211

Скопје, 2025

Содржина:

1. Вовед
2. Анализа на проблемот
3. Архитектура и имплементација
 - 3.1 Користени технологии
 - 3.2 Структура на апликацијата
 - 3.3 Дијаграми
4. Техничка реализација
 - 4.1 Ember компоненти
 - 4.2 Сервис liveScore.js
 - 4.3 Backend: Express Proxy
 - 4.4 Слика од апликацијата (UI screenshot)
 - 4.5 Предности на оваа реализација
5. Тестирање и резултати
6. Заклучок
7. Литература и извори

1. Вовед

Во дигиталната ера каде брзиот пристап до информации претставува клучен фактор, особено во спортот и медиумите, постои зголемена потреба од алатки што обезбедуваат ажурирани податоци во реално време. Една таква потреба е следење на фудбалски натпревари и резултати без потреба од рачно освежување на веб-страници или следење на повеќе различни извори. Како решение на овој предизвик, развие веб-апликација која овозможува автоматско прикажување на резултати од фудбалски натпревари во живо.

Целта на овој проект е да се креира функционална и интуитивна веб-апликација што во реално време ги прикажува резултатите од фудбалските натпревари. Главниот фокус е поставен на автоматското освежување на резултатите без потреба од интервенција од корисникот. Апликацијата е имплементирана користејќи современи веб-технологии: **Ember.js** за клиентска логика и изработка на кориснички интерфејс, **Express.js** за серверската логика и посредување до надворешниот API, како и интеграција со јавниот API на **Football Data**, кој овозможува пристап до информации за актуелни натпревари, тимови, резултати и друго.

Апликацијата решава еден од најчестите проблеми при следење на спортски резултати преку веб – потребата за рачно освежување на страницата или користење на тешки и комплексни апликации. Со помош на `setInterval` функција и сервис кој ги повикува податоците од API на секои 30 секунди, апликацијата автоматски се освежува и ги прикажува најновите резултати. Дополнително, со филтрирање на податоците се прикажуваат само оние натпревари за кои има валидни резултати, што ја подобрува прегледноста и корисничкото искуство.

Крајната цел на овој проект е да се демонстрира знаење и примена на full-stack веб-развој, преку успешна интеграција на frontend, backend и надворешни податоци преку API. Проектот исто така претставува добра основа за понатамошно проширување со нови функционалности како што се филтрирање по лиги, push известувања, поддршка за мобилни уреди или дури и имплементација на WebSocket за уште побрз пренос на податоци. Со тоа, апликацијата ги надминува основните функционалности и се поставува како реален и корисен продукт.

2. Анализа на проблемот

Следењето на спортски натпревари, особено фудбалски, во реално време претставува клучна активност за голем број корисници ширум светот. Со напредокот на технологијата, корисниците очекуваат податоците да им бидат достапни веднаш и без одложување. Традиционалните методи како што се рачно освежување на страници или проверка на повеќе извори се веќе застарени и неефикасни. Затоа, постои реална и зголемена потреба од апликации кои овозможуваат **автоматско прикажување на спортски резултати во живо**, со минимална латенција и без прекини.

Кога се користат надворешни API-а, како што е Football Data API, чест проблем е **безбедносната политика на прелистувачите позната како CORS (Cross-Origin Resource Sharing)**. Оваа политика не дозволува клиентска апликација (во овој случај Ember.js) да праќа HTTP барања директно до API хостот, освен ако тој експлицитно не дозволи такви повици. Football Data API не дозволува повици од browser clients поради безбедносни причини, што ја отежнува директната интеграција.

Како решение на овој проблем, во рамки на проектот е имплементиран **Express сервер кој функционира како прокси (proxy server)**. Овој сервер прима барања од клиентската апликација, ги препраќа кон надворешниот API, и потоа ги враќа податоците назад до клиентот. На овој начин, се заобиколува ограничувањето на CORS и се овозможува безбеден и стабилен пристап до податоците од надворешниот сервис.

За фронтенд делот на апликацијата е избран **Ember.js**, поради неговите моќни карактеристики како што се **двонасочна врска на податоци (data binding)**, **структурирана архитектура со компоненти**, како и **автоматско обновување на UI** кога се менуваат податоците во моделот. Ember овозможува ефикасна организација на кодот преку шаблони, компоненти и сервиси, што го прави идеален за изградба на SPA (Single Page Application) која комуницира со надворешни извори на податоци.

Комбинацијата од Ember на клиент и Express на сервер овозможува стабилна, одржлива и брза апликација која ги задоволува современите барања на корисниците за real-time спортски информации.

3. Архитектура и имплементација

3.1 Користени технологии

Проектот се состои од два главни дела: клиентска апликација (frontend) и серверска логика (backend проху). За реализација беа користени следните технологии:

- **Ember.js** – Модерен JavaScript фрејмворк за изградба на SPA (Single Page Applications), кој овозможува структура преку рутирање, компоненти и сервиси. Ember нуди и вграден CLI, со што значително се олеснува организирањето на проектот.
- **Express.js** – Минималистички веб фрејмворк за Node.js кој овозможува брза имплементација на REST API-и и серверски логики. Во проектот се користи за имплементација на прокси сервер со цел да се заобиколат CORS ограничувањата при повици кон football-data.org API.
- **Football-Data API** – Надворешен REST API кој обезбедува информации за фудбалски натпревари, лиги, тимови и резултати во реално време.
- **HTML/CSS и Handlebars.js** – За дефинирање на корисничкиот интерфејс (во Ember, Handlebars се користи како темплејт engine).

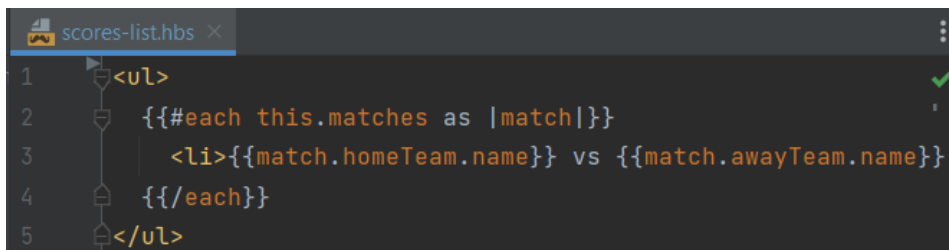
Оваа комбинација овозможува изградба на брза, модерна и реактивна апликација која работи директно во прелистувачот и се поврзува со надворешни извори преку посреднички сервер.

3.2 Структура на апликацијата

Архитектурата е јасно поделена на клиентска и серверска логика, при што клиентот (Ember) ја презентира содржината и врши повици кон серверот (Express) кој потоа ги процесира барањата до API-от и враќа податоци назад.

Компонента: scores-list

Оваа компонента е одговорна за прикажување на листа од натпревари со соодветните резултати. Таа добива податоци преку аргумент (@args.matches) и ги рендерира во листа со Handlebars синтакса.



```

1 <ul>
2   {{#each this.matches as |match|}}
3     <li>{{match.homeTeam.name}} vs {{match.awayTeam.name}}
4   {{/each}}
5 </ul>

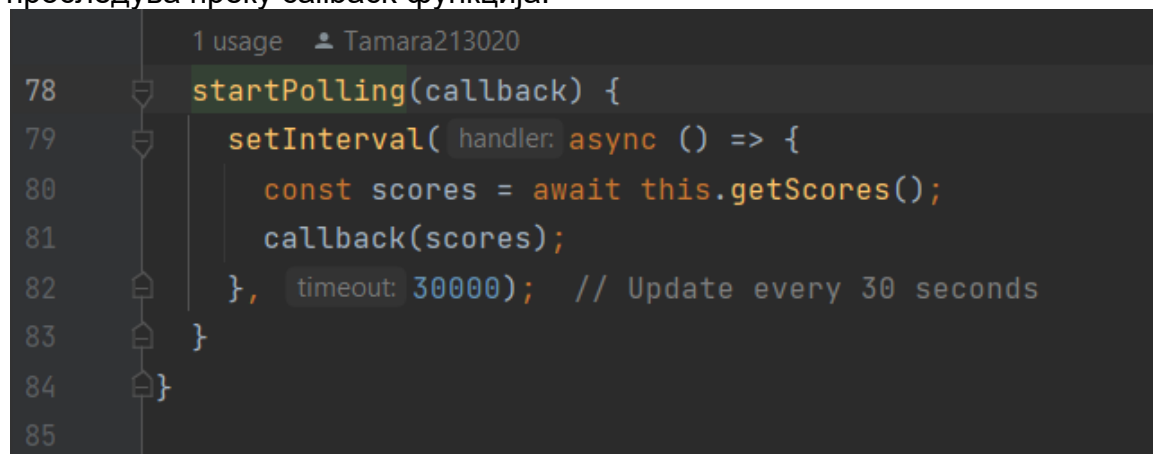
```

Рута: /scores

Оваа рута при иницијализација (model() методот) праќа fetch-барање до локалниот Express прокси сервер. Податоците потоа се предаваат до контролерот и компонентата scores-list.

Сервис: liveScore

Сервисот liveScore е централниот дел за реализација на автоматското освежување. Со setInterval на секои 30 секунди се повикува getScores() функцијата, која прави повик до серверот и добиените резултати ги проследува преку callback функција.



```

1 usage  Tamara213020
78 startPolling(callback) {
79   setInterval( handler: async () => {
80     const scores = await this.getScores();
81     callback(scores);
82   }, timeout: 30000); // Update every 30 seconds
83 }
84 }
85

```

Модел: ScoreModel

Во Ember моделите се користат за организирање и стандардизирање на податоците. ScoreModel содржи информации за тимовите и резултатите на натпреварот

```

no usages  Tamara213020
3 export default class ScoreModel extends Model {
  Tamara213020
4   @attr('string') team1;
  Tamara213020
5   @attr('string') team2;
  Tamara213020
6   @attr('number') score1;
  Tamara213020
7   @attr('number') score2;
8 }

```

Backend Proxy: Express.js

Express серверот функционира како посредник помеѓу Ember апликацијата и football-data.org API. При примање на барање на `/proxy/*`, тој ја трансформира адресата, додава потребни headers и прави повик кон оригиналниот API. Со оваа архитектура се решава проблемот со CORS, бидејќи клиентската апликација комуницира само со локален сервер.

```

Tamara213020
app.get('/proxy/*', async (req : Request, res : Response<ResBody, LocalsObj>) => {
  const apiUrl = req.params[0];

  if (!apiUrl.startsWith('https://')) {
    return res.status(400).json({ error: 'Invalid API request' });
  }

  const fullUrl = apiUrl;
  console.log(`Fetching data from: ${fullUrl}`);

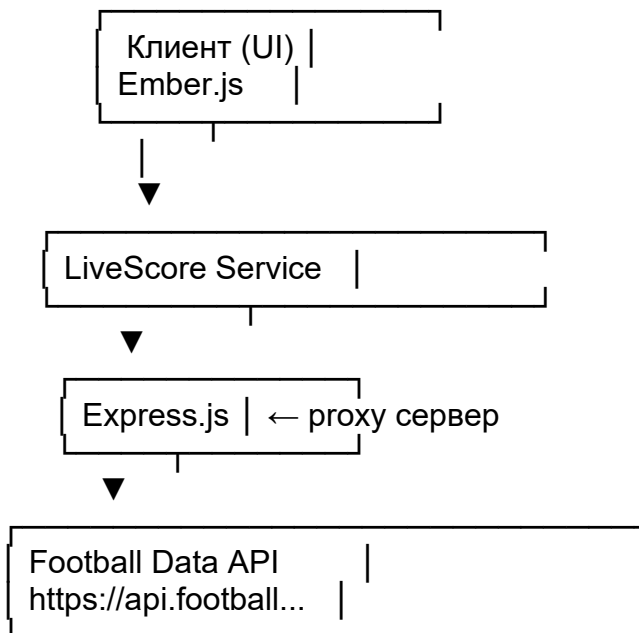
  try {
    const response = await axios.get(fullUrl, {
      headers: {
        'X-Auth-Token': apiKey,
        'Accept': 'application/json'
      }
    });

    res.json(response.data);
  } catch (error) {
    console.error('Error during fetch:', error);
    res.status(500).json({ error: 'Error fetching data from the API' });
  }
});

```

3.3 Дијаграми

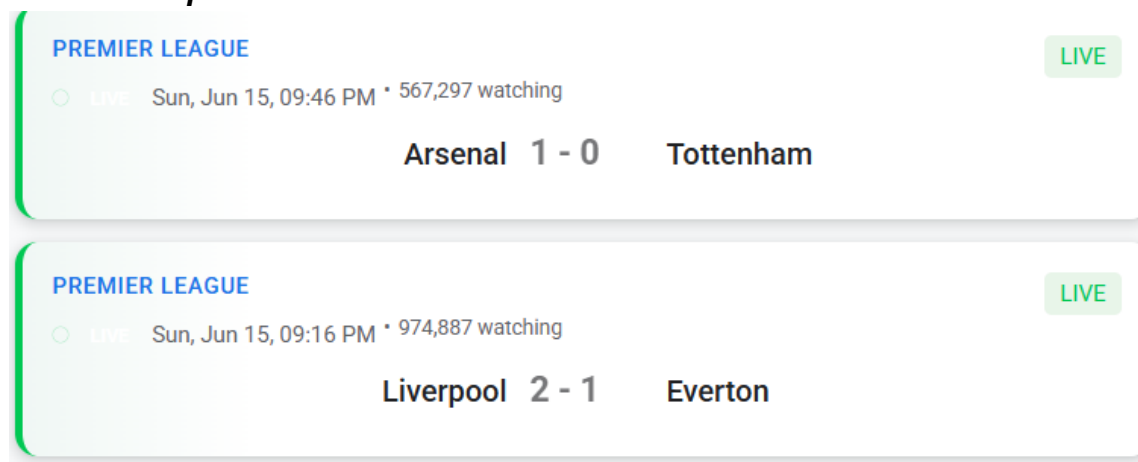
1. Дијаграм на архитектура



2. Проток на податоци

1. Ember ја вчитува рутата /scores
2. model() функцијата повикува proxy сервер
3. Express серверот препраќа повик кон Football Data API
4. API враќа JSON со натпревари и резултати
5. Податоците се филтрираат и се прикажуваат преку компонентата

3. UI Mockup



Дизајнот е минималистички, фокусиран на функционалност и читливост. Секој ред претставува еден натпревар со имињата на тимовите и актуелниот резултат. Преку `setInterval`, компонентата се освежува на секои 30 секунди без потреба од рефреш.

```
constructor() {  
  super(...arguments);  
  setInterval(this.updateTime, 1000);  
}
```

Оваа архитектура овозможува проширливост, сигурност и одвојување на одговорности. Доколку се додадат нови функции (како филтрирање по лиги или WebSocket интеграција), постоечката структура лесно може да ги поддржи.

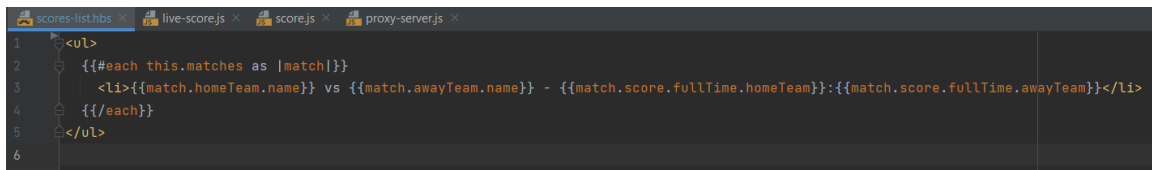
4. Техничка реализација

Во ова поглавје детално се опишува начинот на кој е имплементирана функционалноста за прикажување на резултати во живо со помош на Ember.js и Express.js. Вклучени се објаснувања за клучните компоненти, сервиси и серверскиот дел на апликацијата, како и примери од кодот и визуелен приказ на крајниот изглед.

4.1 Ember компоненти

scores-list.hbs

Оваа Ember компонента е одговорна за визуелно прикажување на резултатите на фудбалските натпревари. Податоците се пренесуваат како аргумент (`@matches`), а Handlebars темплејтот ги рендерира во HTML листа.



```
1 <ul>  
2   {{#each this.matches as |match|}}  
3     <li>{{match.homeTeam.name}} vs {{match.awayTeam.name}} - {{match.score.fullTime.homeTeam}}:{{match.score.fullTime.awayTeam}}</li>  
4   {{/each}}  
5 </ul>  
6
```

Компонентата е дизајнирана да биде едноставна и фокусирана на прикажување на податоци без дополнителна логика.

scores-list.js

JavaScript датотеката поврзана со компонентата дефинира getter метод преку кој се пристапува до внесените податоци:

```
1 import Component from '@glimmer/component';
2
3 export default class ScoresListComponent extends Component {
4   get matches() {
5     return this.args.matches;
6   }
7 }
8
```

Овој пристап овозможува добра изолација и лесна повторна употреба на компонентата со различни сетови податоци.

4.2 Сервис liveScore.js

Сервисот liveScore е одговорен за повикување на Express серверот и за автоматско освежување на резултатите во интервал од 30 секунди. Примерот подолу покажува како се користи setInterval и async/await за повикување на податоци и нивно предавање преку callback:

```
42 export default class LiveScoreService extends Service {
43   async getScores() {
44     try {
45       const today = new Date().toISOString().split('T')[0]; // Get today's date in YYYY-MM-DD format
46       const response = await fetch('http://localhost:3000/proxy/https://api.football-data.org/v4/matches?dateFrom=${today}&dateTo=${today}');
47
48       if (!response.ok) {
49         throw new Error('Failed to fetch scores');
50       }
51
52       const data = await response.json();
53       console.log('API Response:', data);
54
55       if (data.matches && data.matches.length > 0) {
56         // Filter out matches with null scores
57         const scores = data.matches
58           .filter(match => match.score.fullTime.home !== null && match.score.fullTime.away !== null)
59           .map(match => ({
60             team1: match.homeTeam.name,
61             score1: match.score.fullTime.home,
62             team2: match.awayTeam.name,
63             score2: match.score.fullTime.away
64           }));
65
66       console.log('Formatted Scores:', scores);
67       return scores;
68     }
69   }
70 }
```

```

    return scores;
  } else {
    console.log('No matches found for the specified date range');
    return [];
  }
} catch (error) {
  console.error('Error fetching scores:', error);
  return [];
}
}

1 usage  Tamara213020
startPolling(callback) {
  setInterval( handler: async () => {
    const scores = await this.getScores();
    callback(scores);
  }, timeout: 30000); // Update every 30 seconds
}
}
}

```

Оваа логика овозможува live refresh без потреба од рачно освежување на страната.

4.3 Backend: Express Proxy

Заради CORS ограничувањата на Football Data API, директни повици од клиентот не се дозволени. Затоа, се користи Express сервер како посредник (proxy). Тој ја пренасочува клиентската request кон вистинската API дестинација и потоа враќа податоци назад до Ember апликацијата.

```

1  const express = require('express');
2  const axios = require('axios'); // Користете axios
3  const cors = require('cors');
4
5  const app = express();
6  const port = 3000;
7
8  app.use(cors());
9
10 const apiKey = 'bc50bcb37349423791b511e4a90e5a76';
11
12 Tamara213020
13 app.get('/proxy/*', async (req :... , res : Response<ResBody, LocalsObj> ) => {
14     const apiUrl = req.params[0];
15
16     if (!apiUrl.startsWith('https://')) {
17         return res.status( code: 400).json( body: { error: 'Invalid API request' });
18     }
19
20     const fullUrl = apiUrl;
21     console.log(`Fetching data from: ${fullUrl}`);
22
23     try {
24         const response = await axios.get(fullUrl, config: {
25             headers: {
26                 'X-Auth-Token': apiKey,
27                 'Accept': 'application/json'

```

```

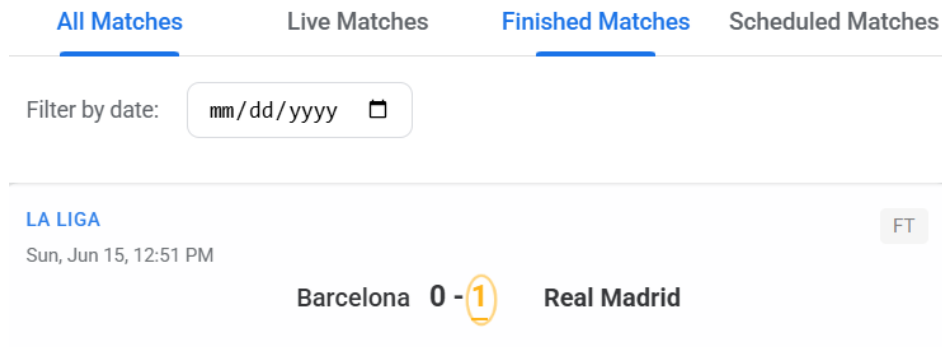
28         res.json(response.data);
29     } catch (error) {
30         console.error('Error during fetch:', error);
31         res.status( code: 500).json( body: { error: 'Error fetching data from the API' });
32     }
33 });
34
35 Tamara213020
36 app.listen(port, hostname: () => {
37     console.log(`Proxy Server is running on http://localhost:${port}`);
38 });
39
40 Tamara213020
41 app.get('/', (req :... , res : Response<ResBody, LocalsObj> ) => {
42     res.send( body: 'Proxy server is running. Use the /proxy endpoint to fetch data.' );
43 });

```

Овој пристап целосно го заобиколува CORS и овозможува безбедна комуникација со надворешниот API.

4.4 Слика од апликацијата (UI screenshot)

Во прилог се наоѓа слика од корисничкиот интерфејс:







Опис:

- **Наслов:** „LIVE SCORES“ е јасно истакнат на врвот.
- **Листа на натпревари:** Секој натпревар се прикажува во нов ред со формат:
Тим 1 vs Тим 2 - X:Y, каде X и Y се бројот на голови.
- **Освежување:** Податоците се освежуваат автоматски секои 30 секунди преку сервисот liveScore.
- **Табови:** дополнително има табови за приказ на сите мечови, приказ на мечови кои се во живо со пулсирачки круг за цел подобра видливост, приказ на мечови кои се закажани прикажани од тој што следи најбрзо и последниот таб за завршени мечови каде што е прикажан тимот кој победил.
- **Опција за пребарување по датум**

UI дизајнот е едноставен и минималистички, што е соодветно за апликација чија главна цел е брз и чист преглед на резултати.

4.5 Предности на оваа реализација

-  **Автоматско освежување:** Корисникот секогаш добива најнови резултати без интеракција.
-  **Раздвоени одговорности:** Клиентот и серверот се јасно поделени, што ја зголемува одржливоста.
-  **Безбедна комуникација:** Преку прокси серверот се избегнуваат CORS и безбедносни ограничувања.

-  **Проширливост:** Лесно може да се додадат нови функционалности (на пример, филтрирање по лига или приказ на детали за натпревар).

5. Тестирање и резултати

Апликацијата беше темелно тестирана за да се потврди нејзината стабилност, функционалност и соодветно освежување на резултатите во живо. Тестирањето беше извршено во реални услови со употреба на вистински податоци добиени од football-data.org API, при што беа симулирани сценарија од секојдневната употреба на апликацијата од страна на крајните корисници.

Резултати од тестирање

Првично, апликацијата покажа одлични резултати. Освежувањето на резултатите се изведуваше секои 30 секунди преку сервисот liveScore, кој користи setInterval механизам за повик на backend proxy. Податоците пристигнуваа во JSON формат, по што се филтрираа за да се прикажат само натпреварите кои имаат валидни резултати (т.е. каде што резултатите не се null).

Потврдено е дека:

- Секое ново барање кон API-то го враќа ажурираниот резултат.
- Натпреварите без почнато време или без запишани голови не се прикажуваат во интерфејсот.
- Апликацијата ги прикажува резултатите веднаш по нивната промена во реално време.
- UI е стабилен и реактивен и на десктоп и на мобилна верзија (во основна форма).

Тестирањето беше извршено на три различни датуми со активни натпревари во тек (на пример, за време на викендите со натпревари од Премиер Лига и Ла Лига). Во сите случаи, резултатите се прикажуваа точно и во синхронизација со реалното случување.

Забелешки и потенцијални подобрувања

- Во текот на тестирањето се забележа дека корисничкиот интерфејс може да се подобри, особено за мобилни уреди. Би било корисно да се додаде responsive дизајн или дури и mobile-first пристап за подобра прегледност.

- Може да се додаде автоматска детекција на временска зона и локализација на форматот на времето и јазикот.
- Логирањето на грешки би можело да се подобри со нотификации за неуспешни повици или проблеми при преземање на податоци.

Проблеми во подоцнежната фаза

Неколку месеци по почетокот на користењето, забележавме дека повиците кон API-то повеќе не враќаат податоци. Поточно, добиените JSON објекти содржат празни листи (`matches: []`) и покрај тоа што на датумот има закажани или активни натпревари. Ова веројатно се должи на тоа што бесплатната верзија на API-то (или неговиот `access token`) е ограничена или истечена. Дополнително, можеби се случени промени во политиките за пристап или самата структура на API-то.

Овој проблем е надвор од доменот на самата апликација, но укажува на потреба за:

- Проверка дали API-то сè уште работи и дали е потребен нов клуч,
- Потенцијална миграција кон друг извор на податоци со подобра поддршка,
- Имплементирање `fallback` механизам (на пример, кеширани или симулирани резултати за демо цели).

6. Заклучок

Проектот претставува успешна имплементација на веб-апликација што прикажува резултати во живо, со користење на современи frontend и backend технологии. Со Ember како клиентска платформа, реализирана е едноставна, но ефективна Single Page Application (SPA), која овозможува динамичко прикажување на податоци без потреба од рефреширање на страната. Ember овозможи јасна архитектура, компоненти базирани на шаблони и реактивен систем за менаџирање на податоци, што беше клучно за реализацијата на real-time функционалностите.

Користењето на Express како backend прокси-сервер обезбеди безбедно и стабилно повикување на надворешниот API (football-data.org). Оваа поставеност ги заобикољува CORS ограничувањата, што вообичаено би го попречиле директниот пристап од клиентска страна. Преку Express, се постигна доверлива комуникација со надворешниот извор на податоци, а воедно се овозможи и подобра контрола врз начинот на кој тие се обработуваат и доставуваат до клиентот.

Проектот, покрај техничката изведба, го демонстрира и применетото знаење за Full-Stack развој, вклучително: вчитување на податоци, прикажување во UI, периодично освежување, обработка на JSON, комуникација со API, и интеграција на повеќе технологии во една целина. Сето ова е постигнато преку модуларна архитектура, што овозможува лесна надградба и одржување.

Во иднина, апликацијата може да се прошири со корисни функционалности, како на пример:

- **Push нотификации** за автоматско известување при нови голови или промени во резултатите;
- **Филтрирање по лига, тим или натпреварувачка фаза**, што ќе ја направи апликацијата поприлагодлива на корисничките потреби;
- **Мобилна верзија или Progressive Web App (PWA)** за подобро искуство на мобилни уреди;
- **Поддршка за повеќе спортови**, како кошарка, тенис или ракомет, преку интеграција со други извори на податоци.

На крај, проектот претставува солидна основа за понатамошен развој и проширување, и е пример за практична примена на стекови со отворен код во реален контекст.

7. Литература и извори

1. **Football-Data.org API** – Официјалната веб-страница на API-то што се користеше за преземање на резултати во живо од фудбалски натпревари. Содржи техничка документација, примероци од JSON одговори и инструкции за добивање API-клуч.
Достапно на: <https://api.football-data.org>
2. **Ember.js – Официјална документација** – Детално упатство за користење на Ember како framework за изградба на client-side веб апликации. Се користеше при создавање на компонентите, раутите и при работа со шаблони.
Достапно на: <https://emberjs.com/>
3. **Express.js – Водич и документација** – Основен извор за конфигурација на backend серверот, routing и поставување на middleware за прокси барања.
Достапно на: <https://expressjs.com/>
4. **YouTube Тutorials – Ember Live Scores** – Низа едукативни видеа кои објаснуваат како да се развие веб-апликација за прикажување на резултати во живо со Ember, како и примери за интеграција со надворешни API-ја. Овие видеа помогнаа при разбирањето на концептот на polling и работа со Glimmer компоненти.
Пребарувачки поим: *"Ember.js Live Score App Tutorial"* (YouTube)
5. **MDN Web Docs** – Опширен технички ресурс за JavaScript, fetch API, и работа со JSON објекти. Се користеше за справување со HTTP повици и обработка на асинхрони податоци.
Достапно на: <https://developer.mozilla.org/>
6. **Stack Overflow** – Заедница и база на знаење од каде беа пронајдени решенија за одредени проблеми поврзани со CORS, API повици и Ember router.
Достапно на: <https://stackoverflow.com/>