Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Aplikácia na sledovanie traťových závodov

Bakalárska práca

2024
Lukáš Cauner

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Aplikácia na sledovanie traťových závodov

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Mgr. Peter Náther, PhD.

Bratislava, 2024
Lukáš Cauner

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lukáš Cauner
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Aplikácia na sledovanie traťových závodov
Live tracking of trail races

Anotácia: Je dnes takmer povinnosťou každých väčších pretekov, že umožnujú sledovanie závodu online na internete. Okrem priamych prenosov sa najčastejšie webové aplikácie, ktoré zobrazujú aktuálnu pozíciu pretekárov a sú pravidelne aktualizované. Oproti klasickému videozáznamu dokážu zobraziť množstvo doplnkových informácií.
Dnešné techológie RIA umožňujú prinášať upraviť výstup aplikácie podľa požiadaviek klientov.
Úlohou bakalára bude implementovať aplikáciu možňujúcu sledovaťpreteky odohrávajúce sa trati, spravidla vyznačenej v mape, predpokladá sa že pôjde o preteky na väčšiu vzdialenosť, kde je možné, že jedinou informáciou budú aktuálne súradnice pretekárov.
Nutnosťou je analyzovať potrebné a možné (v súčasnosti organizátormi bežne získateľné) vstupy, definovanie service API aplikácie, ktoré umožní organizátorom pretekov využitie takejto aplikácie.
Jedným z problémov ,ktoré táto práca kladie bude identifikácia pozície pretekárov na základe ich polohy.
Samotná web aplikácia by mala umožňiť modifikáciu náhľadu na preteky, napr. sledovanie konkrétneho pretekára, vybraného úseku trate, sledovanie preketárov podľa pozície, či prípadná notifikácia možných udalostí.

Cieľ: Vytvorenie webovej aplikácie pre online sledovanie závodov a pozícii pretekárov
pri pretekoch sledovateľných na mape, teda s dlhými, resp. nie striktne vymedzenými traťami.
Aplikácia umožní vyhľadávať a filtrovať pretekárov,
pohyb v mape, či prehratie skončeného závodu.

Vedúci: Mgr. Peter Náther, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 27.10.2014

Dátum schválenia: 19.09.2022 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

....

študent vedúci práce

Poďakovanie: Rád by som sa poďakoval svojmu školitelovi za ochotu, podporu a pomoc pri tvorbe tejto práce.

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Zoznam obrázkov

Úvod

Preteky rôzneho druhu patria k významnej časti športových udalostí, či už ide o cyklistické preteky, motoršport alebo iné. Vedia prinieť množstvo adrenalínu nie len samotným pretekárom, ale aj divákom. Našou prácou sa budeme snažiť vytvoriť vhodnú aplikáciu pre ogranizátorov malých ale aj veľkých pretekov, možnosť zdielať polohu pretekárov, a tak divákom ponúknuť príležitosť sledovať ich online a aby mali aktuálny prehľad o dianí na trati.

V úvodnej kapitole 1 si priblížime problematiku práce a pozrieme sa na už exitujúce riešenia.

V kapitole 2 uvedieme ciele a metodiku práce a v kapitole 3 sa ďalej zameriame na používateľské scenáre.

V nasledujúcej kapitole 4 si predstavíme technológie, ktoré sme využívali pri implementácii a samotný návrh aplikácie.

V kapitole 5 si detailne rozoberieme implementáciu. Budeme sa zaoberať programovou časťou a funkcionalitou pre všetky scenáre. Budeme sa zaoberať kompletnou implementáciou a použitým algoritmom vo frontendovej i backendovej časti. Taktiež si predstavíme databázu a prácou s ňou.

V záverečnej kapitole sa pozrieme na to, ako sme našu aplikáciu testovali a na výsledky týchto testov.

Kapitola 1

Uvedenie do problematiky

1.1 Problematiky práce

Základným pojmom našej práce je webová aplikácia. Používateľovi predstavuje tento typ aplikácie pohodlné využívanie z akéhokoľvek zariadenia s prístupom na internet, bez ohľadu na typ, operačný systém a bez nutnosti inštalácie, na rozdieľ od desktopovej alebo mobilnej aplikácie. Preto predstavuje webová aplikácia najvhodnejšie riešenie ako náš softvér ponúknuť používateľovom.

Jednou z hlavých problematík okrem samotného návrhu a vytvorenia aplikácie, tak aby bola vhodná aj pre organizátorov menších závodov, je určenie pozície pretekára na základe jeho polohy. Existujú rôzne spôsoby ako určiť na akej pozícii sa pretekár momentálne nachádza. V motoršporte na svetovej úrovni ako napríklad Formula 1 sa využívajú zložité riešenia, ktoré zahŕňajú rozsiahlu technickú výbavu na využívaných tratiach. Pozíciu neurčujú len na základe GPS dát, ale aj pomocu senzorov a rádiových snímačou v blízkosti trate alebo priamo pod asfaltom. Takéto riešenie nie je vhodné pre menšie preteky, či už z dôvodu veľkých nákladov alebo nemožnosti umiestnenia zariadení pod trať. Určovanie pozície na základe GPS dát využívajú napríklad v Tour de France, avšak podrobnosti o algoritme nie sú verejne známe. V našej práci sme použili vlastný spôsob na určenie pozície pomocou exitujúcej štruktúry pre geografické objekty.

Aby sme zabezpečili divákom pohodlné sledovanie pretekov, naša aplikácia musí vždy zobraziť čo najaktuálnejšie dáta. Preto ďalšou problematikou je získavanie dát od organizátora a okamžitá reakcia aplikácie na zmeny. V neposlednom rade potrebujeme aj efektívne zhromažďovanie dát, aby si divák vedel pozrieť preteky aj zo záznamu.

Naša aplikácia poskytuje kompletné informácie o pretekoch, ktoré organizátor zadá pri vytvorení udalosti. Preto sme sa okrem samotného live prenosu pretekov museli zaoberať aj ich vytvorením v aplikácii, čo zahŕňa najmä pretekárov, dátum a čas konania udalosti a trať, na ktorej sa bude pretekať.

1.2 Existujúce riešenia

Vytvorením aplikácie pre organizátorov pretekov akéhokoľvek typu sa venovali vývojári už pred nami. Existuje zopár aplikácii, ktoré sa venujú podobnej problematike ako naša práca.

Požiadavkam našej práce najviac vyhovovali nasledovné aplikácie:

1. racetracker.no [1]
2. gbracetracker.co.uk [2]

Obe poskytujú služby pre organizátorov pretekov a live zobrazenie pre používateľov. Prvá uvedená je webová aplikácia, ktorá ponúka komplexné služby pre organizátorov vrátane pokytnutia vlastnej techniky pre pretekárov. Nenašli sme však možnosť registrácie pre obyčajného požívateľa, a tak získať možnosť byť notifikovaný o nových pretekoch a výsledkoch. Nie pri každých pretekoch bola dostupná aj mapa a možnosť prehrania z archívu. Druhá uvedená aplikácia taktiež ponúka komplexné služby pre organizátorov vrátane dodaní GPS zariadení. Na webovej stránke sme však nenašli žiadne demo ani reálne ukážky.

Naša práca bude predstavovať novú aplikáciu, ktorá bude spĺňať stanovené ciele, dostupnosť a spomínané funkcionality.

Kapitola 2

Ciele a metodika práce

Hlavným cieľom tejto práce je vytvoriť webovú aplikáciu na sledovanie traťových závodov. Táto aplikácia nebude určená len pre divákov, ale aj pre organizátorov pretekov, ktorí si budú môcť vytvoriť vlatnú udalosť a aplikácia ponúkne divákom prehľad o aktuálnej situácii v týchto pretekoch, ale aj možnosť pozrieť si už skončené preteky z archívu. Ďalším cieľom je predstaviť čitateľovi prolematiku, možnosti, návrh a riešenie našej praktickej časti práce.

Pred implementáciou aplikácie sme si museli prezieť už existujúce riešenia a naštudovať problematiku. Ďalej sme podľa stanovených cieľov vytvorili návrh architektúry a zvážili dostupné frameworky, spôsoby komunikácie, databázu atď. Ďalej sme navhrli dátový model a architektúry samotných častí aplikácie. Pri návrhu sme kládli dôraz na nasledovné aspeky:

- modularita - aplikáciu sme navrhli tak, aby jej moduly boli, čo najviac nezávislé a mali jasne definovanú svoju úlohu
- rozšíriteľnosť - aplikáciu je možné pohodlne rozšíriť a upraviť podľa aktuálnych potrieb
- efektivita - pri programovaní sme využívali, čo najefektívnejšie algoritmi, najmä na výpočet pozície pretekára. Tento výpočet musí byť, čo najefektívnejší, aby bol výsledok okamžite zobrazený používateľovi
- bezpečnosť - pri implementácii sme dbali na bezpečnosť aplikácie, použili sme najnovšie štandardy pri role based autentifikácii, vstupy od používateľa sú overené a ošetrené. Snažili sme sa tak, aby bola aplikácia čo najviac pripravená pre reálne používanie a odolná voči útokom
- vhodná architektúra a členenie kódu - pri návrhu sme sa držali štandardov príslušných frameworkov

Napriek tomu, že zadanie práce bolo vopred známe, testovanie aplikácie sme neriešili nakoniec, ale spolu s vývojom. Dosiahli sme tým vyššiu flexibilitu a kvalitu. Metodológiu sme tak pripodobnili viac agilnému procesu než vodopádovému. Na záver sme podrobili našu aplikáciu testami, ktoré simulujú reálnu prevádzku.

Kapitola 3

Používateľské scenáre

V aplikácii rozlišujeme používateľov do troch kategórii:

- organizátor,
- prihlásený divák,
- návštevník - neprihlásený divák.

3.1 Scenáre pre návštevníka

1. Návštevník si môže prezrieť zoznam udalostí, tento zoznam vie filtrovať podľa dátumu, organizátora, názvu alebo typu - nadchádzajúca, live alebo archivovaná. Vie ho taktiež zoradiť podľa dátumu zostupne alebo vzostupne.
2. Vie sledovať preteky live alebo z archívu.

3.2 Scenáre pre prihláseného diváka

1. Používateľ má možnosť sa registrovať ako divák.
2. Ďalej sa prihlasuje do aplikácie pomocou používateľského mena a hesla, ktoré si zvolil pri registrácii.
3. Prihlásený divák má všetky scenáre ako návštevník.
4. Výhodou prihláseného diváka je možnosť dať odber organizátorovi a tak dostávať emailové notifikácie o:
    - vytvorení udalosti,
    - zmene dátumu prípadne zrušení,

- upozornenie pol hodiny pred štartom,
- notifikácia po skončení pretekov.

3.3 Scenáre pre organizátora

1. Na to aby používateľ vedel byť organizátor, je potrebná registrácia.
2. Ďalej sa organizátor prihlasuje do aplikácie pomocou používateľského mena a hesla, ktoré si zvolil pri registrácii.
3. Pred vytvorením udalosti, je povinný vytvoriť si kolekciu, ktorá pozostáva s jazdcov prípadne tímov.
4. Pre každého jazdca je organizátor povinný vyplniť:
   - meno a priezvisko,
   - číslo - každý jazdec musí mať iné,
   - GPS identifikátor - každý jazdec musí mať iný,
   - ak sa v kolekcii používajú tímy, organizátor je povinný ho zadať každému jazdcovi, tímu sa určí meno a farba,
   - farbu - ak sa používajú tímy, farba jazdca sa určí podľa tímu.
5. Kolekcie vie organizátor vyvtárať, upravovať, mazať a prezrieť si zoznam svojich vlastných.
6. Organizátor má možnosť vytvoriť si zoznam tratí, ktoré má v pláne často používať, a tento zoznam vie upravovať a zobraziť.
7. Vie si vytvoriť vlastnú udalosť, ktorá pozostáva z názvu udalosti, výberu kolekcie zo zoznamu, výberu používanej trate alebo zadaniu novej, zadaniu počtu kôl ak ide o okruh a určenia dátum konania udalosti.
8. Po úspešnom vytvorení udalosti organizátor obdrží email s potrebnými informáciami.
9. Počas pretekov posiela GPS dáta na určenú adresu s obdržaným autentifikačným tokenom.
10. Organizátor si vie prezrieť svoje udalosti.
11. Svoje udalosti vie editovať a zrušiť najneskôr dve hodiny pred ich začiatkom.
12. Organizátor si vie prezrieť svojich odoberateľov.
13. Navyše platia pre neho všetky scenáre ako pre návštevníka.

Kapitola 4

Návrh riešenia

Pred implementáciu našej aplikácie sme museli vytvoriť vhodný návrh aby sme zachovali vlastnosti spomínané v metodike práce. Návrh sme vytvárali spolu s výberom technológii a snažili sme sa zvoliť najoptimálnešiu možnosť. Pri návrhu sme brali do úvahy všetky používateľské scenáre. Najprv sme si vytvorili prototyp modelu aplikácie, ktorý pozostával z troch častí:

1. databáza,
2. backend,
3. frontend.

Nasledoval prieskum dostupných technológii a frameworkov. Začali sme s výberom frameworku pre backend a vyberali sme medzi frameworkami Django (Python), Spring (Java) a .NET (C#). Pri každom sme vzali do úvahy znalosť programovacieho jazyka, na ktorom je daný framework postavený, ako by bolo možné napojiť frontendovú časť aplikácie a práca s databázou. Keďže pri návrhu chceme docieliť čo najväčšiu modularitu, ako backendovú časť sme zvolili WEB API. Tým dosiahneme úplne oddelenie klientskej časti od serverovej, čo predstavuje nie len väčšiu flexibilitu, ale aj bezpečnosť. Po dôkladom zvážení všetkých faktorov sme sa rozhodli pre .NET. Poskytuje vytvorenie databázy aj pre Postgre SQL, ktoré je bezplatné a máme najviac naštudovanú jeho syntax. V .NET je taktiež defaultné zobrazenie endpointov pomocou služby Swagger, ktorý slúži ako rozhranie pre WEB API pre vývojárov.

Pri výbere frontendového frameworku sme sa rozhodovali najmä medzi javascriptovými frameworkami React a Angular. Zamerali sme sa hlavne na podporu rôznych modulov, ktoré by sme v našej aplikácii mohli teoreticky využiť. Rozhodujúcim faktorom bola väčšia podpora pre OpenStreetMaps vo frameworku React. Tieto mapy budeme využívať pri zobrazovaní pretekov. Taktiež pri výbere zohral úlohu fakt, že React je momentálne populárnejší framework. Základné technológie pre našu aplikáciu teda sú Postgres, .NET a React.

Po výbere vhodných nástrojov sme zohľadnili vyťaženie častí pri reálnej prevádzke a bezpečnosť systému a rozdelili sme backend na dve časti:

1. LiveTrackerWebApi,
2. LiveTrackerSessionApi.

4.1 Databáza

Ako sme už spomínali, pre databázu sme si zvolili Postgres. Databáza sa skladá z dvoch schém public a sessions. V schéme public sú všetky tabuľky relačnej databázy podľa dátového modelu, kotrý sme navrhli. Indexy sme vytvárali tak, aby boli zaindexované všetky stĺpce podľa ktorých vyhľadávame v danej tabuľke. Každá tabuľka má primárny kľúč Id typu integer(autoincrement). Podrobný dátový model si môžeme prezrieť na obrázku 4.1 V návrhu sa nachádzajú nasledovne tabuľky:

- Collection - Id: integer(autoincrement), Name: text, UserId: integer, UseTeams: boolean, Active: boolean

  V tejto tabuľke sú uložené základné informácie o kolekcii. Id je typu integer (autoincremet) a je primárny kľúč. Toto platí pre každú tabuľku. Name je typu text a slúži pre meno kolekcie, UserId type integer je foreign key do tabuľky User. UseTeams je typu boolean, ktorý hovorí o tom či sa v kolekcii používajú tímy. Active je typu boolean a hovorí o tom či je kolekcia aktívna. Indexy sú vytvorené na stĺpce Id, Active a UserId.

- Driver - Id: integer(autoincrement), FirstName: text, LastName: text, Number: integer, TeamId: integer, Color: text, GpsDevice: uuid

  V tabuľke Driver sú uložené informácie o jazdcovi. Jeho meno a priezvisko, Number je číslo, ktoré daný jazdec používa, TeamId je foreign key do tabuľky Team. Tento údaj nie je povinný keďže nie každý jazdec musí mať referenciu na tím. Záleží od nastavenia kolekcie. Color je farba, ktorou sa jazdec zobrazí na mape alebo null ak má určený tím a GpsDevice je jedinečný identifikátor jazdca (Uuid). Indexy sú vytvorené na stĺpce Id a TeamId.

- DriverCollection - Id: integer(autoincrement), DriverId: integer, CollectionId: integer

  Táto tabuľka slúži na prepojenie relácie n to n medzi jazdcom a kolekciou, tj. medzi tabuľkami Driver a Collection. Indexy sú na všekty stĺpce.

- Layout - Id: integer(autoincrement), Name: text, GeoJson: text, UserId: integer, Active: boolean

V tabuľke Layout sa nachádza uložená trať pre používateľa podľa UserId, foreign key do tabuľky User. Name typu text je pomenovanie tohoto okruhu, GeoJson typu text samotná trať, zostringovaný formát geoJson. Active je typu boolean a hovorí o tom, či je táto trať aktívna, používateľ ju neodstránil zo svojho zoznamu. Táto tabuľka má zaindexované stĺpce Id, UserId a Active.

- Registration - Id: integer(autoincrement), Email: text, Link: text, CreationTime: timestamp

  Tabuľka Registration slúži na uchovanie registračnej url adresy pre email používateľa. Indexy sú vytvorené pre stĺpce Id a Link.

- Session - Id: integer(autoincrement), Name: text, ScheduledFrom: timestamp, ScheduledTo: timestamp, UserId: integer, Loaded: boolean, GeoJson: text, Ended: boolean, Laps: integer, CollectionId: integer, CancellationDate: boolean, CreationDate: boolean, Token: text, LayoutId: integer

  Session je tabuľka, ktorá obsahuje informácie o konkrétnej udalosti. Okrem toho ju využívame na zápis technikých udalostí. Loaded znamená, či už je udalosť načítaná v SessionApi na spracovávanie live GPS dát od organizátora a ended zasa, či už udalosť skončila. Táto tabuľka má najviac indexov a to osem. Sú na stĺpce Id, CollectionId, Ended, LazoutId, Loaded, ScheduledFrom, ScheduledTo a UserId.

- SessionNotification - Id: integer(autoincrement), SessionId: integer, NotificationType: integer

  Tanuľka SessionNotification obsahuje informácie, z ktorých sa generujú notifikácie pre používateľov, tj. referencia na udalosť a typ notifikácie. Tie Rozdeľujeme na tri typy:

  1. Created - udalosť bola vytvorená,
  2. Updated - udalosť bola upravená,
  3. Cancelled - udalosť bola zrušená.

  Indexy má na stĺpce Id a SessionId.

- SessionResult - Id: integer(autoincrement), SessionId: integer, ResultJson: text

  V tejto tabuľke ukladáme výsledy pretekov vo formáte json. Indexy sú vytvorené na stĺpce Id a SessionId.

- Subscribe - Id: integer(autoincrement), UserId: integer, OrganizerId: integer

Tabuľka Subscribe obsahuje okrem primárneho kľúča dva cudzie do tabuľky User. UserId je referencia na registrovaného diváka a OrganizerId na organizátora. Hovorí o tom, že registrovaný divák odoberá obsah organizátora. Indexy sú vytvorené pre všetky stĺpce.

- Team - Id: integer(autoincrement), Name: integer, Color: text

  Tabuľka Team obsahuje základné informácie o tíme a to sú meno a farba. Index je vytvorený len pre stĺpec Id.

- TeamCollection - Id: integer(autoincrement), TeamId: integer, CollectionId: integer

  Táto tabuľka slúži na prepojenie relácie n to n medzi tímom a kolekciou, tj. medzi tabuľkami Team a Collection. Indexy sú na všetkých stĺpcoch.

- User - Id: integer(autoincrement), Username: text, Email: text, HashedPassword: text, PasswordSalt: text, Role: integer

  Tabuľka User obsahuje informácie o používateľovi, jeho id, používateľské meno, heslo upravené pomocou hash, soľ k heslu a používateľskú rolu. Tie rozlišujeme na dva typy:

  1. NormalUser - registrovaný divák
  2. Organizer - organizátor,

  Indexy sú vytvorené pre stĺpce Id, Username, Email a Role.

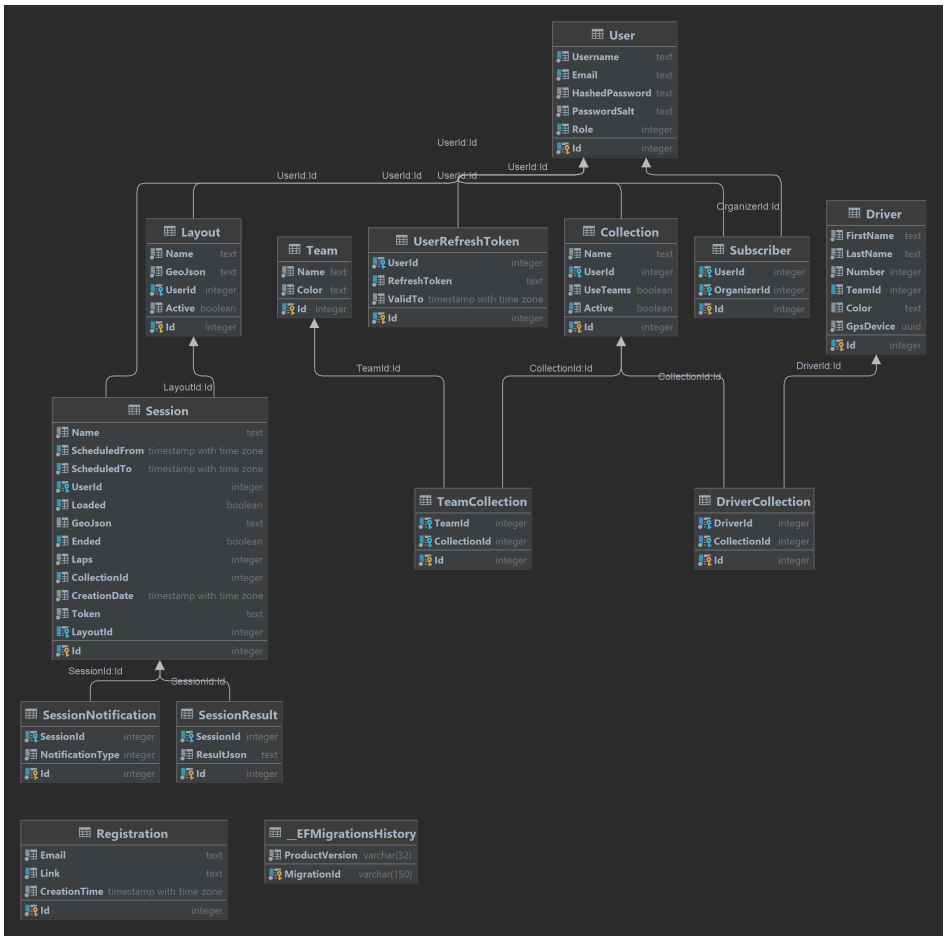
- UserRefreshToken - Id: integer(autoincrement), UserId: integer, RefreshToken: text, ValidTo: timestamp

  V tejto tabuľke nájdeme refreh token prihláseného používateľa a informáciu do kedy je platný. Indexy sú vytvorené pre stĺpce Id, UserId a RefreshToken.

- EFMigrationsHistory - Productversion: varchar, MigrationId: varchar

  Táto tabuľka je vygenerovaná automaticky pomocou Entity Framework v .NET a slúži na uchovávanie migrácii pri zmene v štruktúre databázy.

V schéme sessions sa nachádzajú tabuľky pre závody. Každý závod má v tej schéme vytvorenú vlastnú tabuľku, všetky majú rovnakú štruktúru. Tabuľky majú názov session{id pretekov} a stĺpce - Id: integer(autoincrement), Date: timestamp, Data: text. Toto riešenie zabezpečí lepší prehľad v dátach a rýchlosť selectov.



Obr. 4.1: Dátový model. Tento diagram predstavuje dátový model databázy vygenerovaný pomocou databázového prostredia DataGrip.

4.2 Backend

4.2.1 LiveTrackerWebAPI

LiveTrackerWebAPI je ASP .NET Core Web API, ktoré slúži primárne ako backend pre webovú aplikáciu. Pôvodne sme ho začali vyvýjať na verzii .NET 7.0, ale hneď po vydaní novšej verzie .NET 8.0 sme zvýšili verziu na našom API, nie len kvôli tomu aby sme mohli využívať najnovšie vychytávky, ale verzia 8.0 je dlhodobo podporovaná narozdiel od 7.0. Pri tvorbe tohoto API sme sa držali štandardov a odporúčaní Microsoftu pri práci s .NET [3]. Použili sme trojvrstvovú architektúru:

1. Controller
2. Service
3. Repository

Toto riešenie je pre nás najvhosnejšie, pretože je v ňom dodržané členenie aplikácie a dostatočná modularita a zároveň nie je príliž komplikované na implementáciu. Control-

ler je prezentačná vrstva nášho API. V tejto časti sa nachádzajú endpointy rozdelené do nasledovných celkov, ktoré vyplývajú najmä z používateľských scenárov:

- Archive,
- Collection,
- Layout,
- Recaptcha,
- Registration,
- Session,
- Subscribe,
- User.

Service je logická vrstva a tu sa nachádza celá logika backendovej časti. Pre každý controller existuje samostatný servis, aby sme zachovali členenie kódu a architektúru API. Repository predstavuje v našom prípade Entity Framework. Ten slúži na pohodlnú komunikáciu s databázou bez nutnosti písania sql. S databázou sa pracuje pomocou tried - entít, ktoré sú zhodné s tabuľkami v databáze.

Aplikácia je rozdelená do troch projektov podľa vrstiev:

1. API
2. BL
3. DB

Projekt najvrchnejšej vrstvy API obsahuje súbor Program.cs, kontrolery, konfiguráciu a má referenciu na projekt BL. V Program.cs sú všetky potrebné informácie na spustenie aplikácie, registrácia servisov, načítanie konfigurácie, nastavenie autentifikácie a cors policies. Pomocou cors vieme nastaviť z akých domén bude request na API povolený, prípadne aké metódy, hlavičky requestov, cookies pre túto doménu povolíme. Servisy vieme v .NET zaregistrovať tromi spôsobmi:

- Singleton,
- Scoped,
- Transient.

Registrujú sa takým spôsobom, že pre každý servis máme vytvorený interface, ktorý obsahuje definície metód a servisná trieda tento interface implementuje. V Program.cs ho potom už len zaregistrujeme s vybraným životným cyklom. Singleton sa vytvorí pri štarte aplikácie a po celý čas sa využíva tá istá inštancia. Scoped znamená, že pre každý http request sa vytvorí nová inštancia a tá sa používa počas trvania tohoto request. Ak zvolíme Transient, znamená to, že pri každom vyžiadaní triedy dostaneme novú inštanciu. Tým, že servisy registrujeme pomocou interface, máme ošetrenú dependency injection. Triedy servisov nevytvárame, ale pridáme ich do konštruktora triedy, v ktorej cheme servis využiť. .NET sa postará o to, aby nám v tomto konštruktore bola vydaná inštancia podľa životného cyklu s ktorým sme serivs zaregistrovali [4].

Orem servisov vieme v .NET zaregistrovať aj http klientov, ktorým definujeme kľuč, base url, prípadne autentifikáciu načítanú z konfigurácie. Potom v servise vieme zavolať tohoto klienta pomocou HttpCientFactory. Toto riešnie predstavuje omnoho efektívnejšie využívanie http klientov ako zakaždým vytvárať novú inštanciu triedy HttpClient. Ak by sme pri každom volaní vytvárali novú inštanciu, pri väčšej prevádze by mohlo prísť k vyčerpaniu soketov. Používať http klienta ako singleton nie je správne riešnie, pretože môžu nastať problémy pri zmenách DNS. HttpCientFactory rieši všetky tieto obtiažnosti so samotnou inštanciou a navyše IHttpClientFactory vieme použiť v konštrutore servisu pomocou dependency injection [5].

V projekte pre vrstvu BL sa nachádza logika celej apliácie a má referenciu na DB projekt. Sú tu definície aj implementácia servisov, modelové triedy, vlastné výnimky, middlevéry, utility, workery, konštanty a enumy. Okrem servisov pre controlleri sa v našej aplikácii nachádza EmailService určený na posielanie emailov pomocou knižnice MimeKit a MailKit, pomocu ktorých vieme posielať html emaily. Ďalej sa tu nachádza NotificationService, ktorý slúži na načítanie a mazanie notifikácii, SessionAPIService určený pre komunikáciu so SessionAPI a JWTService pomocou ktorého generujeme token.

Modelové triedy máme rozdelené do priečinkov podľa ich významu:

- Dto,
- Logic,
- ViewModel.

Dto je skratkové slovo pre data transfer object a tieto triedy slúžia len na prenos dát medzi prezentačnou vrstvou a dátovou [6]. Tieto modely využívame na vstupe do POST alebo PUT metód. Často sa stáva, že objekt s frontendu má trochu inú štruktúru ako databázová entita, či už z bezpečnostných alebo optimalizačných dôvodov. K tomu nám slúžia dto. V logickej vrstve tento objekt spracujeme a pretransformujeme do entít, aby sme s nimi vedeli ďalej pracovať na úrovni databázy. Logické triedy využívame najmä

na pomocnú funckiu v logickej vrstve. View modely používame na výstp z GET metód. Tieto triedy sú zhodné s frontendovými, ktoré slúžia na zobrazenie dát pre používateľa. Skladať view modely na backende je efektívnejšia cesta, než skladať výsledný objekt až na frontende. Ušetríme tým viac http volaní, čo značne urýchli načítanie stránky.

V projekte DB vrstvy máme uložené entity a definíciu DbContextu Entity frameworku a definície pre cudzie kľúče medzi jednotlivými entitami.

4.2.2 LiveTrackerSessionAPI

LiveTrackerSessionAPI je taktiež ASP .NET Core Web API vyvýjaný na .NET 8.0. Táto aplikácia slúži na prijímanie GPS dát od organizátora, ich spracovanie, vyhodnotenie pozícii, odoslania do frontedovej časti a uloženia dát do databázy. Tak ako aj pri našom Web API sme sa rozhodli pre trojvrstvovú architektúru, avšak vrstvy sme nerozdelovali do samostatných projektov. Aplikácia obsahuje dva kontrolery:

- RaceController,
- TokenController.

RaceController je prístpný pre organizátora a slúži na načítanie pretekov, príjem GPS dát a zíkanie výsledkov. TokenController zasa vygeneruje autentifikačný token pre organizátora. Tak ako pri predošlej aplikácii, každý kontroler má svoj servis a nachádza sa tu aj trieda Repository na komunikáciu s databázou a jej interface. V tomto projekte nepoužívame EntityFramework, kvôli potrebe ukladania dát do tabuliek zo schémy Sessions. Entity Framework neumožnuje mať jednu entitu pre viac tabuliek a keďže session tabuľky majú rovnakú štruktúru a pre každý závod existuje samostatná tabuľka, k databáze pristupujeme cez NpgSql. V tomto projekte sa ešte nachádzajú logiké tiredy.

V samostantnom projekte SessionAPIModels sa nachádzajú Dto a ViewModel triedy. Je to z toho dôvodu, že LiveTrackerWebAPI a LiveTrackerSessionAPI musia spolu komunikovať a tým vzniká potreba mať rovnaké triedy na oboch stranách. Týmto vyčleneím spoločných tried do samostaného projektu, máme možnosť z neho vytvoriť nuget package a naimportovať do nášho WEB Api, takže nemusíme vytvárať duplicitné triedy v dvoch projektoch.

Okrem toho sa v tomto solution nachádza ešte projekt TestData, ktorý slúži na spustenie simulácie pretekov. Taktiež má referenciu na SessionAPIModels.

4.3 Frontend

Frontend v našej aplikácii predstavuje WebApplication, pre ktorú sme zvolili framework React. Namiesto jazyka javascript sme si vybrali typescript. Výhodou typescriptu je

určovanie typov, čo nám robí kód lepšie čitateľným a vieme predísť neočakávaným chybám, ak nevieme akého typu daná premenná je. Používame verziu typescriptu 4.9.5. Aplikácia je založená na node 20.12.0 a vytvorili sme ju pomocou Vite. Je to nástroj, ktorý nám ponúka jednoduché vytvorenie aplikácie a jej základnej štruktúry a navyše nám zabezpečí optimálnejšie build procesy [7].

Vo frameworku React vytvárame používateľské prostredie pomocou komponentov. Ten väčšinou pozostáva z dvoch častí. Prvá časť obsahuje premenné, konštanty, funkcie a hooks. Hooks sú špeciálne fukcionálne kompomenty, ktoré majú jasne definované správanie a manažujeme pomocou nich zmeny na webovej stránke. V našej práci sme využívali hlavne tieto 3 typy:

- useSate - slúži na uchovanie stavu a sa môže meniť,
- useEffet - reaguje na zmenu naviazaného stavu,
- useRef - uchováva hodnotu medzi renederovaním.

UseState má takú vlastnosť, že vždy pri zmene jeho stavu sa znovu vykreslí daný komponent. Preto sme tento hook napojili na všetky vstupy od používateľa vo formulároch alebo sme doň uložili dáta, ktoré nám prišli z backendu a používateľ videl zmenu stavu okamžite. Ak nebolo nutné prekreslenie komponentu, použili sme useRef, ktorý udržuje referenciu aj medzi týmito prekresleniami. Ak sme potrebovali vykontať akciu hneď po zmene stavu, naviazali sme na tento stav useEffect, v ktorom sme danú akciu definovali [8].

Druhá časť komponentu je return a tu pomocou html definujeme to, čo sa vykreslí používateľovi na stránke. Hlavný komponent môže obsahovať v sebe množstvo menších komponentov, čím vieme dosiahnuť peknú štruktúru a modularitu aplikácie. Vyčlenením spoločných častí do samostaného komponentu predídeme duplikácii kódu.

Pri návrhu sme sa držali defaulnej štruktúry Reactu a celá implementácia sa nachádza v priečinku src. V ňom je umiestnený dôležitý súbor App.tsx, ktorý obsahuje hlavnú definíciu aplikácie, je to koreňový komponent, každý ďalší na nachádza pod ním. Nájdeme tu aj route definície, čo znamená na akej ceste v url adrese sa aký komponent použije. Ďalej sa v hlavnom priečinku src nachádza ešte App.css, kde sú spoločné štýly a Constatns.ts, kde sú spoločné konštanty.

Ďalej sme zvolili nasledovnú štruktúru:

- Components,
- Models,
- Services

- Utils.

Komponenty sú ešte ďalej rozdelené podľa používateľských scenárov prípadne funkcionality podobne ako v LiveTrackerWebAPI. Rozdelili sme ich nasledovne:

- Collections,
- ErrorPages,
- HomePage,
- Layouts,
- Login,
- Navigation,
- Race,
- Registration,
- Sessions,
- Users,
- Common.

Rovnako sú rozdelené aj modelové interfejsi. Tie sme nerozdelili do samostaných priečinkov, ale len typescript súborov. Servisy sú zhodné s kontrolermi v LiveTrackerWebAPI, pre každý backend kontroler existuje servis na frontende. V Utils sa nachádzajú spoločné fukcie využívané vo viacerých komponentoch a trieda apiClient.ts. Ide o našu vlastnú nadstavbu nad axios, ktoré slúži na vykonávanie HTTP dotazov a tým zabezpečuje komunikáciu s Web API.

Pre responzivitu a používateľsky príjemný dizajn webovej aplikácie sme zvolili react-bootstrap a ikony z react-bootstrap-icons. Všetky fromuláre a vyskakovacie okná sú z knižnice react-bootstrap. Pre tabuľky sme zasa zvolili knižnicu mui/material.

Literatúra

[1] RaceTracker AS. Racetracker as, 2018. Dostupné z https://racetracker.no/.

[2] GB RaceTracker. Gb race tracker, 2022. Dostupné z https://www.gbracetracker.co.uk/.

[3] Rick Anderson, Kirk Larkin. Tutorial: Create a web api with asp.net core, 2024. Dostupné z https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&tabs=visual-studio/.

[4] Kirk Larkin, Steve Smith, Brandon Dahler. Dependency injection in asp.net core, 2023. Dostupné z https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0/.

[5] Microsoft. Use ihttpclientfactory to implement resilient http requests, 2023. Dostupné z https://learn.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/use-httpclientfactory-to-implement-resilient-http-requests/.

[6] Microsoft. Create data transfer objects (dtos), 2022. Dostupné z https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5/.

[7] Vite Team. Why vite. Dostupné z https://vitejs.dev/guide/why.html/.

[8] Meta Open Source. Built-in react hooks. Dostupné z https://react.dev/reference/react/hooks/.

[9] Jignesh Trivedi. Jwt authentication in asp.net core, 2023. Dostupné z https://www.c-sharpcorner.com/article/jwt-json-web-token-authentication-in-asp-net-core/.

[10] Google. recaptcha v3, 2022. Dostupné z https://developers.google.com/recaptcha/docs/v3/.