FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Systém pro Mistrovství ČR v autostopu
Student:	Bc. Karím Abu Nofal
Vedoucí:	Ing. Jana Ernekerová
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je kompletní tvorba webové aplikace pro správu obsahu webu charitativního závodu MČR v autostopu.

Postupujte dle následujících kroků:

1. Analyzujte případy užití současného řešení a zjistěte, jaké nové funkce by měl systém podporovat.
2. Na základě analýzy proveďte vhodný návrh.
3. Návrh zrealizujte v podobě funkčního prototypu.
4. Prototyp podrobte vhodnými Vámi navrženými testy.
5. Na základě testování prototyp upravte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 23. září 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Systém pro Mistrovství ČR v autostopu

Bc. Karím Abu Nofal

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Jana Ernekerová

6. května 2021

Poděkování

Děkuji vedoucí mé diplomové práce za její čas a podporu. Dále děkuji organizátorům závodu za příjemnou spolupráci a v neposlední řadě děkuji rodině a přátelům za podporu, kterou mě povzbuzovali po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2021 . . . . . .

Odkaz na tuto práci

Abu Nofal, Karím. *Systém pro Mistrovství ČR v autostopu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá vývojem nové verze webové aplikace pro závod Mistrovství ČR v autostopu, který pořádá Světem stopem z. s. Text obsahuje seznámení se současnou verzí webových stránek, na jejichž základě stojí návrh nové aplikace. Hlavním cílem bylo analyzovat požadavky organizátorů, vybrat technologie potřebné k implementaci, navrhnout novou verzi a tu následně implementovat. Vývoj probíhala ve frameworku Django v programovacím jazyce Python. Výsledný prototyp byl nasazen do cloudu, aby mohl být podroben uživatelskému testování. To odhalilo drobné nedostatky, které byly posléze opraveny.

Klíčová slova webová aplikace, Django, Python, Mistrovství ČR v autostopu

Abstract

This work deals with the development of a new version of the web application for the Czech Championship race in hitchhiking, organized by Světem stopem z. s. The main goal was to analyze the requirements of the organizers, select the technologies needed for implementation, design a new version and then implement it. The development was carried out in the Django framework in the Python programming language. The resulting prototype was deployed in the cloud to be tested by users. This revealed minor flaws that were later corrected.

Keywords web application, Django, Python, Czech Hitchhiking Championship

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Cestování patří mezi oblíbené volnočasové aktivity mnoha lidí. Existuje široká škála způsobů cestování, od pěší turistiky až po zaoceánské plavby. Jedním z takových způsobů je autostop. Ten se díky jeho nízkým nákladům a vidinou nepředvídatelných událostí stává favoritem u lidí vyhledávajících nové zážitky. Tato skutečnost dala vzniku několika úspěšným závodům po Evropě, které mají stopování jako hlavní disciplínu.

Jedním z těchto závodů je Mistrovství ČR v autostopu. Organizátoři však závodu chtěli dát nějaký hlubší smysl, a tak ke klasickému hodnocení v závodě (tj. kdo je nejblíž cíli, ten je první v pořadí) přidali element sponzorů. To znamená, že diváci sledující tento závod mohou své oblíbené posádce posílat peníze, díky kterým posádka obdrží plusové body do hodnocení. Peníze vybrané od všech posádek po ukončení závodu putují na pomoc předem vybrané osobě, na kterou pořádá sbírku Konto Bariéry

Světem stopem z. s. jsou nespokojeni s aktuální verzí webových stránek závodu, a proto se rozhodli vypsat téma této diplomové práce. Některé úkony ohledně správy závodu by chtěli automatizovat a tím usnadnit práci jak sobě, tak účastníkům závodu. Například odevzdávání úkolů nemusí probíhat tak, že fotky a texty poslané organizátorům přes sociální sítě budou do systému zadávat organizátoři ručně. Cílem práce je tedy vytvořit webovou aplikaci, která bude lépe splňovat jejich požadavky.

Text této práce popisuje softwarový vývoj webové aplikace, včetně administračního prostředí. V první kapitole seznamuje čtenáře se současnou verzí webu. V další kapitole je analyzována architektura, technologie, požadavky a případy užití. Na základě těchto dat je v třetí kapitole navržena nová webová aplikace, jejíž implementace je popsaná v kapitole čtvrté. Následující dvě kapitoly se zabývají nasazením a uživatelským testováním. Poslední kapitola shrnuje výsledky práce a diskutuje o možnosti dalších rozšíření aplikace do budoucna.

Popis závodu

Do hlavního závodu je potřeba se kvalifikovat. Jak v hlavním, tak kvalifikačním závodě se účastní posádky o dvou členech. Posádky, kterých může být libovolné množství, se registrují do kvalifikačního závodu mimo webové stránky. Následně soutěží v krátkém závodě a prvních dvanáct posádek se kvalifikuje do hlavního závodu.

Hlavní závod, při kterém soutěžící stopují po Evropě, trvá přibližně týden. Jejich trasa je určena různými úkoly, většina z nich je povinná, ale existují i dobrovolné úkoly. Pro splnění úkolu musí posádka pomocí sociálních sítí nebo SMS/MMS zpráv odeslat fotku a text se zážitkem z plnění úkolu. Počet splněných úkolů je jedním ze dvou současných kritérií při hodnocení posádky. Za každý splněný úkol posádka obdrží jeden bod, v některých případech i jiný bonus, jako zkrácení celkového času, za který závod dokončila. Druhým kritériem je množství vybraných peněz od návštěvníků webu. Posádky u sebe vozí GPS zařízení, pomocí kterého je jejich poloha zobrazována na webových stránkách. Současně si návštěvníci webu mohou přečíst zážitky z cest, které na web píší posádky. Návštěvníci webu se tak mohou rozhodnout některou posádku podpořit, a to ve formě DMS.

Po ukončení závodu se vybraným obnosem přispěje do veřejné sbírky Konto Bariéry na pomoc předem vybrané osobě. Více informací o Konto Bariéry je možné získat na jejich webových stránkách **kontobariery.cz** [3]

Kapitola 1

Stávající řešení

V této kapitole je rozebráno současné řešení webových stránek Mistrovství ČR v autostopu (**www.mcrautostop.cz**). Podle organizátorů jsou nynější stránky zastaralé, uživatelsky nepřívětivé, což dalo vzniku této diplomové práci. Nedostatky současných webových stránek jsou shrnuty v podsekcích popisujících jejich části. Analýza současného řešení je kvalitním zdrojem informací o tom, na co je potřeba se při vývoji nové verze zaměřit, čeho se vyvarovat a jaké nové funkcionality by bylo dobré přidat.

1.1 Klientská část

Nynější verze webových stránek je napsána ve skriptovacím jazyce PHP. Při implementaci nebyl kladen velký důraz na návrh, který by umožňoval budoucí rozšíření aplikace. Absence použití jakéhokoli webového frameworku tohoto jazyka činí údržbu webu značně složitou. Pro ukládání dat aplikace využívá databázový server MariaDB a je nasazená na serverech společnosti **WEDOS**.

1.1.1 Sledování posádek

Jedním z hlavních požadavků na stránky je sledování poloh posádek a vzdálenosti, kteou od začátku závodu stihly urazit. Služby zajišťující tyto informace zprostředkovává hlavní partner závodu GPS Dozor. Každá posádka s sebou vozí GPS zařízení, které o sobě každých třicet vteřin odesílá informace na servery GPS Dozoru. Z těchto serverů je následně možné pomocí vystaveného API informace získat a zobrazit. API, které využívá současný web, je však staré a společností GPS Dozor již není podporované. Je třeba aktualizovat aplikaci, aby využila jejich nově vystaveného API.

1.1.2 Platby

První ze dvou kritérií při vyhodnocení závodu je množství peněz, které dané posádce poslali fanoušci. Momentálně lze posílat peníze pouze formou DMS. Každá posádka má přidělené jedno telefonní číslo, na kterém se akumulují pro ně poslané peníze. Dárcovské SMS je však možné posílat pouze na území České republiky, což značně omezuje množinu lidí, kteří by potenciálně mohli přispět. Detailnější informace o DMS jsou popsány v sekci 4.6.

1.1.3 Organizace posádek

Druhé kritérium hodnocení je počet úkolů, které posádka stihla do ukončení závodu splnit. Aby úkol mohla dokončit, musí organizátorům, odeslat fotografii nebo videozáznam z jeho konání společně s textem, který popisuje průběh plnění. Správa úkolů však není implementována v klientské části, takže závodníci nemohou přidávat své příspěvky skrze webové stránky. Místo toho komunikují s organizátory přes sociální sítě nebo formou SMS/MMS. Tímto způsobem odešlou splnění úkolu organizátorům, kteří ho následně musí ručně zadat v administrační části aplikace do systému.

1.1.4 Lokalizace

Během závodu účastníci cestují po celé Evropě. Na cestách se setkávají s mnoha lidmi, kterým vysvětlují, čeho se právě účastní. Tito lidé by třeba chtěli sledovat, jak se dané posádce v závodu daří, popřípadě ji i podpořit skrze DMS, ale nemůžou. Současná aplikace nepodporuje vícejazyčnost, která je v dnešní době pro webové aplikace standardem.

1.1.5 Vzhled aplikace

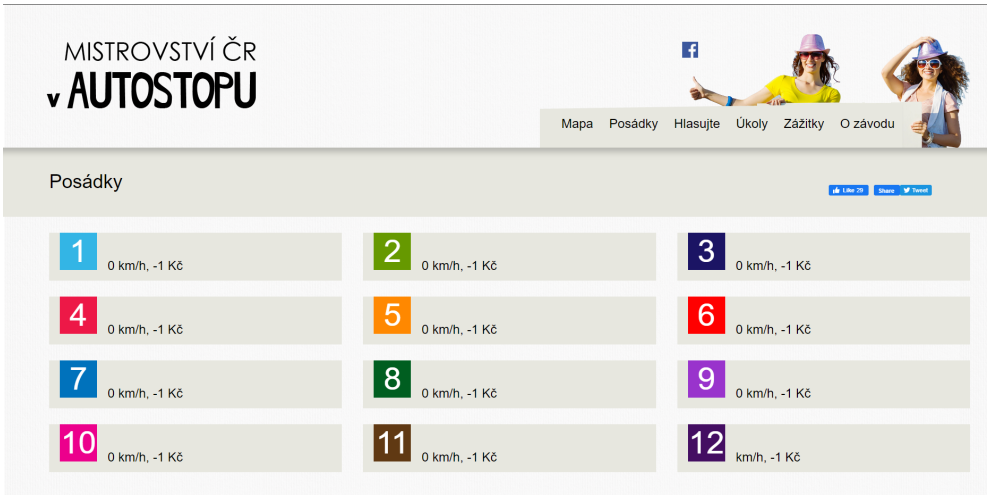
V současném řešení nebyla použita pro úpravu vzhledu základních HTML komponent žádná šablona, ani žádná CSS knihovna, jako např. Bootstrap. Vzhled je řešen sadou vlastních CSS stylů, které v porovnání s moderními webovými stránkami působí zastarale.

Jelikož aplikace neprošla žádným uživatelským testováním, rozvržení stránek a následná orientace v nich není vždy intuitivní a pro návštěvníky webu může být matoucí.

1.2 Administrativní část

Administrační část současné aplikace je CMS[1], který si zakladatel závodu nechal zhotovit pro svou bývalou firmu, a zde je použita jeho upravená verze.

[1] CMS, anglicky Content Management System - rozhraní, které umožňuje uživateli publikovat obsah přímo na web.[4]



Obrázek 1.1: Snímek obrazovky webových stránek současného řešení

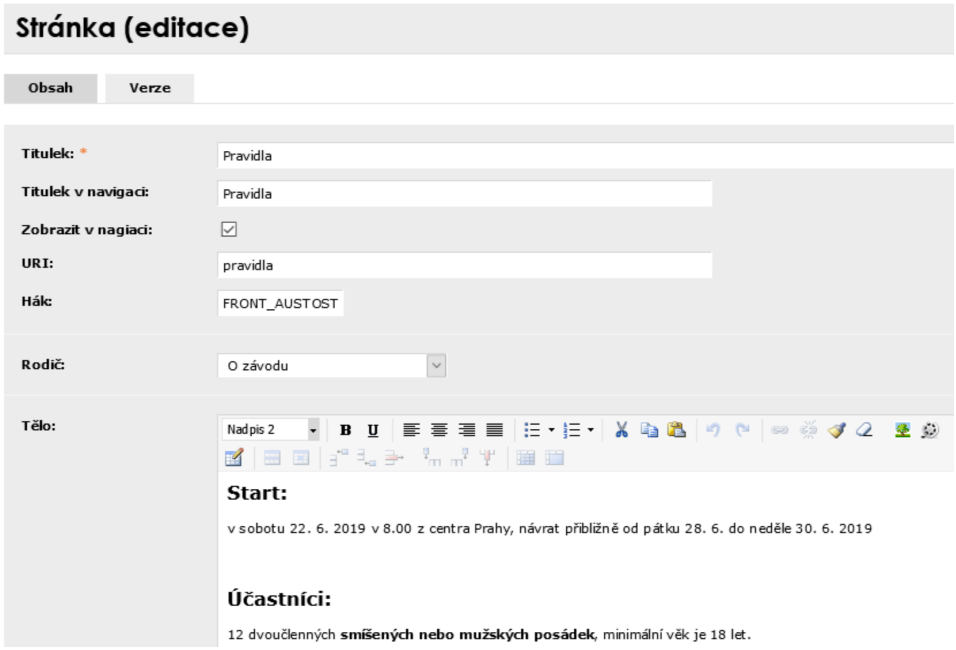
Pomocí tohoto systému se generují šablony, které se následně používají pro zobrazování stránek v klientské části. Toto řešení připadá současným organizátorům velice nepřehledné a zmatečné. Například pro založení nového závodu se musí založit kompletně nová stránka, místo toho aby se založil jen záznam v databázi a použila se jednotná šablona. Proto jsou webové stránky jednotlivých závodů odlišné a tato nekonzistentnost může uživatele zmást. Organizátoři by raději měli jednotnou šablonu a spravovali by pouze zobrazovaná data, namísto vytváření celých stránek jako takových.

1.3 Shrnutí nedostatků aplikace

V poslední části této kapitoly je popsáno několik základních nedostatků současné aplikace, které bude cílem eliminovat v nově navrhované verzi.

- **Vzhled aplikace** - zastaralý vzhled a nekonzistentní styly HTML komponent,
- **Platby** - posílání příspěvků posádkám je možné pouze na území České republiky,
- **Posádky** - registrace na závod, odevzdávání úkolů a psaní vlastních zážitků je zajištěno pomocí externích komunikačních kanálů, nikoli pomocí webových stránek,
- **Administrativní část** - pro organizátory nepřehledná a složitá,
- **GPS Dozor API** - použití starého, již nepodporovaného API,

- **Lokalizace** - webové stránky jsou pouze v českém jazyce.



Obrázek 1.2: Snímek obrazovky administrační části současného řešení

Kapitola 2

Analýza

2.1 Funkční požadavky

Funkční požadavky jsou vlastnosti produktu nebo funkce, které musí vývojář implementovat, aby uživatelům umožnil splnit jejich úkoly.[5]

FP1: Správa účtu
Uživateli bude umožněno registrovat svoji posádku do kvalifikačního závodu. Pokud ta postoupí do hlavního závodu, bude jí vytvořen uživatelský účet, přes který se budou moci oba členové posádky přihlásit do aplikace, kde jim bude umožněno profil spravovat.

- Registrace do kvalifikačního závodu,
- Přihlásit se,
- Odhlásit se,
- Upravit profil,
- Obnovit heslo.

FP2: Zobrazení výsledků
Návštěvníci webových stránek budou mít možnost zobrazit si aktuální výsledky a statistiky závodu. Kromě celkového pořadí budou moci sledovat i výsledky jednotlivých částí hodnocení. Poloha posádek bude zobrazena na mapě.

- Zobrazit celkové pořadí,
- Zobrazit pořadí podle počtu splněných úkolů,
- Zobrazit pořadí podle vzdálenosti od cíle,

- Zobrazit pořadí podle vybraných peněz,
- Zobrazit posádku na mapě,
- Zobrazit celkové množství vybraných peněz.

FP3: Přehled posádek
Aplikace bude umožňovat procházení posádek aktuálního závodu.

- Zobrazit seznam posádek,
- Zobrazit detail posádky.

FP4: Přehled úkolů
V aplikaci bude možné procházet úkoly právě probíhajícího závodu. Přihlášený člen posádky bude moci odevzdat vybraný úkol. Odevzdaný úkol se stane viditelným pro veřejnost a bude započítaný do statistik po schválení jedním z administrátorů závodu.

- Zobrazit seznam úkolů,
- Zobrazit detail úkolu,
- Odevzdat úkol.

FP5: Přehled zážitků
Posádka může během závodu přidávat své zážitky, které budou k přečtení pro návštěvníky webových stránek. Přidaný zážitek musí být nejdříve schválen jedním z administrátorů závodu. Zážitky se budou moci filtrovat podle posádky.

- Zobrazit seznam zážitků,
- Zobrazit detail zážitku,
- Vytvořit zážitek.

FP6: Informace o závodu
Návštěvníkům webových stránek bude umožněno přečíst si veřejné informace o závodu, jeho pravidla a často kladené otázky.

FP7: Články o charitativní činnosti závodu
Každý závod bude mít jeden článek před a po dobu konání závodu a jeden článek po ukončení. První s informacemi o tom, na co se daný ročník vybírají peníze, a druhý jako shrnutí daného ročníku. Návštěvníci webových stránek si budou moci procházet jak aktuální články, tak i ty z minulých let.

FP8: Přispívání posádce
Návštěvníci stránek budou moci přispívat jimi vybrané posádce. Budou mít na výběr z několika způsobů placení.

- DMS,
- PayPal,
- Platební karta.

FP9: Administrace webu
Aplikace bude umožňovat přihlásit se jako administrátor do administrační části aplikace. Ten bude mít pravomoci cokoli vytvářet, měnit i mazat. Dále bude v této části aplikace moci schvalovat zážitky posádek a potvrzovat splnění úkolů.

FP10: Lokalizace klientské části aplikace
Návštěvníci webových stránek budou mít možnost vybrat si jazyk, ve kterém se budou stránky zobrazovat.

FP11: Sociální sítě
Aplikace bude umožňovat otevřít nová okna se sociálními sítěmi Mistrovství ČR v autostopu.

2.2 Nefunkční požadavky

Nefunkční požadavky popisují, jak se systém musí chovat, a stanovují omezení jeho funkčnosti.

NP1: Webová aplikace
Aplikace bude webová a dostupná přes webový prohlížeč.

NP2: Podpora webových prohlížečů
Aplikace bude podporovat níže zmíněné webové prohlížeče, a to jak na počítači, tak na mobilních zařízeních.

- Chrome,
- Firefox,
- Safari.

NP3: Responzivita
Klientská část aplikace bude responzivní, a tím dobře zobrazitelná i na mobilních zařízeních.

NP4: Vícejazyčnost
Klientská část bude implementována v českém a anglickém jazyce s možností přidání dalších jazyků.

NP5: Rozšiřitelnost
Aplikace bude napsána tak, aby bylo možné přidávat rozšíření o další funkcionality bez většího zásahu do již implementovaných.

2.3 Případy užití

Případy užití vznikly na základě funkčních požadavků na aplikaci. Diagramy případů užití byly kvůli přehlednosti rozděleny podle jednotlivých aktérů. Některé z nich byly rozepsány do kroků, které je potřeba provézt pro docílení chtěné funkcionality.

2.3.1 Aktéři

Aplikaci mohou využívat tři typy uživatelů:

- **Nepřihlášený uživatel**
  Nepřihlášený uživatel může využívat většinu funkcí webových stránek (viz Diagramy případů užití).
- **Přihlášený uživatel**
  Přihlášený uživatel má kromě možností nepřihlášeného uživatele k dispozici funkce pro správu vlastní posádky.
- **Administrátor**
  Administrátor má jako jediný přístup do administrační části aplikace, kde má právo spravovat veškerá data aplikace.

2.3.2 Případy užití nepřihlášeného uživatele

2.3.2.1 PU1: Zobrazit výsledky

1. Případ užití začíná na úvodní obrazovce, kde je zobrazená mapa Evropy a sloupeček s celkovými statistikami.
2. Uživatel si zvolí, podle kterého kritéria chce zobrazit výsledky.
3. Aplikace zobrazí tabulku posádek s jejich pořadím.
4. Uživatel klikne na řádek s posádkou.
5. Aplikace zobrazí detailní informace o výsledcích zvolené posádky a zvýrazní její polohu na mapě.

2.3.2.2 PU10: Přispět posádce

1. Případ užití začíná, když se uživatel rozhodne podpořit posádku.
2. Uživatel spustí akci chci přispět.
3. Aplikace zobrazí seznam posádek, na které může přispět.
4. Uživatel vybere posádku.
5. Aplikace zobrazí možnosti platebních metod a nechá uživatele vybrat částku.
6. Uživatel po vyplnění částky a zvolení platební metody dokončí platbu na základě typu platby, kterou vybral.
7. Pokud to platební metoda umožňuje, aplikace zobrazí informaci o úspěšně provedené platbě.

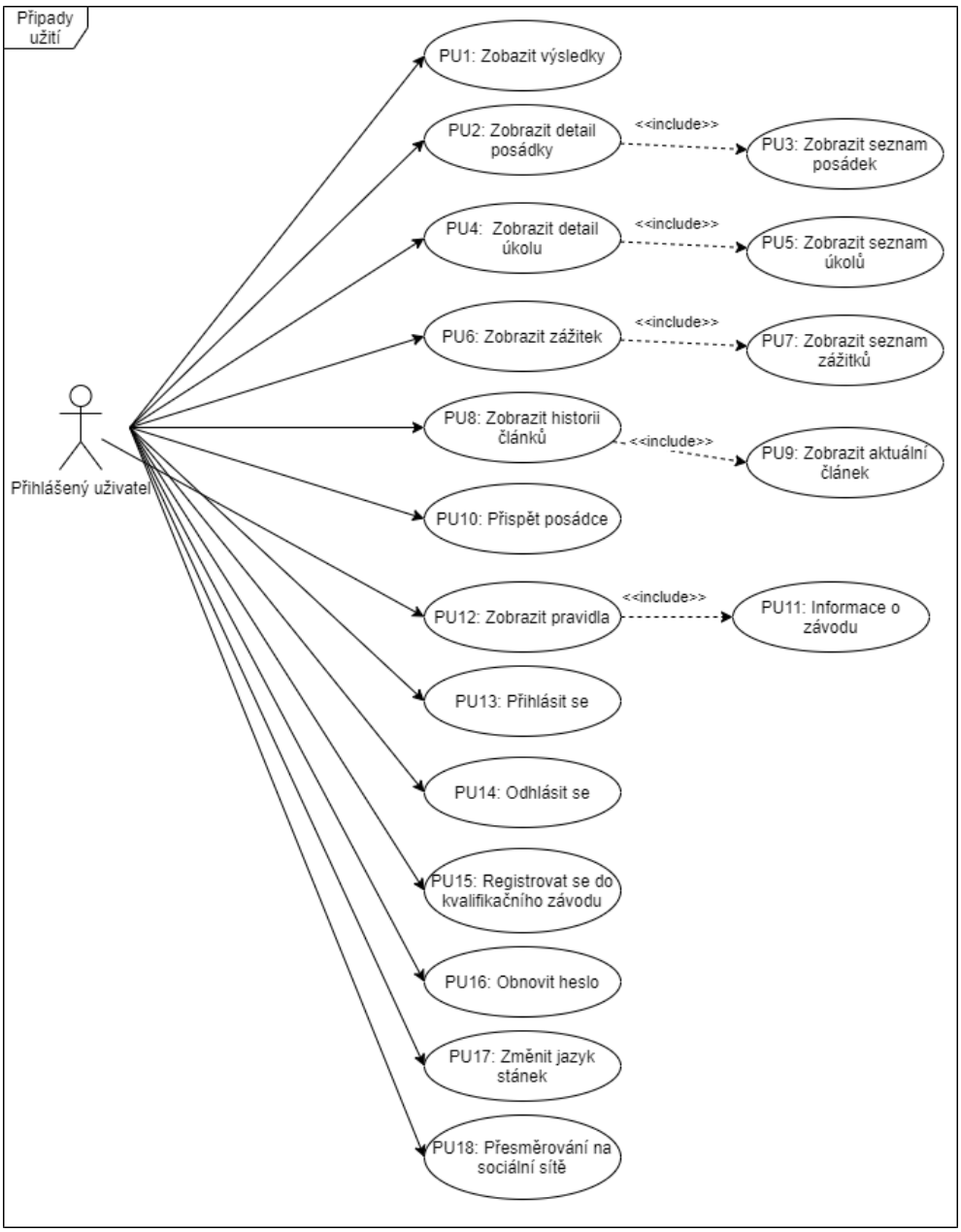
2.3.2.3 PU15: Registrace do kvalifikačního závodu

1. Případ užití začíná za předpokladu, že je aktivní kvalifikační závod a uživatel se chce registrovat.
2. Uživatel zvolí akci registrovat se.
3. Aplikace postupně zobrazí tři formuláře: První člen posádky, Druhý člen posádky a Informace o posádce.
4. Pokud byly zadané informace validní, aplikace informuje uživatele o dokončení registrace a automaticky odešle e–maily s dalšími informacemi.

2.3.2.4 PU16: Obnovit heslo

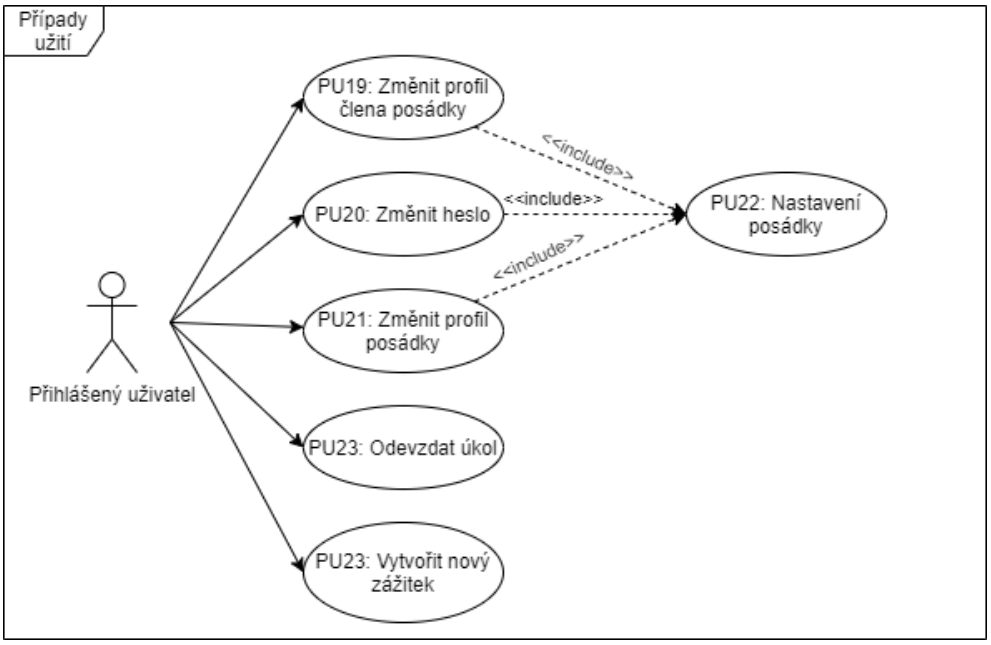
1. Případ užití začíná, když uživatel potřebuje změnit své přihlašovací heslo, ale zároveň není přihlášen do aplikace.
2. Na stránce s přihlašovacím formulářem uživatel spustí akci obnovit heslo.
3. Aplikace zobrazí formulář pro obnovení hesla.
4. Uživatel vyplní formulář.
5. Pokud byl formulář vyplněný správně, odešlou se na e–maily obou členů posádky URL adresy pro obnovení hesla. V opačném případě aplikace informuje uživatele o chybně zadaných hodnotách.
6. Po otevření zaslaného URL je uživatel přesměrován na formulář pro změnu hesla.

7. Pokud uživatel vyplní formulář správně, tak je přesměrován na přihlašovací obrazovku a je informován o dokončení změny hesla. V opačném případě je aplikací informován o chybně zadaných hodnotách.



Obrázek 2.1: Diagram případů užití pro nepřihlášeného uživatele

2.3.3 Případy užití přihlášeného uživatele



Obrázek 2.2: Diagram případů užití pro přihlášeného uživatele

2.3.3.1 PU20: Změnit heslo

1. Případ užití začíná, když je uživatel v nastavení posádky a chce změnit heslo.

2. Uživatel zvolí akci pro změnu hesla.

3. Aplikace zobrazí formulář pro změnu hesla. V tomto případě je vyžadováno stávající heslo.

4. Pokud uživatel vyplní formulář správně, tak je přesměrován na stránku nastavení posádky a je informován o úspěšném dokončení změny hesla. V opačném případě je informován o chybně zadaných informacích.

2.3.3.2 PU23: Odevzdat úkol

1. Případ užití začíná, když posádka splnila úkol a chce ho odevzdat. Přitom se musí nacházet na stránce s detailem daného úkolu.

2. Uživatel spustí akci pro odevzdání úkolu.

3. Pokud uživatel daný úkol může odevzdávat, aplikace zobrazí formulář pro odevzdání úkolu. V opačném případě aplikace informuje uživatele, z jakého důvodu momentálně nemůže daný úkol odevzdat.

4. Uživatel vyplní formulář, vybere fotografii a ořízne ji na fixní poměr stran. Následně formulář odešle ke schválení.

5. Pokud byl formulář vyplněn správně, aplikace odešle e–mail organizátorům, že vznikl nový požadavek na schválení úkolu, a informuje uživatele, že jeho žádost byla odeslána. V opačném případě aplikace informuje uživatele o chybně vyplněném formuláři.

2.3.4 Případy užití administrátora

V administrativní sekci může administrátor vytvářet, upravovat i mazat veškeré subjekty datového modelu aplikace. V diagramu případu užití pro administrátora byly pro přehlednost subjekty shrnuty do čtyř logických celků.

2.3.4.1 PU27 - PU30

Tyto případy užití znázorňují správu subjektů, které danému logickému celku náleží. Níže je rozepsána pouze úprava posádky, a to proto, že ostatní případy užití fungují obdobně, pouze se změnou subjektu a akce, kterou s danou posádkou chce provést.

1. Případ užití začíná v administrativní části aplikace, když administrátor chce upravit profil posádky.

2. Administrátor zobrazí sekci s posádkami.

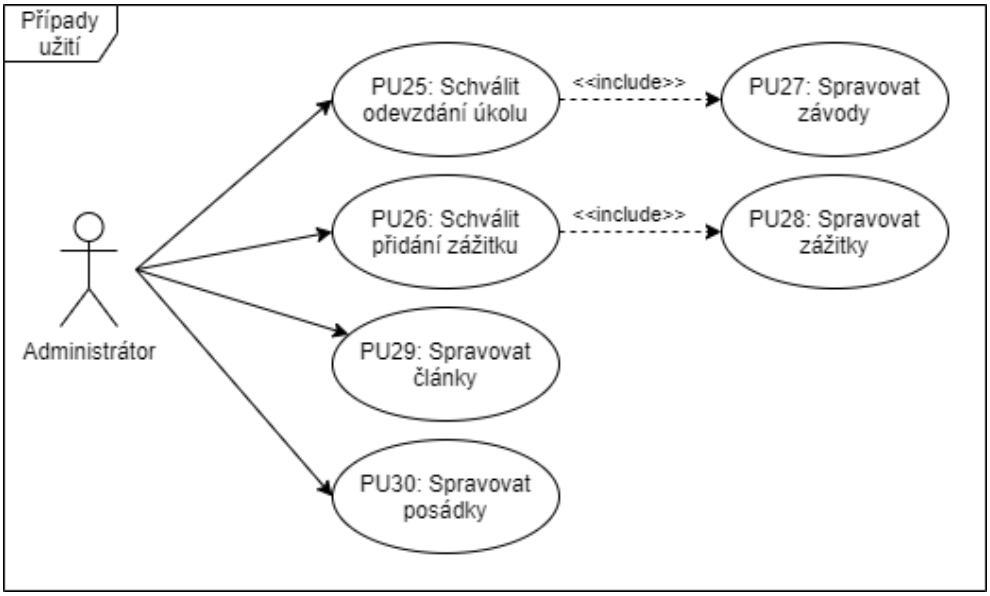
3. Aplikace zobrazí seznam všech posádek, jak aktivních, tak neaktivních.

4. Administrátor si zvolí posádku a akci, kterou na ní chce provést. V tomto případě upravit.

5. Aplikace zobrazí detail posádky jako formulář.

6. Po upravení dat ve formuláři administrátor odešle žádost o uložení změn.

7. Pokud byla zadaná data v pořádku, aplikace informuje administrátora o dokončení změn a přesměruje ho na seznam posádek. V opačném případě aplikace administrátora upozorní na špatně vyplněná pole formuláře.



Obrázek 2.3: Diagram případu užití pro administrátora

2.4 Kontrola pokrytí požadavků

Níže zobrazená tabulka 2.1 znázorňuje naplnění funkčních požadavků případy užití. Vzhledem k tomu, že se v každém sloupci vyskytuje alespoň jedna značka splnění, je evidentní, že všechny funkční požadavky byly namodelovanými případy užití pokryty. Tato skutečnost ukazuje, že aplikace bude splňovat veškeré funkcionality, které se po aplikaci požadují.

	FP1	**FP2**	**FP3**	**FP4**	**FP5**	**FP6**	**FP7**	**FP8**	**FP9**	**FP10**	**FP11**
PU1		✓									
PU2			✓								
PU3			✓								
PU4				✓							
PU5				✓							
PU6					✓						
PU7					✓						
PU8							✓				
PU9							✓				
PU10								✓			
PU11						✓					
PU12						✓					
PU13	✓										
PU14	✓										
PU15	✓										
PU16	✓										
PU17										✓	
PU18											✓
PU19	✓										
PU20	✓										
PU21	✓										
PU22	✓										
PU23				✓							
PU24					✓						
PU25										✓	
PU26										✓	
PU27										✓	
PU28										✓	
PU29										✓	
PU30										✓	

Tabulka 2.1: Tabulka splnění funkčních požadavků

2.5 Výběr technologií

Před samotným návrhem aplikace je nejprve potřeba vybrat technologie, pomocí kterých bude aplikace implementována nebo kterých bude využívat. V této sekci je popsáno, jaké technologie byly brány v potaz a které se nakonec použijí pro samotnou implementaci.

Výběr technologií je rozdělen do pěti částí:

- Serverová část aplikace,
- Klientská část aplikace,
- Databáze,
- Služby zprostředkovávající mapy a navigaci,
- Nasazení.

2.5.1 Serverová část

V této části bylo nutné nejdříve vybrat, v jakém programovacím jazyce bude aplikace napsána. Pro rychlejší vývoj webové aplikace je vhodné vybrat některý z webových frameworků, které celý proces značně zjednodušují. Proto byl jazyk vybírán zprvu podle toho, jaký pro něj existuje webový framework. Mezi dva hlavní kandidáty patřily programovací jazyky **C#** a **Python**, a to na základě **2020 Developers Survey**[6] společně s autorovou znalostí těchto jazyků. Po diskuzi s organizátory závodu a jejich spolupracovníky, kteří aplikaci po produkčním vydání budou spravovat, byl vybrán jazyk Python, neboť s ním mají větší zkušenosti.

Python je interpretovaný, objektově orientovaný programovací jazyk s dynamickou sémantikou. Je vhodný jak pro rychlý vývoj aplikací, tak pro skriptování. Má jednoduchou a snadno naučitelnou syntaxi, díky které se kód snadno čte a tím i lépe udržuje. Jelikož je to interpretovaný jazyk, odpadá zde krok kompilace, a proto je cyklus upravit-otestovat-vyladit velice rychlý.[7]

Po zvolení programovacího jazyka bylo na řadě vybrat webový framework, ve kterém bude aplikace implementována. Na základě **Python Developers Survey**[8] jsou zdaleka nejpopulárnějšími frameworky pro vývoj webových aplikací v Pythonu **Flask a Django.**

2.5.1.1 Flask

Flask je nenáročný webový framework WSGI aplikací[2]. Vznikl jako nadstavba nad Jinja2 (šablonový framework) a Werkzeug (serverový framework). Ze zá-

[2]WSGI - Web Server Gateway Interface. WSGI aplikace implementuje serverovou část WSGI rozhraní pro nasazení webových aplikací napsaných v Pythonu[9]

kladu ho tvoří pouze funkce pro práci s URL, HTTP a šablonami.[10] Vše ostatní programátor musí dodat ve formě rozšíření, které si sám vybere. Například, pokud chce programátor do projektu zahrnout práci s databází, musí sám vybrat nějaký ORM framework, např. FlaskSQLAlchemy. To poskytuje širokou škálu možností, ale také velký prostor pro chyby a nekompatibilitu mezi potřebnými rozšířeními. Vzhledem k jeho jednoduchosti je velice rychlé vyvinout v něm webovou aplikaci. Nicméně díky absenci jakékoli dané struktury může být pro nově příchozí vývojáře obtížné se v aplikaci zorientovat.

Výhody

- Rychlý vývoj aplikace,
- Absence robustní, předem dané struktury,
- Kompletní kontrola nad použitými rozšířeními.

Nevýhody

- Potencionální nekompatibilita potřebných rozšíření,
- U větších projektů horší přehlednost aplikace,
- Je nutná dobrá znalost velkého množství rozšíření pro docílení běžně potřebných funkcionalit webových aplikací.

2.5.1.2 Django

Na rozdíl od Flasku je Django velice robustní webový framework. Součástí základního balíčku je bohaté databázové API, dobře strukturované adresy URL, propracované administrační prostředí pro správu dat, vícejazyčnost a spoustu jiného. Díky tomu je možné v Djangu vyvíjet aplikace velice rychlým tempem, a jelikož je programátor nucen držet se stanovených struktur, výsledné aplikace dodržují principy dobrého návrhu. Django se snaží co nejvíce funkcionalit automatizovat a dodržovat princip DRY[3]. Django je mocný nástroj, který programátorovi umožňuje plně se soustředit na funkcionality vyvíjené aplikace a starosti s technickými záležitostmi přenechat na frameworku samotném.[11]

Výhody

- Velká část funkcionalit pro tvorbu webových aplikací je již ve frameworku implementována,
- Kvalitní administrační prostředí,

[3] DRY - Don't Repeat Yourself - neopakuj se

- Skvělá dokumentace a velice rozsáhlá aktivní komunita,
- Vestavěné middleware knihovny, které se starají o běžné bezpečnostní hrozby, např. SQL, injection[4].

Nevýhody

- Funkcionality, které nejsou plně podle principů frameworku Django, se implementují s velkými obtížemi,
- Potencionálně příliš mnoho nevyužitých souborů jako součást základního projektu.

2.5.1.3 Výběr frameworku

Po dlouhém zvažování byl vybrán framework Django. Jelikož autor práce neměl zkušenosti ani s jedním, dokumentace a rozsáhlá aktivní komunita byly hlavní faktory při rozhodování. Dalším důvodem byl fakt, že implementovaná aplikace využije velkou část základních funkcí frameworku, což značně usnadní a urychlí vývoj. V neposlední řadě díky struktuře a přehlednosti projektu nebude v budoucnu problém s ním seznámit správce závodu, kteří výslednou webovou aplikaci budou udržovat.

2.5.2 Klientská část

Pro tvorbu webových stránek budou použity technologie **HTML, CSS** a **JavaScript**. Aby nebylo potřeba upravovat veškeré HTML komponenty vlastními styly, používají se při vývoji webových aplikací buď předpřipravené šablony, nebo některou z CSS knihoven. Pro tuto práci byl zvolena knihovna **Bootstrap**. Pro pohodlnější a efektivnější práci s CSS soubory se používají CSS preprocessory, mezi které patří např. **LESS** a **SASS**. V této práci byl vybrán preprocessor SASS z důvodu zkušeností autora práce s danou technologií.

2.5.3 Databáze

Při výběru databáze musel být brán v potaz seznam podporovaných databází webovým frameworkem Django. Mezi ty patří[12]:

- PostgreSQL,
- MariaDB,
- MySQL,

[4] SQL injection - technika napadeni databázové vrstvy systému vložením vlastního SQL dotazu přes neošetřený vstup

- Oracle,
- SQLite.

Jelikož aplikace komunikuje s databázovou vrstvou skrze ORM, tak pro implementaci není výběr databáze podstatný, pokud se jedná o jednu z výše uvedených. Django umožňuje vyměnit tyto databáze bez většího zásahu do zdrojových kódů. Nakonec tedy byla z autorovy osobní preference vybrána databáze PostgreSQL.

PostgreSQL je objektově relační databázový systém, který využívá a rozšiřuje jazyk SQL. Tento systém je v aktivním vývoji již přes 30 let a je plně kompatibilní s vlastnostmi ACID od roku 2001. Běží nativně na všech rozšířených operačních systémech, jako je Linux, systémy UNIX a Windows. Kromě standardních SQL datových typů obsahuje i ty moderní, jako např. JSON nebo XML [13].

2.5.4 Služby zprostředkovávající mapy a navigaci

Z funkčních požadavků plyne, že musí být na mapě znázorněné polohy posádek a statistiky o nich, které se týkají ujetých vzdáleností a vzdáleností posádky od cíle. Pro splnění tohoto požadavku bylo nutné využít nějaké aplikace třetí strany, protože implementace takovéto služby je velice složitá, časově náročná a nad rámec této diplomové práce. Vzhledem k tomu, že závod pořádá nezisková organizace, tak byly při vybírání služby hlavním kritériem co nejmenší náklady při splnění požadavků.

2.5.4.1 Komerční řešení

Existuje mnoho komerčních společností, které potřebné služby poskytují. Jednou z největších takových společností je Google, která nabízí jak statické mapy, které se zobrazují jako obrázky, tak dynamické mapy, které jsou interaktivní a přizpůsobitelné. Z jejich nabídky směrování našim požadavkům vyhovuje pouze matice vzdáleností, která pro zadanou sekvenci světových souřadnic vrátí vzdálenosti a časy tras kombinací souřadnic v reálném provozu.

Společnost Google poskytuje uživatelům každý měsíc 200 $, za které může jejich služby využívat. Ceny za tyto služby v době psaní tohoto textu jsou zobrazeny v tabulce 2.2 [14].

Služba	**Částka za 1000 požadavků**
Statické mapy	5 $
Dynamické mapy	7 $
Matice vzdáleností	5 $

Tabulka 2.2: Tabulka cen služeb společnosti Google

2.5.4.2 Open source řešení

Open source řešení existuje ve dvou variantách. Buď je možné využít některého z veřejně dostupných API, nebo použít jejich zdrojové kódy pro vlastní nasazení dané služby. Výhodou tohoto řešení jsou nulové náklady a kromě limitovaného počtu požadavků, který je možné posílat na veřejně dostupné API, není toto řešení ničím omezeno.

2.5.4.3 Výběr

Kvůli absenci statistik ohledně návštěvnosti současných webových stránek nebylo možné odhadnout, jaké náklady by obnášelo použití služeb společnosti Google, jelikož celková cena závisí na počtu zobrazení mapy na úvodní stránce. Proto byla tato možnost zavržena.

Pro tuto práci bylo nakonec vybráno nasazení vlastního směrovacího stroje, konkrétně **Open Source Routing Machine**. Jelikož autorem nasazený směrovací stroj nebude veřejně dostupný, bude obsluhovat pouze požadavky z autorem nasazené webové aplikace. Díky tomu nebude provoz této služby natolik náročný, aby nemohla být nasazena na stejný server, jako samotná webová aplikace. To má za důsledek nulové náklady na provoz požadovaných služeb. Také díky tomuto řešení nebude nijak omezeno rozšiřování nebo upravování závodu, jako by tomu mohlo být v případě placených služeb a veřejně dostupných API.

2.5.5 Nasazení

Aby výsledná aplikace fungovala, musí být nasazeny 3 části: Databáze, Webová aplikace a OSRM směrovací stroj. V této práci byl využit nástroj Docker, který každou část obalí do jednoho kontejneru. Ty poté budou nasazeny do cloudu, a kontejner obsahující webovou aplikaci bude zpřístupněn veřejnosti.

Pro nasazení výsledné aplikace do cloudu byla zvolena platforma AWS[5] od společnosti Amazon. Tato platforma poskytuje bezplatnou infrastrukturu v míře, která je dostačující pro tuto práci.

Bylo zde využito dvou služeb: EC2 (Elastic Compute Cloud) pro spuštění Docker imagů a EBS (Elastic Block Storage) pro ukládání dat. Více informací o jejich nastavení a použití je popsáno v sekci 5.3.

Django obsahuje jednoduchý webový server, který slouží čistě pro lokální vývoj, a tak bylo potřeba vybrat některý z WSGI webových serverů. Pro tuto práci byl vybrán server **Gunicorn**. Poskytování statických souborů je ve frameworku Django pomalé. Aby se tomuto omezení předešlo, používá se pro poskytování statických souborů samostatně běžící webový server. Na základě doporučení v Django dokumentaci byl vybrán webový server **Nginx** [15].

[5] Amazon Web Services

Kapitola 3

Návrh

3.1 Architektura aplikace

3.1.1 Vícestránková aplikace (MPA)

Architektura vícestránkové aplikace je považována za tradiční způsob, jak vyvíjet webové aplikace. Ty jsou složeny z více stránek, které je při každém požadavku od uživatele potřeba celé obnovit. Uživatelův požadavek je zpracován v serverové části, kde se následně vyrenderuje celá HTML stránka a odešle se na klientskou část. Některé stránky mohou potencionálně obsahovat velké množství dat, což může vézt k pomalému načítání. Opakované načítání takové stránky způsobuje špatný uživatelský zážitek. Proto je někdy potřeba načítat data na stránku postupně, nebo obnovit pouze její část. Pro tento účel existují AJAX požadavky. Ty ze serveru získají data, která se následně vloží do stránek pomocí jazyku Javascrip [16].

Vícestránková aplikace má velice úzce spjatou klientskou a serverovou část. Většina logiky aplikace je napsána v serverové části a klientská část slouží především pro prezentaci dat a zachycování vstupů od uživatele.

Výhody

- Dobrá navigace v rámci aplikace,
- Nižší riziko bezpečnostních hrozeb,
- Rozšiřitelnost - snadné přidání nového obsahu,
- Jednoduchá implementace SEO[6],

[6]SEO, anglicky Search Engine Optimization - optimalizace pro vyhledávače zajišťují, aby se stránka zobrazovala na předních místech ve výsledcích vyhledávání.

- Možnost využití analyzátorů dat, které poskytuji data o fungování systému, chování uživatelů a dalších užitečných informací.

Nevýhody

- Úzce spjatá klientská a serverová část,
- Je nutné pro každou stránku psát vlastní HTML kód,
- Dlouhé načítání stránek.

3.1.2 Jednostránková aplikace (SPA)

Jak už název napovídá, jedná se o samostatnou stránku, která při prvním zobrazení načte do lokálního uložiště veškerý obsah na stranu klienta. Na rozdíl od vícestránkových aplikací není potřeba načítat celou stránku znovu při změně zobrazovaných dat. Informace jsou na stránku vykreslovány pomocí jazyka JavaScript. Serverovou část používají pouze jako zdroj a uložiště dat, se kterou komunikují přes serverem vystavené API. Tuto architekturu se vyplatí použít, pokud je nebo v budoucnu bude požadovaná aplikace se stejnou funkcionalitou jako webové stránky na jiné platformě, např. mobilní aplikace. Tu je poté snazší implementovat, protože může využít stejné serverové části jako webová aplikace, kdežto u vícestránkové aplikace je nutné buďto vystavit API v serverové části, nebo napsat pro novou aplikaci vlastní server [16].

Pro implementaci jednostránkové aplikace je tedy nutné vystavit API a implementovat její klientskou část, která se nejčastěji implementuje pomocí některého z JavaScript frameworku, jako např. **Angular, React.js nebo Vue.js**.

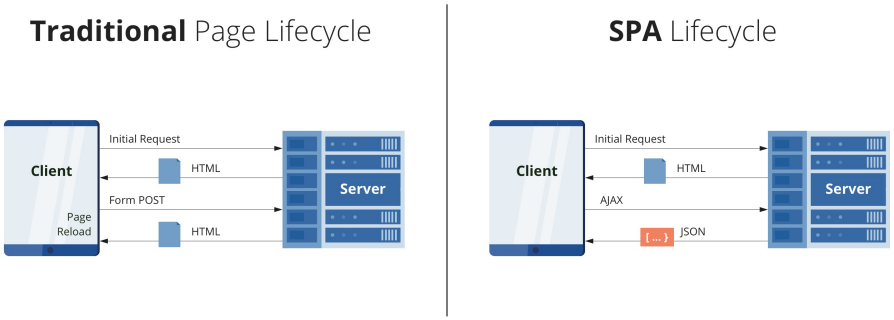
Výhody

- Po prvním načtení stránky rychlejší načítání obsahu - posílá se pouze část dat, nikoli celá stránka,
- Jsou multiplatformní - vzhledem vystavenému API může být serverová část použita pro více platforem, např. mobilní zařízení,
- Použití bez přístupu k internetu - jelikož je vše uloženo lokálně v prohlížeči, je možné některé funkcionality používat i bez připojení k internetu.

Nevýhody

- Složitá implementace SEO,
- Potencionálně dlouhé první načtení - musí se stáhnout veškeré JavaScript soubory do lokálního uložiště klienta,

- Statistiky - není možné využít statistické služby jako např. Google Analytics, protože se jedná pouze o jednu stránku. Pro tyto účely je tedy potřeba implementovat vlastní analytický nástroj.



Obrázek 3.1: Životní cyklus MPA vs. SPA [1]

3.1.3 Výběr rozdělení

Po důkladném zvážení obou možností bylo rozhodnuto implementovat aplikaci jako vícestránkovou. Hlavním důvodem pro tento výběr byl fakt, že výsledná aplikace nebude poskytovat rozsáhlé uživatelské rozhraní s mnoha funkcemi. Na stránkách pro uživatele bude obsah převážně pro čtení. V době psaní práce organizátoři neuvažovali ani o rozšíření aplikace na jinou platformu.

3.1.4 MVT (Model-View-Template) architektura

MVC[7] je jedním z architektonických vzorů pro vývoj webových aplikací. Odděluje řídící logiku od datového modelu a uživatelského rozhraní pomocí tří základních vrstev: Model, Pohled a Řadič. Výměna jedné vrstvy by ve výsledku měla co nejméně ovlivnit ty ostatní, a tím docílit lepší škálovatelnosti a rozšiřitelnosti.

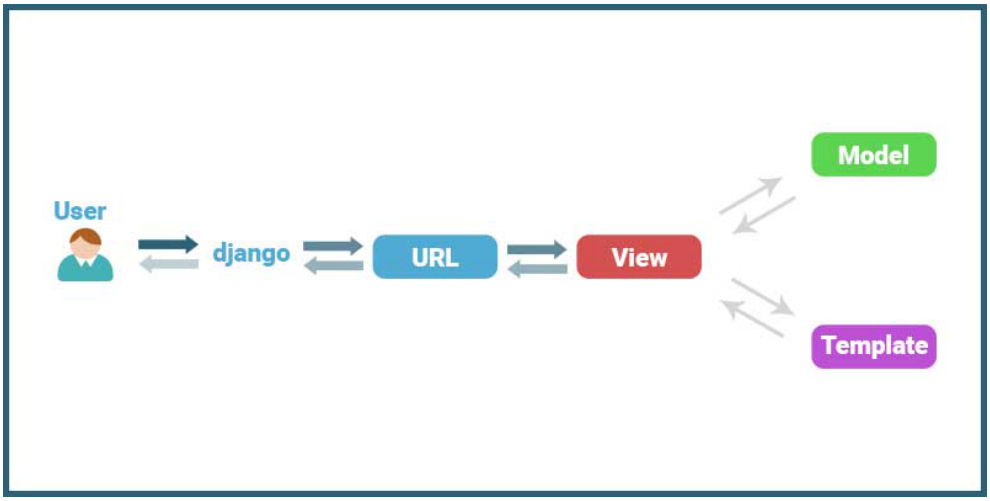
Webový framework Django implementuje MVT architekturu, která je od MVC odvozena. V architektuře MVT je vynechána vrstva řadič, o jejíž funkcionalitu se stará framework sám. Nicméně také rozděluje aplikaci do tří vrstev:

Model Model zajišťuje veškerou logiku ohledně dat a komunikaci s databází.

[7] MVC, anglicky Model View Controller

Pohled Ve frameworku Django stojí vrstva pohled mezi modelem a šablonou. Tato vrstva shromažďuje data od uživatele nebo z modelu a následně je pomocí šablony zobrazí.

Šablona Šablony se využívají ke specifikaci struktury výstupu. Šablona je mix HTML a DTL[8].



Obrázek 3.2: Znázornění architektury MVT

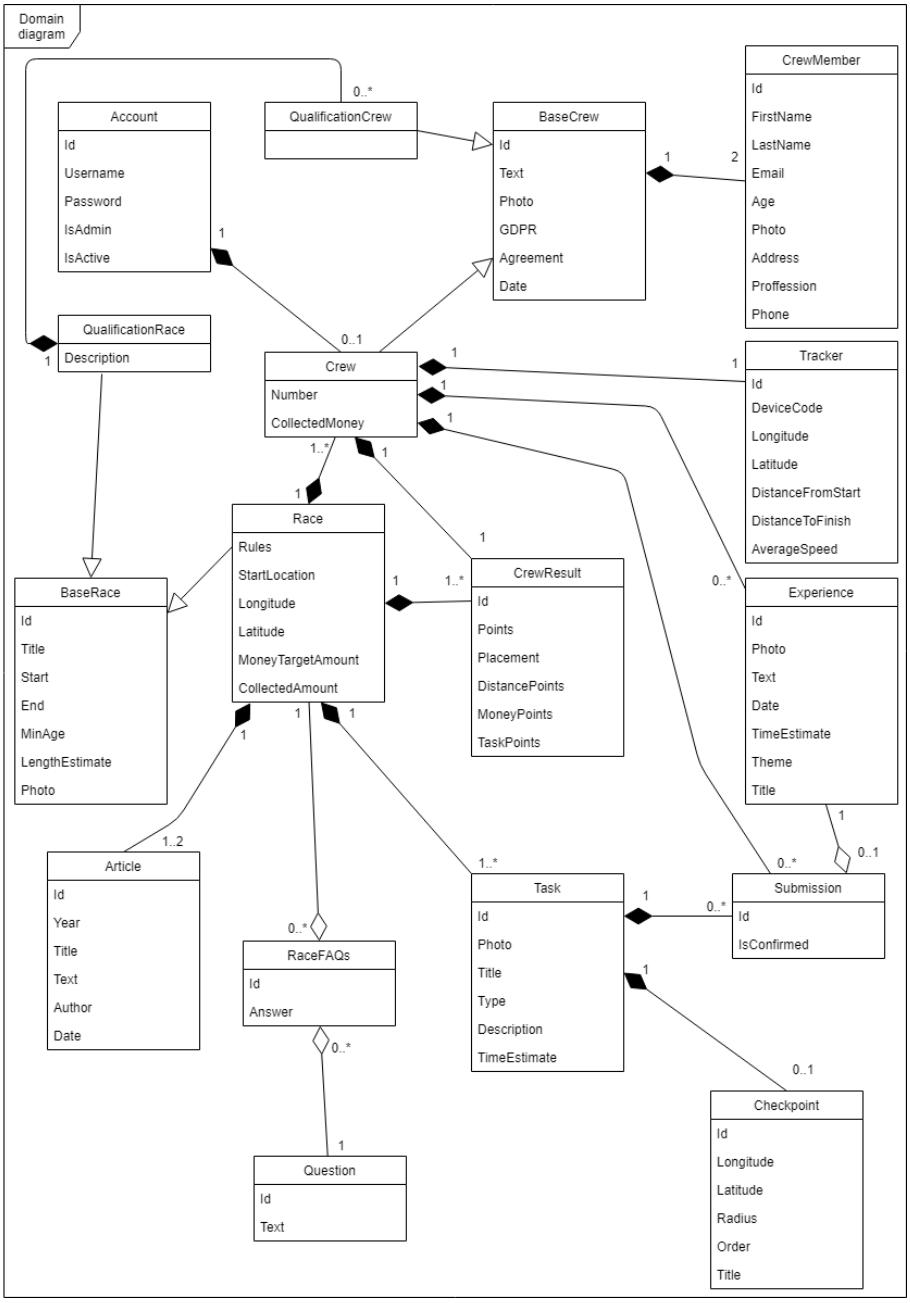
3.2 Doménový model

Doménový model vzniká po vydefinování případů užití a jeho úkolem je lépe pochopit a identifikovat jednotlivé subjekty v systému a vazby mezi nimi. Doménový model je odlehčenou formou diagramu tříd. Jeho základním prvkem je také třída, nicméně neobsahuje žádné implementační detaily (je implementačně nezávislý).

V diagramu 3.3 jsou použity tyto vazby [17]:

Asociace Asociace určuje základní vztah mezi entitami, které mohou existovat nezávisle na sobě. Tento vztah se v diagramu znázorňuje jednoduchou plnou čarou. V tomto vztahu mají obě entity odkaz na tu druhou. Pokud je na jednu stranu přidána šipka, odkaz na druhou entitu má již pouze ta, ze které šipka směřuje.

[8]DTL, anglicky Django Template Language - Django šablonový jazyk



Obrázek 3.3: Doménový model

Agregace Agregace znázorňuje vztah celku a části. V diagramu je znázorněna jednoduchou plnou čárou, na jednom konci zakončenou prázdným kosočtvercem. Kosočtverec se nachází na straně celku. Entita, která je celkem, může být také částí jiného celku.

Multiplicita Multiplicita, neboli násobnost je vyjádření, kolikrát instancí entity A může být ve vztahu s entitou B.

Kompozice Kompozice, stejně jako agregace, znázorňuje vztah celku a části. Avšak s větší závislostí. U kompozičního vztahu entita, která spadá do celku, nemá bez celku smysl. V diagramu se znázorňuje stejně jako agregace, ale s plným kosočtvercem. Multiplicita na straně kosočtverce musí být vždy 1.

Generalizace Generalizace znázorňuje vztah, kdy entita A dědí vlastnosti a chování entity B. V diagramu je tento vztah znázorněn jednoduchou plnou čárou ukončenou na jednom konci prázdným trojúhelníkem. Na tomto konci se nachází entita, ze které se dědí.

3.3 Uživatelské rozhraní

Na základě funkčních požadavků vznikl návrh uživatelského rozhraní. Tato sekce se zabývá celkovým vzhledem a popisem jednotlivých částí aplikace.

3.3.1 Vzhled

Pro celkový vzhled nebyla v této práci použita žádná šablona. Pro styl a animace HTML komponent byla použita CSS knihovna Bootstrap, jejíž styly byly mírně upraveny, aby se webová aplikace odlišila od ostatních webů, které knihovnu Bootstrap používají také.

3.3.2 Záhlaví

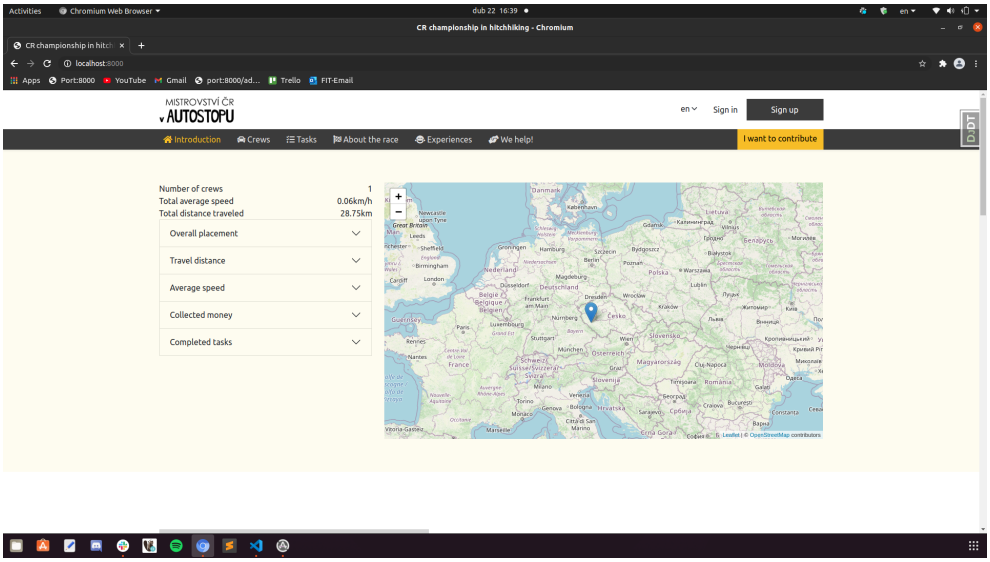
Pro přehlednější orientaci tvoří záhlaví dvě pod sebou umístěné lišty. V té první se nachází logo závodu, přepínání jazyku aplikace a akce spojené s profilem posádky: přihlášení, odhlášení, registrace a správa profilu. Druhá lišta slouží k navigaci mezi jednotlivými stránkami.

3.3.3 Hlavní část

3.3.3.1 Úvodní stránka

Úvodní a zároveň hlavní stránka aplikace slouží pro zobrazení výsledků závodu a také jako rozcestník na ostatní stránky.

Nejprve je zobrazen sloupeček s výsledky, který má vedle sebe mapu. Na mapě jsou zobrazené body, kde se v danou chvíli posádky nachází. Celou dobu jsou zobrazeny celkové statistiky závodu, jako například celkový počet ujetých kilometrů. Uživatel poté může kliknout na jednu z kategorií hodnocení, což mu zobrazí výsledky a pořadí posádek v dané kategorii. Při najetí myší na řádek s posádkou v tabulce se zvýrazní poloha konkrétní posádky na mapě.



Obrázek 3.4: Snímek obrazovky úvodní stránky

Pokud uživatel na řádku klikne, zobrazí se detailní informace ohledně dané posádky.

Dále je na stránce popis dalších částí webových stránek, např. text o plnění úkolů s odkazem na ně.

Hlavní stránka také zobrazuje počet vybraných peněz a cílovou částku, která byla tento rok stanovena. Nakonec jsou zde zobrazena loga hlavních partnerů závodu.

3.3.3.2 Posádky

Na této stránce je seznam posádek zobrazený formou dlaždic. Ty obsahují fotografii a základní informace o členech posádky. Uživatel má možnost otevřít detail každé posádky.

V detailu posádky je zobrazena stejná fotka jako na stránce se seznamem posádek. Uživatel zde může najít detailnější informace o členech posádky a také popis dvojice jako takové, který se vyplňuje při registraci. Na pravé straně je viditelný panel se stručným popisem, jak přispívat posádce, s odkazem na stránku s přispíváním.

V dolní části je pak seznam zážitků, které posádka přidala. Každý z nich obsahuje fotku, datum a prvních pár vět daného textu. Po kliknutí na článek se zobrazí vyskakovací okno, ve kterém je zobrazen celý článek zážitku.

3.3.3.3 Úkoly

Stejně jako u posádek se úkoly zobrazují na dvou stránkách: seznam všech úkolů a detail úkolu. Stránka se seznamem má stejné rozvržení jako stránka se seznamem posádek.

Na stránce detailu úkolu jsou informace o úkolu, jako je zadání, odhadovaný čas plnění a kolik posádek daný úkol splnilo. Vedle toho jsou vyobrazené miniatury profilových obrázků posádek, které úkol splnily. Ty slouží jako odkazy na stránku detailu dané posádky.

Na konci stránky je list zážitků, které popisují plnění zobrazeného úkolu. Každá položka seznamu obsahuje fotografii, autora s odkazem na detail posádky, datum, téma, odhadovaný čas čtení a prvních pár vět textu. Článek si lze přečíst stejně jako na stránce detailu posádky.

Pokud je uživatel přihlášen, má možnost pokusit se daný úkol odevzdat. Pokud již úkol odevzdal nebo nemá právo ho odevzdat, není mu tlačítko na odevzdání úkolu zpřístupněno. Pro odevzdání úkolu se zobrazí vyskakovací okénko, kde je zobrazen formulář pro odevzdání. Po odeslání formuláře se na stránce s detailem odevzdávaného úkolu zobrazí zpráva s informací o úspěšném odevzdání.

3.3.3.4 Zážitky

Na stránce zážitků je zobrazený seznam všech zážitků všech posádek, který je řešen stejně jako seznam zážitků na stránce **Detail posádky**. Uživatel si může filtrovat zážitky podle posádek.

3.3.3.5 O závodu

Na této stránce jsou veškeré informace o závodu s odkazem na stránku s pravidly závodu. Na stránce s pravidly závodu jsou zobrazeny často kladené otázky s odpověďmi.

3.3.3.6 Pomáháme

Na této stránce je zobrazen aktuální článek s informacemi o tom, na co se v závodě přispívá. Pod tímto článkem je zobrazena historie článků, která je rozdělena podle ročníků.

3.3.4 Zápatí

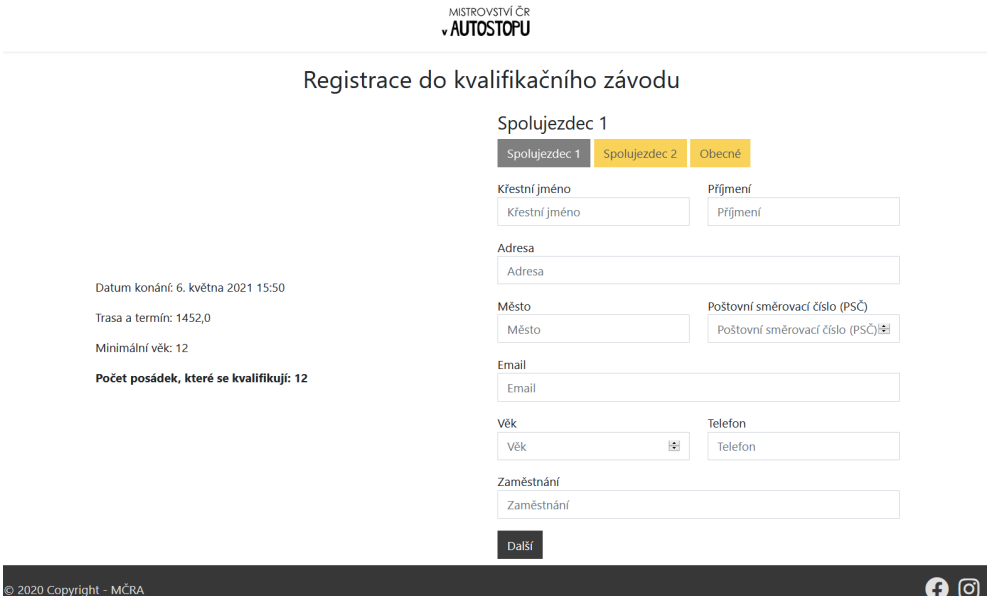
Zápatí obsahuje základní informace o webových stránkách (např. rok vzniku), kontaktní informace na pořadatele závodu, číslo bankovního účtu, na který mohou lidé přispívat a také odkazy na sociální sítě závodu.

3.3.5 Autentizace

Stránky, které zajišťují autentizaci, mají jiné celkové rozvržení stránek. Záhlaví je změněno pouze na logo závodu, které je současně odkaz zpět na klasické stránky.

3.3.5.1 Přihlášení

Pro přihlášení je uživateli zobrazen klasický formulář pro zadání uživatelského jména a hesla. Dále je zde poskytnuta možnost pro obnovení hesla, pokud ho uživatel zapomněl.



Obrázek 3.5: Snímek obrazovky registrace

3.3.5.2 Registrace

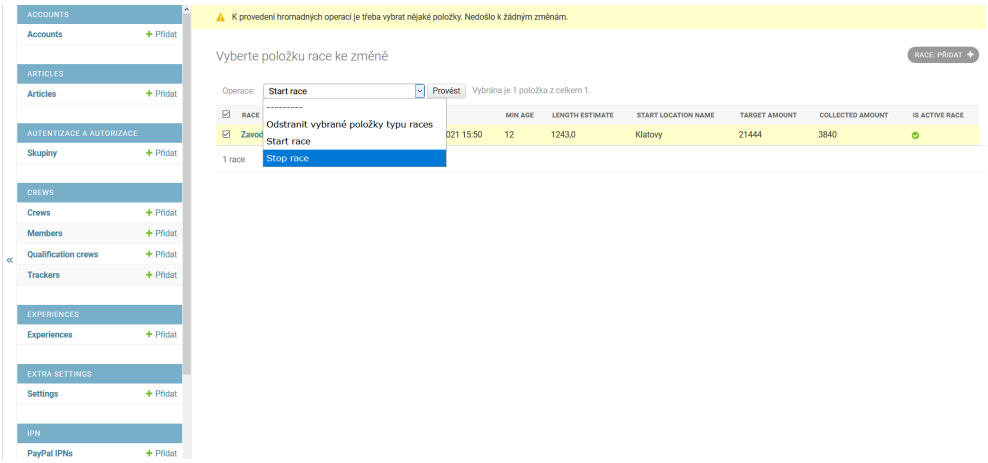
Registrace do závodu probíhá najednou pro celou posádku a je rozdělena do třech částí. První dvě části jsou totožné, u obou je zobrazen formulář pro vyplnění údajů o jednom ze spolujezdců. Poslední část je věnována informacím o posádce, kde je potřeba nahrát fotografii a napsat příběh, který popisuje posádku.

Mezi jednotlivými částmi registrace se dá navigovat pomocí zobrazených navigačních tlačítek.

Po úspěšné registraci je zobrazena stránka s informací o dokončení registrace.

3.3.6 Adminské prostředí

Webový framework Django přichází s kvalitním a přehledným administračním prostředím. Pro účely práce bylo vyhovující, proto návrh uživatelského rozhraní administrační části není součástí práce.



Obrázek 3.6: Snímek obrazovky administračního prostředí

3.3.7 Mobilní verze

Z nefunkčních požadavků plyne, aby webové stránky byly optimalizované i pro mobilní zařízení. Uživatelské rozhraní se mění v závislosti na rozlišení obrazovky, na kterém je zobrazeno. Při zobrazení na mobilním zařízení se některé prvky, které jsou ve výchozím rozložení vedle sebe, zobrazí pod sebou, např. mapa na hlavní obrazovce se na mobilním zařízení zobrazuje až pod tabulku výsledků. Stejně tak se mění i záhlaví stránky. Na mobilním zařízení je vyobrazena pouze horní lišta záhlaví (tlačítka jsou pod sebou, nikoli vedle sebe) a místo druhé lišty je tlačítko, které po stisknutí zobrazí vyjíždějící menu s odkazy.

Kapitola 4

Implementace

4.1 Struktura projektu

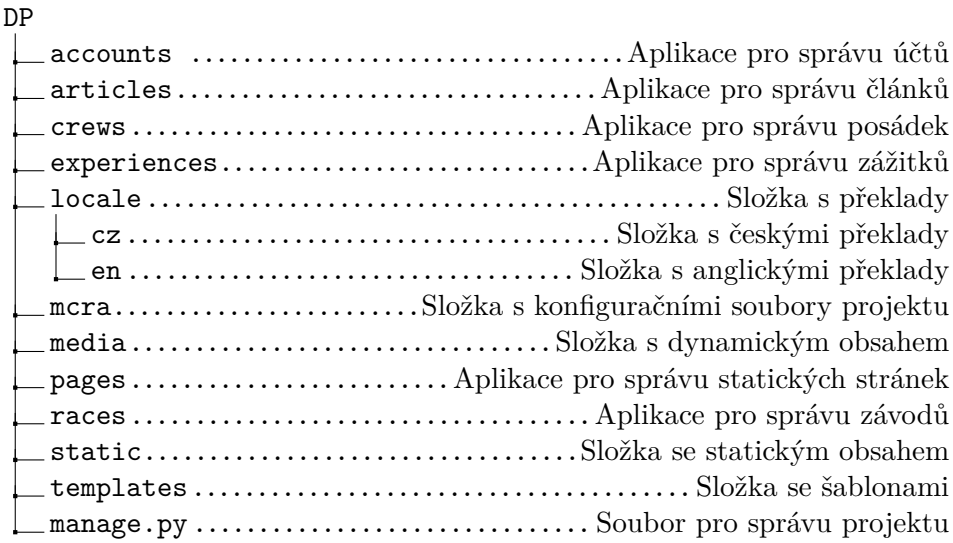
Webový framework Django vyžaduje pevně danou základní strukturu projektu. Hlavním prvkem projektu této práce je složka **mcra**, která obsahuje konfigurační soubory projektu. Pro větší přehlednost je projekt rozdělen do několika tzv. **aplikací**, které definují funkcionality ohledně logických celků projektu. Složka **locale** obsahuje překlady statického obsahu aplikace, v případě této práce anglický a český. Složka **static** obsahuje veškerá statická data aplikace. Naopak složka **media** slouží k ukládání dynamicky nahrávaných dat, jako například obrázků. Ve složce **templates** jsou veškeré šablony aplikace. Celý projekt je spravován souborem **manage.py**, který obsahuje například funkce pro vytvoření migrací, uživatelů atp.

Struktura hlavních prvků projektu je zobrazena na obrázku 4.1. Některé méně důležité soubory byly pro větší přehlednost vynechány.

4.2 Struktura jednotlivých aplikací

Django také také přichází se strukturou jednotlivých aplikací, jejíž nejdůležitější části jsou popsány v této sekci. Mezi ty patří:

- Modely,
- Pohledy,
- Šablony,
- Směrování,
- Admin.



Obrázek 4.1: Caption

4.2.1 Modely

Pomocí modelů Django vytváří tabulky, jejich atributy a omezení, které jsou v modelu uvedeny. Do tabulek relační databáze jsou modelové třídy nahrány pomocí **databázových migrací**.

Každý model musí dědit od třídy django.db.Model. Atributy jsou definovány jako třídní proměnné, kde každá z nich má svůj datový typ. Základní datové typy jsou přímými potomky třídy django.db.models.Field. Mezi takové patří například django.db.models.CharField nebo django.db.models.IntegerField. Django však poskytuje i datové typy, které rozšiřují funkcionalitu těchto základních typů. Příkladem je django.db.models.EmailField, který automaticky nastaví validátor e–mailu na tento atribut. Pokud ani tyto rozšířené možnosti nestačí požadavkům, existuje spousta rozšíření třetích stran, které poskytují více datových typů. V této práci byl použit datový typ ckeditor.fields.RichTextField, který se stará o formátovaný text. Uživateli je na stránce zobrazen textový editor, kde uživatel může psaný text formátovat a následně uložit do databáze. Dalším užitečným datovým typem, který byl v práci použit, je django.db.models.ImageField, který se stará o ukládání obrázků na server a do databáze uloží cestu k nahranému souboru.

Vztahy mezi modely jsou definovány pomocí tří atributů [18]:

- django.db.models.OneToOneField - 1:1,
- django.db.models.ForeignKey - 1:N,
- django.db.models.ManyToManyField - M:N.

4.2.1.1 Databázové migrace

Databázové migrace jsou ve frameworku Django způsob, jak propagovat změny modelových tříd do databázového schématu. Jsou připraveny tak, aby byly co nejvíce automatické. Nová migrace vytvoří soubor, ve kterém jsou uloženy veškeré informace o tom, jaké změny se v jakém modelu staly. Tyto soubory Django ukládá do databáze, aby se dalo mezi migracemi přepínat a aby byl někde seznam všech změn modelů. Každá aplikace má svůj vlastní soubor s migracemi, aby byly stále co nejvíce odděleny pro případné samostatné využití v jiném projektu.

Migrace se vytvářejí pomocí příkazů, které jsou definované v souboru **manage.py**. Nejprve je třeba spustit příkaz **python manage.py makemigrations**, na základě kterého se ve všech aplikacích vytvoří nový migrační soubor, pokud v modelech dané aplikace proběhla nějaká změna. Následně příkazem **python manage.py migrate** se aplikují migrace do databázového schématu. Pokud nějaká změna vyžaduje další informace, programátor je může zadat do příkazové řádky, kde migraci spustil. Takovou informací může být například výchozí hodnota u atributu [19].

4.2.1.2 Django databázové API

Po vytvoření modelových tříd je programátorovi automaticky k dispozici databázové API, které umožňuje CRUD operace nad databází [20].

Pro vytváření záznamu v databázi stačí vytvořit instanci modelové třídy. To samo o sobě neodešle žádné informace do databáze. Pro vyvolání požadavku na vytvoření záznamu v databázi je nutné použít metodu **save()**, která na na pozadí provede SQL příkaz **INSERT**. Pro vytvoření a uložení objektu najednou lze použít metodu **create()**. Metoda **save()** se také používá pro uložení změn instance, která již do databáze vložena byla.

Pro získávání objektů z databáze používá Django databázové API třídy **QuerySet** a **Manager**.

QuerySet reprezentuje množinu objektů v databázi a může obsahovat filtry, které danou množinu zužují, na základě jejich parametrů. Ekvivalent QuerySetu v SQL jazyce je **SELECT** s filtry jako **WHERE** a **LIMIT**. Programátor má přístup ke QuerySetu skrze třídu **Manager**.

Manager je hlavním zdrojem QuerySetů pro model. Ten ze základu obsahuje jednoho Managera se jménem **objects**. Pomocí toho je programátor schopen dostat QuerySet se všemi záznamy daného modelu v databázi.

Filtrování a vyžádání QuerySetu samo o sobě nevyvolá žádný SQL dotaz, který by se aplikoval na databázi. Dotaz je do databáze odeslán až po samotném zeptání se na obsah dat (např. vypsání některého z atributů modelu). Při načítání dat z databáze se s modelem nenačítají související entity přes

cizí klíče. Pokud entita obsahuje pole souvisejících entit o velikosti N, které by chtěl programátor vypsat, do databáze se odešle N dotazů. Tento problém se nazývá **N + 1** [21], který se ve frameworku Django řeší použitím metod **select_related a prefetch_related**.

select_related následuje cizí klíče a načte související entity v rámci původního dotazu. Díky tomu není potřeba nadbytečných N dotazů do databáze. Tato metoda však nelze použít pro vztah M:N.

prefetch_related narozdíl od **select_related** je možné použít pro vztah M:N, jelikož pro každý vztah modelu zavolá vlastní dotaz do databáze a výsledné spojování probíhá na straně serveru v Pythonu.

4.2.1.3 Signály

Django implementuje tzv. signály, které pomáhají odděleným aplikacím dostávat upozornění, že se stala nějaká akce kdekoli jinde v aplikaci. Pro to, aby byl signál přijat, se musí zaregistrovat přijímací funkce pomocí použitím funkce Signal.connect(). Tato funkce se zavolá pokaždé, kdy je odeslán signál, ke kterému se zaregistrovala [22].

Django poskytuje několik předdefinovaných signálů, které jsou programátorovi k dispozici. Jedním z nich je signál django.db.models.signals.post_save(). Tento signál je odeslán ve chvíli, kdy je zaregistrovaná entita uložena do databáze. V této práci je použit například pro vytvoření účtu pro posádku, která se kvalifikovala do hlavního závodu. Po obdržení signálu se spustí funkce, která dané posádce vytvoří účet pro přihlášení do aplikace a odešle posádce e–mail s potřebnými informacemi.

```
1  def crew_account(sender, instance, created, **kwargs):
2      if created:
3          password = Account.objects.make_random_password()
4          user = Account.objects.create_user(str(instance), password,
5                                             instance)
6          emails = (instance.member_one.email,
7                    instance.member_two.email)
8          EmailSender.send_template(emails, "Account credentials",
9                                    "create_account.html",
10                                   {'username': user.username,
11                                   'password': user.password})
12
13 post_save.connect(crew_account, sender=Crew)
```

Listing 4.1: Signál pro založení účtu po vytvoření posádky

4.2.2 Pohledy

Pohledy jsou v rámci frameworku Django funkce, které jako vstupní parametr přijímají webový požadavek společně s dalšími možnými parametry z URL dotazu a vrací webovou odpověď. Tou může být HTML kód, přesměrování, obrázek, JSON dokument nebo cokoli jiného. V rámci této funkce je definovaná veškerá logika, která je potřeba k připravení a odeslání odpovědi.

Django poskytuje několik předpřipravených odpovědí. V této práci jsou nejčastěji použity tyto:

- django.shortcuts.render - Tato funkce přijímá parametry: požadavek, název šablony, která se má vykreslit, a kontext. Kontextem je objekt, který obsahuje data potřebná k vykreslení šablony.
- django.shortcuts.redirect - Tato funkce slouží k přesměrování na jiný pohled aplikace, jehož jméno je do funkce předáno jako parametr.
- django.http.HttpResponseBadRequest - Klasická HTTP odpověď se status kódem 400 .
- django.http.HttpResponseNotFound - Klasická HTTP odpověď se status kódem 404.
- django.http.JsonResponse - Tato funkce přijímá data, která mají být serializována do JSON objektu, který následně odešle jako odpověď. Tato funkce se v práci používá pro posílání dat v rámci AJAX dotazů.

Pokud je potřeba zobrazovat na stránce pouze list instancí modelu nebo stránku detailu modelu, je možné vytvořit třídu, která dědí od těchto tříd:

django.views.generic.ListView V dané třídě je nutné definovat model, jehož instance mají být zobrazeny v listu, a název šablony, do které se mají data zobrazit. Množinu zobrazovaných objektů je možné specifikovat zadáním QuerySetu, ze kterého se mají data nahrát z databáze.

django.views.generic.DetailView Stejně jako u ListView je potřeba definovat model, šablonu a je také možné zadat QuerySet. V požadavku na použití této třídy je potřeba zadat identifikátor objektu, jehož detail má být zobrazen. Pokud chce programátor upravit objekt, který je do šablony poslán, může toho docílit přepsáním funkce **get_object()**. Pro doplnění kontextu o jiná data je možné přepsat funkci **get_context_data()**, ve které se přidají do aktuálního JSON objektu reprezentující kontext.

4.2.3 Šablony

Šablony jsou způsob, jak framework Django dynamicky generuje HTML obsah. Šablona se skládá ze statické části tvořené klasickým HTML kódem a speciální

synataxí, která definuje, jak se má vkládat dynamický obsah. Ta se nazývá **Django Template Language (DTL)**[9]. Tento obsah se skládá ze čtyř základních komponent [23]:

Proměnné Pomocí této komponenty se zobrazují hodnoty proměnných nebo kontextu. Syntaxe proměnných je **{{ proměnná }}**

Tagy Tagy mají v šablonách mnoho využití. Mohou generovat text, zajišťují možnost používání cyklů a podmínek nebo nahrávají do šablony externí informace. Syntaxe tagů je následující: **{% tag %}**. Některé tagy potřebují i ukončovací tag, např. for-cyklus: **{% for %} logika cyklu {% endfor %}**.

Filtry Pro upravování zobrazení proměnných DTL využívá tzv. filtrů. V práci se využívá filtr **safe**, který vypne escapování[10] znaků a tím je možné zobrazit formátovaný text, který je do šablony poslán jako řetězec obsahující HTML kód. Syntaxe filtru je následující: **{{ proměnná|safe }}**.

Komentáře Syntaxe komentářů je: **{# komentář #}**. Pokud je potřeba komentáře na více řádek, je možné použít tag **comment**.

V ukázce kódu 22 je zobrazen základ kostry pro znázornění použití DTL. Celá šablona je k nalezení v příloze (TODO).

4.2.3.1 Statické soubory

Veškeré statické soubory jsou uloženy ve složce **mcra/static**. Pro produkční prostředí je potřeba pomocí příkazu **python manage.py collectstatic** přesunout statické soubory do složky **static** v kořenovém adresáři projektu, ze které je bude příslušná služba poskytovat. V této práci statické soubory poskytuje webový server Nginx.

Aby bylo možné statické soubory používat v šablonách, je nutné je pomocí tagu **{% load static %}** do šablony nahrát. Poté už je možné ke statickým souborům přistupovat pomocí tagu **{% static 'název statického souboru' %}**.

4.2.3.2 Dědičnost

Jednou z užitečných vlastností Django šablon je dědičnost. Pomocí té jde napsat kostru stránky, na které se definují bloky. Šablona, která od dané kostry dědí, může bloky přepsat a tím vytvořit specifický obsah stránky. Pokud některý z bloků v šabloně nebude přepsán, použije se blok z rodičovské šablony. Syntaxe dědění např. od šablony **base.html** je: **{% extends**

[9] DTL, anglicky Django Template Language - Django šablonovací jazyk

[10] Escaping je metoda, která nám umožňuje říci počítači, aby ignoroval speciální funkci znaku [24].

"base.html"%} [23] V této práci sou vytvořeny dvě kostry. Jedna pro stránky zabývající se autentizací a druhá pro ostatní. V těchto kostrách je nahrán všechen obsah, který mají stránky společný, včetně společných referencí na styly a souborů JavaScript.

```
{% load static %}
{% load  i18n %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    {% block stylesheet %}
    {% endblock %}
    <title>{% trans 'CR championship in hitchhiking' %}</title>
</head>
<body>
    {% block content %}
    {% endblock %}
</body>
</html>
<script src="{% static 'js/main.js' %}"></script>
<script src="{% static 'js/bootstrap.js' %}"></script>
<script src="{% url 'javascript-catalog' %}"></script>
{% block js_block %}
{% endblock }
```

Listing 4.2: Základ kostry pro šablony implementovaných webových stránek

4.2.4 Adresy URL

Každá aplikace projektu obsahuje soubor **urls.py**, ve kterém jsou definované veškeré URL adresy aplikace. URL adresa zde přímo odkazuje na jeden pohled. Tento soubor je potom nahrán do souboru **mcra/urls.py**, kde musí být nahrány veškeré URL adresy projektu, aby framework Django mohl obstarávat směrování.

4.3 Autentizace

Framework Django poskytuje autentizaci uživatelů jako výchozí funkcionalitu. Avšak pro účely této práce musela být upravena. Posádku tvoří dva závodníci a přihlašují se za účelem odevzdání úkolů nebo psaní zážitků posádky. Z tohoto důvodu bylo rozhodnuto, že bude účet pro posádku, nikoli pro jednotlivé závodníky. Ve výchozím nastavení autentizace ve frameworku Django používá pro přihlašování e–mail. Toto nastavení bylo potřeba změnit na přihlašování pomocí uživatelského jména, čehož bylo docíleno napsáním vlastního přihlašovacího backendu, který dědí od výchozího **django.contrib.auth.backends.ModelBackend**. Zde bylo nutné přepsat funkci **authenticate**, kde

bylo pro nalezení účtu použito uživatelské jméno. Dále bylo potřeba vytvořit vlastní model pro přihlášené uživatele. Ti jsou dvojího typu: posádka a admin. Výchozí třída již obsahuje proměnné **is_admin** a **is_superuser**, podle kterých framework Django pozná, zda se jedná o admina či nikoli. Bylo tedy potřeba vyřešit, jak do účtu zahrnou informace o posádce, aniž by tyto informace vyžadoval i účet typu **admin** nebo **superuser**. To bylo vyřešeno pomocí samostatného modelu **Crew**, který slouží jako profil daného účtu. Ten v případě **admina** a **superusera** neexistuje. Aby framework Django používal vlastně definované třídy, bylo nutné je explicitně zadat v souboru **mcra/settings.py**.

4.4 Překlady

Pro překlady webových stránek do více jazyků bylo nutné provést tyto kroky:

1. V souboru **mcra/settings.py** nastavit, kam se mají vygenerované soubory pro překlad ukládat, jaké jazyky bude aplikace podporovat a jaký je výchozí jazyk.
2. Všechny řetězce, které mají být přeloženy předat do funkce pro přeložení. V Python a Javascript kódu je v této práci použita funkce **gettext**. V šablonách je místo funkce použit tag **{% trans %}**
3. Pomocí příkazu **django-admin makemessages** vygenerovat soubory s příponou **.po**. Pro každý podporovaný jazyk se vygeneruje jeden takový soubor, ve kterém je potřeba doplnit překlady řetězců označených v bodě 2.
4. Po přeložení všech řetězců je potřeba soubory s koncovkou **.po** zkompilovat do binárních souborů s koncovkou **.mo**, a to příkazem **djano-admin compilemessages**.
5. Do navigační lišty umístit výběr požadovaného jazyka.

Po splnění všech kroků byla aplikace schopná zobrazovat obsah ve více jazycích, konkrétně v angličtině a češtině. Pro přidání dalšího podporovaného jazyka je nutné pouze zopakovat kroky 3 a 4 s tím, že v kroku 3 je nutné doplnit překlady jen pro nový jazyk. Při prvním spuštění webových stránek se aplikace pokusí zjistit, jaký jazyk má nastavený prohlížeč, ze kterého požadavek přišel. Pokud tuto informaci získá a daný jazyk je mezi podporovanými, pak bude nastaven jako výchozí. V opačném případě bude použit výchozí jazyk ze souboru **mcra/settings.py**.

4.5 Odesílání e–mailů

Aplikace musí být schopná odesílat informace organizátorům a závodníkům pomocí e–mailů. Proto bylo nutné vybrat nějakou službu, která odesílání

umožňuje. Vzhledem k tomu, že bude aplikace nasazena na serverech AWS, nabízela se možnost využít jejich službu **Amazon Simple Email Service (SES)**. Jelikož služba, která se v aplikaci použije je snadno zaměnitelná, bylo rozhodnuto SES pro implementaci použít. Hlavním důvodem byla autorova možnost prozkoumat a naučit se používat další z AWS služeb. SES je cenově efektivní, flexibilní a škálovatelná e–mailová služba, která umožňuje vývojářům odesílat e–maily z jakékoli aplikace. Pomocí Amazon SES můžete e–maily posílat bezpečně a globálně[25]. Po založení má účet status **SANDBOX**. To znamená, že není možné posílat e–maily na jakoukoli adresu ani z jakékoli e–mailové schránky. V tomto stavu je možné používat pro přijímání i odesílání pouze ověřené účty, zadané ve vývojářské konzoly. V době psaní této práce ještě účet vyvíjené aplikace neprošel schvalovacím procesem a má tedy status **SANDBOX**. Proto byly pro účely testování zřízeny speciální e–mailové schránky, které testované osoby mohou využívat. Vzhledem k tomu, že existuje pouze jedno GPS zařízení, není potřeba více než dvou posádek. Jednu s GPS zařízením a jednu bez ní, aby se otestovala funkčnost aplikace. Proto byly vytvořeny pouze čtyři e–mailové schránky pro účty posádek a jedna pro samotnou aplikaci. Jejich přihlašovací údaje jsou uvedeny v uživatelské příručce.

Po získání ověření účtu aplikace je možné odesílat e–maily z jakékoli schránky na jakoukoli adresu. Pokud je aplikace nasazená na **Amazon EC2** nebo pomocí **AWS Lambda**, což je případ této práce, je možné odeslat zdarma 62000 e–mailů za měsíc [26]. Vzhledem k tomu, že aplikace odesílá –emaily primárně při registraci, obnově hesla nebo odevzdání úkolu, je nepravděpodobné, že by danou hranici přesáhla.

4.6 Platby

Z funkčních požadavků vyplývá, že má aplikace uživateli umožňovat platit třemi způsoby: DMS, PayPal, platební karta. V následujících podsekcích je popsána integrace těchto platebních metod.

4.6.1 DMS platby

Placení pomocí DMS nevyžaduje žádnou interakci uživatele a webové stránky. Jde o odeslání SMS ve specifickém tvaru na předem dané číslo. Tyto údaje uživatel může najít na webových stránkách. Několik dní před spuštěním závodu jsou zřízeny počítadla pro posádky závodu. Ty mají za úkol uschovávat informace o příchozích SMS, podle kterých je možné zjistit, kolik přidružené posádce bylo posláno peněz. Informace z počítadel jsou dostupné přes vystavené API, které odesílá dva různé formáty. Buďto odešle pouze číslo reprezentující celkovou vybranou sumu, nebo detailní výpis, ve kterém jsou vybrané sumy rozděleny podle typu poslané SMS.

V aplikaci je vybírání peněz pomocí DMS simulováno automatickým přičítáním peněz posádce v přesně definovaném intervalu, protože v době psaní práce nebyl aktivní žádný závod a autorovi nebylo poskytnuto testovací API. V reálném provozu by ve stejném intervalu byly odesílány dotazy pro získání celkové vybrané sumy, která by se posádce přidělila. Proto je přičítání peněz adekvátní simulace.

4.6.2 PayPal platby

Přijímání plateb přes službu PayPal vyžaduje založení firemního účtu na jejich stránkách. Po založení účtu byla služba integrována pomocí připojitelné Django aplikace **django-paypal**, která zajišťuje komunikaci se službami **PayPal Payments** a **Payments Pro** [27].

Pokud uživatel zvolí platbu pomocí služby PayPal a určí částku, kterou chce odeslat, je přesměrován na stránky PayPalu, kde po přihlášení do svého PayPal účtu provede samotnou platbu. Částku je možné zaplatit pomocí zůstatku na PayPal účtě nebo kreditní kartou. Po dokončení platby je přesměrován zpět do aplikace. Ve chvíli kdy PayPal platbu zpracuje, odešle do aplikace **Instant Payment Notification (IPN)**[11] [28]. Tato informace se uloží do databáze a současně se aktualizuje počet vybraných peněz posádkou, na kterou byla tato platba poslána.

Pro účely vývoje aplikace je zpřístupněno testovací prostředí, pomocí kterého jde aplikaci testovat bez posílání reálných peněz, tzv. **PayPal sandbox** [29]. Toto prostředí je využito i v této práci a přihlašovací údaje jsou k dispozici v uživatelské příručce.

4.6.3 Platby platební kartou

Z funkčních požadavků plyne, že má aplikace podporovat platby platební kartou. V rámci této práce proběhla komunikace skrze organizátory s jejich bankou ohledně platebních bran, které jsou potřeba pro přijímání takových plateb. Z komunikace vyplynulo, že bankou podporovaná platební brána je pro účely závodu příliš drahá. Organizátoři tedy dospěli k závěru, že by pro ně byl tento způsob platby spíše ztrátový, a proto není v této práci implementován.

4.7 Odevzdávání úkolů typu Místo

Během závodu je potřeba splnit několik úkolů a některé z nich jsou typu **místo**. Takové úkoly mají přesné souřadnice a rádius, ve kterém je možné úkol splnit. Aby mohla posádka pokračovat v cestě a zážitek o splnění úkolu odeslat později, existuje pro ně možnost **Navštívit místo**. Tato akce určí,

[11] IPN - detailní informace o provedené platbě

zda se posádka nachází v povolené vzdálenosti od místa plnění, a pokud ano, započítá se jim tento bod jako navštívený.

Jelikož směrovací stroj OSRM je nastaven na hledání cest pro automobily, není vhodné ho použít pro tento typ určování vzdálenosti. Je možné, že dané místo bude ležet mimo pozemní komunikace, a pokud by byl rádius pro odevzdání úkolu dost malý, nebylo by možné dané místo podle směrovacího stroje OSRM nikdy navštívit.

Proto byl v práci použit tzv. **Haversine vzorec** [30], který slouží pro výpočet přímé vzdálenosti mezi dvěma body na mapě. Ty jsou určeny zeměpisnou šířkou a délkou. Jedno omezení tohoto vzorce je, že je v něm Země uvažována jako dokonalá koule. Nicméně pro účely této práce, kde budou vypočítávané vzdálenosti maximálně do několika jednotek kilometrů, je jeho přesnost dostačující. Funkce pro výpočet vzdálenosti je zobrazen v ukázce kódu 9.

```
1 def haversine(lat1, long1, lat2, long2):
2     lat1, long1, lat2, long2 = map(radians, [lat1, long1, lat2,
      long2])
3
4     dlong = long2 - long1
5     dlat = lat2 - lat1
6     a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlong/2)**2
7     c = 2 * asin(sqrt(a))
8     return c * 6371
```

Listing 4.3: Výpočet vzdálenosti dvou bodů pomocí Haversine vzorce

4.8 OSRM směrovací stroj

Jedná se o výkonný směrovací stroj napsaný v C++14 a je navržený pro používání dat z **OpenStreetMap** [31]. Tento stroj je dostupný ve dvou formách: Zdrojové kódy v C++ a Docker image. Jelikož nebylo v plánu zdrojové kódy nijak upravovat a výsledná aplikace bude nasazena do cloudu pomocí Dockeru, byla zvolena forma Docker image. Pro zprovoznění OSRM směrovacího stroje je potřeba provézt následující kroky (uvedené příkazy byly použity v této práci):

1. Stáhnout Docker image **osrm/osrm-backend**.
2. Stáhnout výpisy dat OpenStreetMap.
   - **wget http://download.geofabrik.de/europe/czech-republic-latest.osm.pbf**
3. Předzpracovat stažené výpisy pro jeden ze tří profilů: auto, kolo a chodec.
   - **docker run -t -v ”$PWD:/data”osrm/osrm-backend osrm-extract -p /opt/car.lua /data/czech-republic-latest.osm.pbf**

- **docker run -t -v "$PWD:/data"osrm/osrm-backend osrm-partition /data/czech-republic-latest.osrm**
- **docker run -t -v "$PWD:/data"osrm/osrm-backend osrm-customize /data/czech-republic.osrm**

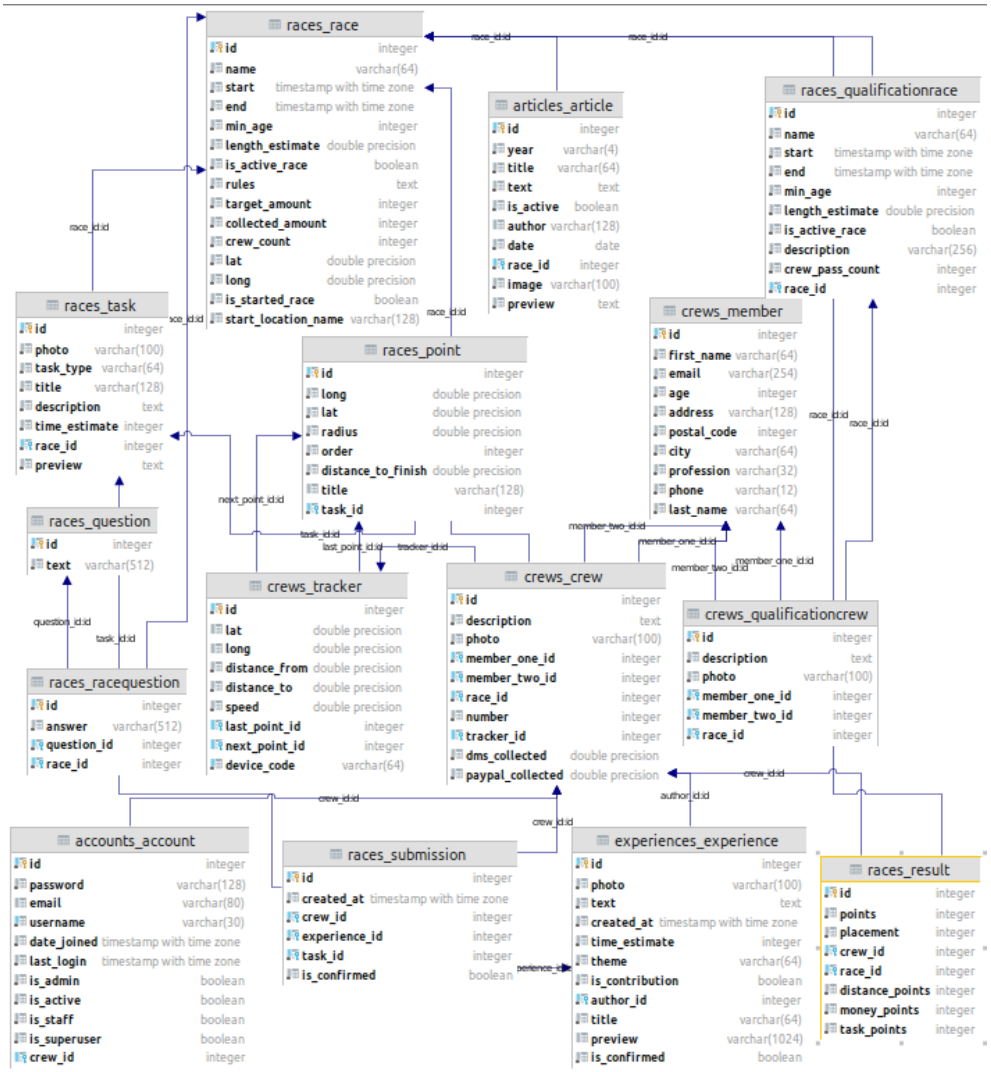
4. Spustit Docker image za přítomnosti předzpracovaných dat.

Výše uvedené řešení funguje pouze pokud pouštíte příkaz ve složce, která data obsahuje. Tento postup byl použit pro lokální vývoj, kdy nebyly služby spouštěny přes Docker kontejnery. V této práci jsou v rámci úspory času použita data pouze pro Českou republiku. První krok předzpracování dat pro celou Evropu na autorově pracovním stroji trval přes dva dny čistého času a vyžadoval přes 70GB paměti RAM. To bylo vyřešeno pomocí tzv. **Swap memory**.[12]. V rámci úspory času bylo rozhodnuto použít jen omezenou množinu dat, která pro účely testování prototypu aplikace je omezená množina dat dostačující. Nicméně při produkčním nasazení budou pouze změnou parametru příkazů použita data pro celou Evropu.

4.9 Databázové schéma

Na obrázku 4.2 jsou zobrazené hlavní entity aplikace v databázovém schématu. Vynechány jsou entity vygenerované frameworkem Django.

[12]Swap memory - odkládací prostor, jehož primárním úkolem je nahradit místo na disku pamětí RAM, ve chvíli, kdy se skutečná RAM zaplní, ale stále je potřeba více místa [32]



Obrázek 4.2: Základ databázového schématu

Kapitola 5

Nasazení

5.1 Architektura

Výsledná aplikace se skládá ze tří komponent. Hlavní komponentou je samotná webová aplikace, která je napsaná ve frameworku Django. Další komponenty jsou Směrovací stroj OSRM a databáze PostgresSQL. Aplikace dále využívá dalších externích komponent. První takovou komponentou je Google, která slouží k odesílání e–mailů. Další komponenty jsou KontoBariery a PayPal, které se starají o funkcionality ohledně příspěvků. Poslední komponentou je API DPS Dozor, pomocí kterého aplikace získává informace o poloze posádek.

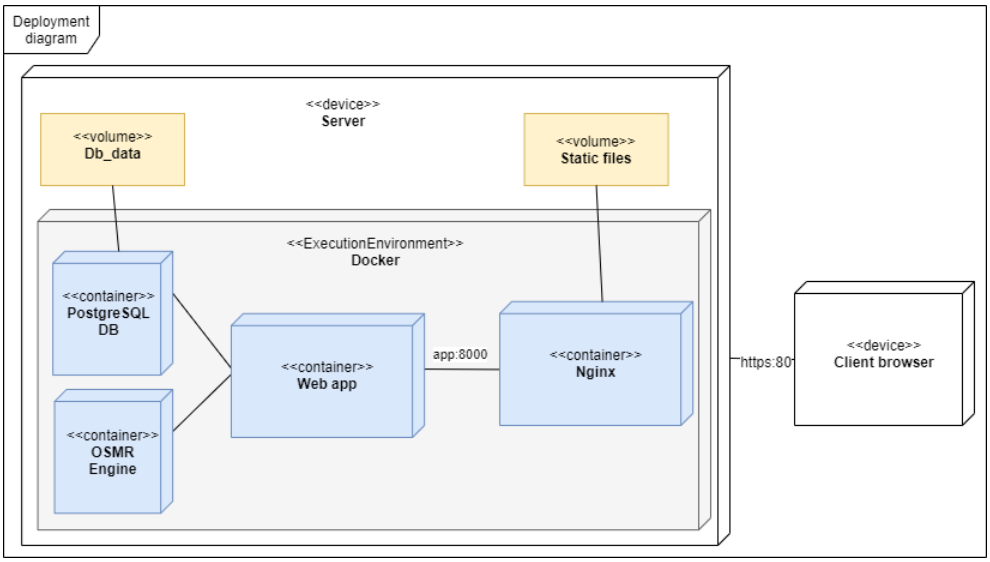
Komponenty, které nejsou externí, jsou nasazeny na serveru v samostatných Docker kontejnerech. Na hostovacím operačním systému jsou tedy nainstalovány pouze služby Docker a Docker Compose. Navíc se zde nachází kontejner obsahující Nginx server, který slouží pro poskytování statických souborů, ostatní požadavky přeposílá do kontejneru s webovou aplikací.

5.2 Docker

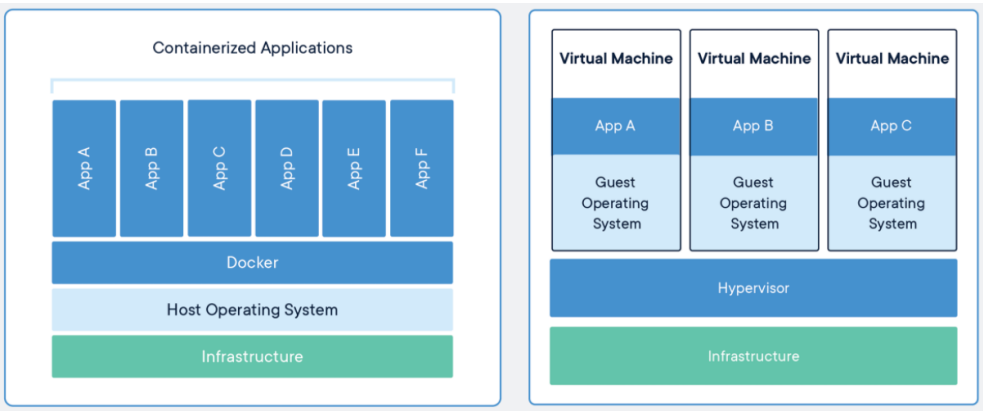
Docker je projekt s otevřeným kódem, který zajišťuje virtualizaci nad operačním systémem, takzvanou kontejnerizaci. U klasické virtualizace obsahuje každý virtuální stroj vlastní operační systém. Ten pak komunikuje s hostitelským operačním systémem přes hypervizora, kdežto Docker kontejnery přímo používají funkce hostitelského operačního systému, viz obrázek 5.2.

Výhody použití Dockeru jsou[33]:

- Kontejnery jsou izolované prostředí, a tím absolutně nezávislé na hostitelském operačním systému.
- Hostitelský operační systém neobsahuje žádné závislosti aplikace nasazené v kontejneru.



Obrázek 5.1: Diagram nasazení



Obrázek 5.2: Porovnání klasické virtualizace a kontejnerizace [2]

- Důsledkem toho, že kontejnery nevidí navzájem běžící procesy, je zvýšená bezpečnost. Pokud je aplikace v jednom kontejneru napadena, neovlivní to aplikace v jiných kontejnerech.

5.2.1 Konfigurace

Pro vytvoření Docker image je potřeba vytvořit soubor Dockerfile, ve kterém je popsán postup sestavení image. V této práci existuje Dockerfile pouze pro webovou aplikaci a pro Nginx server. Jelikož databáze PostgreSQL a směrovací stroj OSRM přichází s již dostupným Docker imagem, není potřeba psát vlastní Dockerfile i pro ně.

5.2.1.1 Dockerfile

Docker umožňuje automatizované sestavování imagů pomocí souboru Dockerfile. To je textový dokument, který obsahuje příkazy, které by uživatel musel zadat do příkazové řádky, aby image vytvořil [34]. Níže jsou popsány kroky, jak je sestaven Docker image výsledné aplikace. Celý Dockerfile je možné najít v příloze.

1. Specifikace základního image pro kontejner. V práci je využit image **python:3.8-apline**.
2. Nastavení cesty ke skriptům do proměnné prostředí **PATH** kontejneru.
3. Nakopírování souboru **requirements.txt** do kontejneru. To je textový soubor, ve kterém jsou vypsány veškeré závislosti, které aplikace má. Je vytvářen pomocí příkazu **pip freeze**.
4. Instalace knihoven, které mají být nainstalovány, dostupné po celou dobu běhu kontejneru. Na těchto knihovnách jsou závislé knihovny ze souboru **requirements.txt**.
5. Instalace knihoven ze souboru **requirements.txt**. Některé tyto knihovny mají závislosti, které jsou potřeba pouze pro instalaci, nikoli pro běh knihovny. Proto jsou na začátku tohoto kroku tyto závislosti nainstalovány a na konci kroku jsou smazány z důvodu redukce velikosti výsledného image.
6. Nakopírování dat projektu.
7. Nastavení spouštěcích práv nahraným skriptům.
8. Vystavení portu, přes který image bude komunikovat.
9. Spuštění skriptu, který spustí webový server.

5.2.1.2 Docker Compose

Docker Compose je nástroj, pomocí kterého se dají spouštět aplikace, obsahující více než jeden kontejner. Pro konfiguraci se používá soubor typu **YAML**, ve kterém jsou popsané jednotlivé služby. Poté stačí spustit příkaz **docker-compose up**, který spustí veškeré služby, které jsou v souboru **docker-compose.yml** nadefinované. Aby nebylo nutné pokaždé spouštět všechny nadefinované služby, Docker Compose umožňuje specifikovat jméno služby, která se má znovu spustit. V případě této práce jsou služby OSRM a PostgreSQL neměnné (Docker image jsou stahovány, nikoli vytvářeny vlastní), tudíž opakované spouštění by znamenalo opakované stahování a spouštění toho samého. Proto je dobré specifikovat jméno nasazované webové aplikace při opakovaném spouštění, a to **docker-compose up mcra** [35].

Pro přehlednost bude popsána pouze služba webové aplikace s názvem **mcra** (viz 20). Celý soubor **docker-compose.yml** je možné najít v příloze. Níže jsou popsány prvky, které byly použity pro nastavení této aplikace.

build Možnosti konfigurace, které jsou aplikovány při sestavování image. V tomto případě je určena pouze složka, ve které se nachází soubor **Dockerfile**. V konfiguraci služeb **osrm** a **db** je **build** nahrazen konfigurací **image**, která specifikuje, jaký Docker image se má použít, popřípadě stáhnout.

env_file Jeden nebo více souborů obsahující proměnné, které se mají v kontejneru nastavit jako proměnné prostředí. Očekávaný formát je **VAR=VAL**. Každý takovýto pár je na řádce samostatně.

ports Tato konfigurace nastavuje, na kterém portu bude služba dostupná. Obsahuje dvojici **HOST_PORT:CONTAINER_PORT**. Z hostitelského prostředí je služba dostuplná na portu **HOST_PORT**. Docker Compose vytvoří výchozí vlastní síť při každém spuštění. Každá služba se k této síti připojí a může komunikovat s ostatními službami připojenými k této síti pomocí portu **CONTAINER_PORT**.

depends_on Nastavení závislostí mezi službami. Tyto závislosti zajišťují mimo jiné pořadí, ve kterém se budou služby sestavovat, zastavovat nebo mazat.

command Přepisuje příkaz CMD v Dockerfilu. V této práci je přepsání použito proto, aby mohl být použit jiný **command** pro vývoj a pro nasazení se stejným souborem **Dockerfile**. V případě nasazení je použit skript **entrypoint.sh**, spustí čtyři příkazy:

1. **python manage.py collectstatic** –**no-input** pro shromáždění statických souborů ze všech aplikací do určené složky.

2. **python manage.py migrate –no-input** pro aplikování migrací do databáze.

3. **python manage.py createsuperusercustom –username admin –password admin –noinput –email admin@admin.cz** pro vytvoření prvotního uživatele v databázi, pokud ještě neexistuje.

4. **gunicorn mcra.wsgi:application –bind 0.0.0.0:8000** pro spuštění webového serveru. **gunicorn**, který přijímá požadavky na portu 8000.

volumes Specifikují připojení uložiště v kontejneru a hostitelského souborového systému. V případě služby **mcra** je použito **Named Volume**, u kterých není potřeba specifikovat, na kterou složku hostitelského systému se uložiště v kontejneru připojí.

```
  mcra:
    build:
      context: .
    env_file:
      - .env
    volumes:
      - media:/mcra/media
      - static:/mcra/static
    ports:
      - "8000:8000"
    depends_on:
      - "db"
      - "osrm"
    command: /scripts/entrypoint.sh

volumes:
  database_data:
  static:
  media:
```

Listing 5.1: nastavení služby mcra pro Docker Compose

5.3 Amazon web services

Amazon Web Services poskytuje širokou škálu globálních cloudových produktů, např. výpočetní, úložné, databázové, analytické, síťové, mobilní nebo vývojářské nástroje. Celkem je služeb poskytovaných AWS přes 175. Služby jsou dostupné na vyžádání během několika vteřin a jsou účtovány podle strategie **pay-as-you-go**[13]. AWS také přichází s volně dostupnými prostředky,

[13] pay-as-you-go je platební metoda pro cloudové služby, kde se účtují pouze využité služby [36]

které kdokoli může volně využívat. Tyto prostředky jsou výkonnostně a velikostně značně omezeny, ale pro účely této práce dostačující [37].

V následujících podsekcích jsou popsány služby, které byly při nasazení využity.

5.3.1 Elastic Compute Cloud

Pro nasazení aplikace byla využita služba EC2 (Elastic Compute Cloud), což je webová služba, která poskytuje výpočetní kapacitu v cloudu. Ta vývojářům umožňuje úplnou kontrolu nad poskytnutými výpočetními zdroji a skrze její webové rozhraní jde snadno konfigurovat [38]. Pro spuštění EC2 instance je potřeba provézt tyto kroky:

1. **Výběr Amazon Machine Image (AMI)** Ami je šablona obsahující operační systém, aplikační server a další aplikace, které jsou potřeba pro nastartování instance. Jelikož nasazovaná aplikace bude spuštěna jako Docker image, není výběr operačního systému podstatný. Byl tedy vybrán **Ubuntu server 20.04 LTS**, protože na stejném operačním systému byla aplikace vyvíjena.

2. **Výběr instance virtuálního serveru** Instance jsou různými kombinacemi parametrů: CPU, paměť, velikost uložiště a síťová kapacita. Výběr kombinací je rozsáhlý, avšak pouze jedna instance patří mezi volně dostupné prostředky. Jelikož v této práci není nasazován finální produkt, ale pouze prototyp pro testování, omezené prostředky jsou dostačující. Pokud bude rozhodnuto nasadit produkční verzi na AWS, změna instance je snadno proveditelná změna. Specifikace vybrané instance je následující:
   - **vCPU** - 1×2.5GHz
   - **Paměť** - 1GB
   - **Uložiště** - pouze Elastic Block Store (EBS)
   - **Síťová kapacita** - Nízká až střední

3. **Výběr uložiště** Uložiště je v práci řešeno pomocí externího EBS. To lze přiřazovat i ostatním instancím, vznikne tedy uložiště nezávislé na serveru a tím bude přechod na jinou instanci snazší. Byla tedy vybrána výchozí varianta poskytovaného uložiště s maximální možnou velikostí 30GB.

4. **Specifikace bezpečnostní skupiny** Tyto skupiny definují pravidla pro přístup na instanci po síti. V této práci je nastaveno, že má k instanci na portu 80 přístup jakákoli IP adresa.

5.3.2 Elastic Block Storage

Elastic Block Storage svazek je úložné zařízení na úrovni bloku, který je možné připojit k instancím. Po připojení svazku k instanci funguje, jako kdyby byl připojen fyzický pevný disk. Je možné dynamicky měnit jejich velikost, jejich typ i kapacitu IOPS [39]. V práci je použit svazek typu **General Purpose SSD (gp2)**. Tento svazek poskytuje rovnováhu mezi cenou a výkonem a je společností Amazon doporučován pro pokrytí většiny potřeb.[14]. Použitý svazek má následující specifikace:

- **Velikost** - 30GB
- **IOPS** - 100/3000 (minimum 100IOPS, při potřebě možné rozšířit až na 3000IOPS)

5.4 Výsledek nasazení

Výsledná webová aplikace je nasazená v cloudu pomocí služby AWS a je dostupná na adrese **ec2-3-122-61-13.eu-central-1.compute.amazonaws.com**. Jelikož výsledkem této práce je funkční prototyp, nikoli produkční verze aplikace, není nasazena pod doménou závodu. Ta je v době psaní práce stále využívána starou webovou aplikací. Aplikace byla nasazena, aby ji bylo možné uživatelsky otestovat.

[14] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html?icmpid=docs_ec2_console

Kapitola 6

Testování

6.1 Testování programátorem

Testování programátorem probíhalo po celou dobu vývoje webové aplikace. Po každém logickém celku kódu přidaným do aplikace byla otestována jeho funkcionalita a jeho dopad na ostatní části aplikace. Odhalení chyby v této fázi vývoje může předejít velkým chybám v budoucnu a cena za opravu je malá [40].

Během vývoje aplikace bylo tímto způsobem odhaleno velké množství chyb, které byly následně ihned opravovány.

6.2 Nielsenova heuristická analýza

Před otestováním webové aplikace uživateli bylo uživatelské rozhraní otestováno pomocí heuristické analýzy podle Jacoba Nielsena. Heuristika se skládá z 10 hlavních bodů, které by mělo uživatelské rozhraní splňovat. Pokud tyto body splňuje, pak by mělo být uživatelsky přívětivé a snadno použitelné [41].

6.2.1 Viditelnost stavu systému

Uživatel při používání systému musí být obeznámen s tím, kde se právě nachází a co systém v danou chvíli dělá.

Stránky v aplikaci obsahují přehledné nadpisy, které uživateli poskytují informaci o tom, kde se právě nachází. V navigační liště je dále zvýrazněný odkaz na danou část aplikace. Po odeslání jakéhokoli formuláře je uživatel informován o výsledcích jeho požadavku nebo je mu poskytnuta možnost přesměrování na nově vytvořenou položku.

6.2.2 Shoda mezi systémem a realitou

Systém by měl používat jazyk uživatele a měl by zachovávat konvence reálného světa, např. ikona koše by měla symbolizovat smazání.

Aplikace je lokalizovaná do dvou jazyků a v obou případech je využívá v jednoduché formě. Ikony jsou použity pouze v navigační liště, a to za účelem doplnění textu, nikoli samostatně.

6.2.3 Uživatelská kontrola a svoboda

Uživatelé občas spustí akci v systému omylem. Systém jim tedy musí umožnit jednoduchou cestu zpět.

Z povahy webové aplikace je možnost vracet se na předchozí navštívenou stránku splněna. V aplikaci jsou všechny důležité akce, např. odesílání formuláře se splněným úkolem, doplněny dialogem, zda uživatel chce tuto akci opravdu provést.

6.2.4 Konzistence a standardizace

Systém by měl dodržovat platformní a průmyslové konvence, aby uživatel nemusel přemýšlet, zda různá slova, situace nebo akce znamenají totéž.

Napříč celou aplikací mají komponenty se stejnou, či podobnou funkcionalitou konzistentní tvar i barvu a rozložení webových stránek podobného typu je vždy stejné.

6.2.5 Prevence chyb

Systém by měl předcházet chybovým stavům, upozornit uživatele na to pokud zapomene vyplnit povinné políčko formuláře. To by mělo být jasně zvýrazněno, aby se v první řadě uživatel vůbec nepokusil odeslat formulář bez jeho vyplnění. Dále jsou užitečné dialogy, ve kterých se systém uživatele zeptá, zda danou akci opravdu chce provést, např. při mazání obsahu.

6.2.6 Rozpoznání místo vzpomínání

Uživatel by neměl být nucen pamatovat si informace z jedné části systému pro použití druhé části. Informace potřebné pro používání určité části systému by měly být viditelné, nebo snadno vyhledatelné.

Uživateli k ovládání systému stačí navigační tlačítka a nadpisy obrazovek. Akce, které momentálně uživatel nemůže provézt mu buďto nejsou vůbec zobrazeny, nebo jsou zobrazeny tak, aby bylo patrné, že danou akci není možné použít.

6.2.7 Flexibilní a efektivní použití

Pokročilí uživatelé systému by měli mít možnost využívat zkratek, které urychlují proces častých akcí.

6.2.8 Estetický a minimalistický design

Obsahem stránek by neměli obsahovat informace, které jsou irelevantní nebo zřídka potřebné. Takové informace snižují viditelnost těch relevantních.

V aplikaci je použito zobrazování různé úrovně detailu informací. Např. v seznamu posádek jsou zobrazena pouze jména, věk a povolání členů posádek. Na stránce detailu je navíc dostupný jejich popis a zážitky.

6.2.9 Pomoc uživatelům pochopit a vzpamatovat se z chyb

Chybové hlášky by měly být vyjádřeny v prostém jazyce (žádné chybové kódy), přesně definovat vzniklý problém a konstruktivně navrhnout jeho řešení.

Například pokud posádka zvolí akci **Navštívit místo**, ale není ještě dostatečně blízko bodu odevzdání, zobrazí se o tom chybová hláška.Ta obsahuje informaci o tom, že se nenachází v akceptovatelné vzdálenosti od místa úkolu a kolik km by ještě mela urazit.

6.2.10 Nápověda a dokumentace

Nejlepší je, pokud systém nevyžaduje žádné další vysvětlení. Může však být nutné poskytnout uživatelům dokumentaci, která jim pomůže dokončit své úkoly.

Nápověda a dokumentace systému není momentálně vytvořena.

6.3 Uživatelské testování

Nasazený prototyp byl nakonec uživatelsky otestován. Testování probíhalo s pěti osobami, neboť podle [42] je to ideální počet pro odhalení většiny chyb systému. Testování probíhalo na předem připravených datech, aby pro účastníky testu bylo možné otestovat i funkčnost validace souřadnic. Každý z nich používal svoje vlastní prostředí, aby se otestovala kompatibilita aplikace na různých přístrojích. Účastníci měli chvíli na seznámení s webovými stránkami a následně jim byl předložen seznam úkolů, které se měli pokusit bez pomoci splnit. Administrační část jim byla v menší míře předem vysvětlena, jelikož není veřejně dostupná a pro její používání je třeba rozumět fungování závodu a vztahům mezi daty, které se v administračním prostředí spravují.

1. Najděte počet kilometrů, které ujela posádka tvořená **Karímem a Pepou**.

2. Zjistěte, jaké posádka splnila.
3. Přečtěte si, jak se posádka dala dohromady a proč se k závodu přihlásili.
4. Podívejte se na zážitky, které přidala posádka tvořená **Honzou a Adamem**.
5. Zjistěte, jaké posádky splnily úkol **Navštívit náměstí v Klatovech**.
6. Prohlídněte si, jaké zážitky měly posádky při plnění úkolu **Sehnat nůžky na Klenové**.
7. Zjistěte, do kdy je možné se registrovat do kvalifikačního závodu.
8. Přečtěte si o tom, na co se přispívalo v roce 2019.
9. Na mapě si projděte trasu závodu.
10. Přispějte posádce tvořené **Honzou a Adamem**.
11. Registrujte se do kvalifikačního závodu.
12. Přihlaste se do administračního prostředí pod uživatelským jménem **admin** s heslem **admin**.
13. V administračním prostředí kvalifikujte vámi vytvořenou posádku do hlavního závodu a přiřaďte jí zařízení s kódem **ZIDEPERUHA**.
14. Odhlaste se a přihlaste se jako vámi vytvořená posádka.
15. Odevzdejte úkoly **Sežeň nůžky na Klenové** a **Navštívit náměstí v Klatovech**.
16. V administračním prostředí schvalte odevzdané úkoly.

6.3.1 Výsledky uživatelského testování

Pozorování účastníků při plnění seznamu úkolů poskytlo důležité informace o tom, které části aplikace jsou pro uživatele náročnější na pochopení nebo orientaci. S každým z účastníků testování byl posléze veden rozhovor o uživatelské přívětivosti a jejich názoru, co by na aplikaci změnili.

Následuje seznam zjištěných hlavních problémů, návrhů vzniklých na základě rozhovorů a jejich řešení:

- **Tlačítka znázorňující aktuální krok v rámci registrace - na první pohled není zřejmé, k čemu slouží.**
  Tlačítkům byl přidán popis kroku, na který odkazují.
- **Nadpis pro položku formuláře zmizí při jejím vyplňování**
  Nad všechny položky formulářů byl doplněn název položky. Doposud byl vidět pouze v nevyplněném políčku jako nápověda.

- **Při nahrávání obrázků do zážitků a úkolů není popsáno, co se od uživatele očekává.**
  Byl doplněn popisek, který vysvětluje, jaký soubor se má nahrát.
- **Po obnovení hesla uživatel zůstane na stránce, ze které není jasné jak se má dostat pryč.**
  Bylo přidáno tlačítko s odkazem na úvodní stránku a tlačítko s odkazem na přihlášení.
- **Uživatel není informován o úspěšném navštívení místa.**
  Byla přidána zpráva s danou informací.
- **Nejasně pojmenované tlačítko na přečtení zážitku.**
  Tlačítko bylo přejmenováno na **Přečíst více**, před tím **Přečíst článek** (Článek je v kontextu závodu něco jiného).
- **Buňky na mapě s posádkami nejsou na první pohled přiřazeny posádkám.**
  Místo výchozí buňky byla použita profilová fotka posádky. Pokud uživatel najede myší na buňku, ukáží se jména členů posádky.Pokud na ni klikne, zobrazí se detailní výsledky posádky.
- **Často kladené otázky pod pravidly závodu jsou matoucí.**
  Tato sekce byla přesunuta na stránku s obecnými informacemi o závodu.

Kapitola 7

Nápady pro další rozvoj

Výsledkem této práce je funkční prototyp, který obsahuje všechny náležitosti potřebné k pořádání závodu. Jelikož je aplikace napsána ve frameworku Django, je jednoduché díky rozdělení na aplikace přidávat nové funkcionality. V této kapitole je popsáno několik rozšíření aplikace, která by ji mohla vylepšit.

7.1 Hledání spolujezdce

Pro účast v závodu je zapotřebí dvou účastníků v posádce. Ne každý má však ve svém okolí někoho, kdo by s ním tento závod podstoupil. Proto by bylo dobré pro takové lidi vytvořit místo, kde by o sobě mohli napsat nějaké detailnější informace a na základě těch si najít spolujezdce do závodu.

7.2 Komunikace mezi posádkou a organizátory

Momentálně posádky komunikují s organizátory přes sociální sítě nebo přes SMS. Dobrým rozšířením by byl komunikační kanál ve formě okna pro chat, kde by komunikace probíhala. Organizátoři by tedy měli veškeré komunikační kanály na jednom místě a organizace závodu by pro ně byla pohodlnější.

7.3 Komunikace mezi posádkou a návštěvníky stránek

Bylo by dobré návštěvníkům stránek poskytnout možnost vyjádřit podporu posádkám i jiným způsobem, než anonymním příspěvkem. Toho by se dalo docílit pomocí přímých zpráv posádkám, nebo formou komentářů, např. pod zážitky posádky nebo přímo pod jejich profil.

7.4 Zaznamenání trasy jednotlivých posádek

Každá posádka pro svojí cestu voli jinou trasu. Proto by mohlo být zajímavé, ukládat trasu posádky, která by byla zobrazena na mapě v detailu posádky. Návštěvníci webových stránek by tak měli lepší představu, kudy posádka jela a která místa po své cestě navštívila.

Závěr

Cílem této práce bylo vytvořit a otestovat prototyp webové aplikace pro neziskovou organizaci Světem stopem, která by v budoucnu nahradila jejich stávající, pro organizátory nevyhovující webové stránky. Jednalo se o aplikaci spravující závod Mistrovství ČR v autostopu.

Nejprve byla provedena analýza stávajícího řešení. Při této analýze byla odhalena spousta nedostatků, které bylo potřeba eliminovat v nově navrhované aplikaci. Na základě této analýzy a rozhovorech s organizátory závodu byly definovány funkční a nefunkční požadavky, které výsledná aplikace musí splňovat. Dále byly v kapitole Analýza vybrány technologie, pomocí kterých se aplikace implementovala. Návrhu aplikace a jejího uživatelského rozhraní se věnuje kapitola 3. Důležité a zajímavé části implementace, která nakonec probíhala ve frameworku Django programovacího jazyka Python, jsou popsány v kapitole 4. Po dokončení implementace bylo potřeba prototyp uživatelsky otestovat. Proto byla aplikace nasazena do cloudu pomocí služby AWS a je dostupná na adrese **http://ec2-3-122-61-13.eu-central-1.compute.amazonaws.com**. Podrobnostem nasazení je věnována kapitola 5 a jejímu testování kapitola 6. Na konci práce bylo diskutováno, jaká rozšíření by se v budoucnu mohla do aplikace dodat pro její zkvalitnění.

Všechny cíle, které vycházely jak ze zadání této práce, tak z funkčních a nefunkčních požadavků na výslednou aplikaci, byly splněny. Aplikace poskytuje uživatelům přehledné webové stránky s informacemi o závodu Mistrovství ČR v autostopu a administrátorům závodu kvalitní administrační prostředí, ve kterém mohou závod spravovat. Aplikace nebyla nasazena do produkčního prostředí, protože organizátoři chtějí aplikaci nejprve vyzkoušet na cvičném závodě, který nebylo možné kvůli vládním opatřením v době psaní práce zorganizovat.

Literatura

[1] Valuy, S.: A Comparison of Single-Page and Multi-Page Applications. červen 2020, [Online]. Dostupné z: https://dzone.com/articles/the-comparison-of-single-page-and-multi-page-appli

[2] Docker, Inc.: *What is a Container? [online].* [cit. 2021-4-7]. Dostupné z: https://www.docker.com/resources/what-container

[3] Nadace Charty 77: *Konto Bariéry [online].* [cit. 2021-4-6]. Dostupné z: https://www.kontobariery.cz/O-nas/Konto-Bariery

[4] Techopedia Inc.: *Content Management System (CMS) [online].* [cit. 2021-4-7]. Dostupné z: https://www.techopedia.com/definition/24075/content-management-system-cms

[5] AltexSoft: *Functional and Nonfunctional Requirements: Specification and Types [online].* [cit. 2021-4-9]. Dostupné z: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

[6] Stack Exchange Inc: *Most Popular Technologies [online].* [cit. 2021-4-10]. Dostupné z: https://insights.stackoverflow.com/survey/2020#most-popular-technologies

[7] Python Software Foundation: *What is Python? Executive Summary [online].* [cit. 2021-4-10]. Dostupné z: https://www.python.org/doc/essays/blurb/

[8] JetBrains s.r.o. and Python Software Foundation: *Python Developers Survey 2020 Results [online].* [cit. 2021-4-10]. Dostupné z: https://www.jetbrains.com/lp/python-developers-survey-2019/

[9] Makai, M.: WSGI Servers. *Full Stack Python [online]*, květen 2014, [cit. 2021-4-12]. Dostupné z: https://www.fullstackpython.com/wsgi-servers.html

[10] Tutorials Point India Limited: *Flask – Overview [online]*. [cit. 2021-4-7]. Dostupné z: https://www.tutorialspoint.com/flask/flask_overview.htm

[11] Tutorials Point India Limited: *Django - Basics [online]*. [cit. 2021-4-7]. Dostupné z: https://www.tutorialspoint.com/django/django_basics.htm

[12] Django Software Foundation and individual contributors: *Databases [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.djangoproject.com/en/3.2/ref/databases/

[13] The PostgreSQL Global Development Group: *About [online]*. [cit. 2021-4-7]. Dostupné z: https://www.postgresql.org/about/

[14] Google: *Google Maps Platform [online]*. [cit. 2021-4-7]. Dostupné z: https://cloud.google.com/maps-platform

[15] Django Software Foundation and individual contributors: *Deploying Django [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.djangoproject.com/en/3.2/howto/deployment/

[16] Skólski, P.: Single-page application vs. multiple-page application. *Neoteric [online]*, prosinec 2016, [cit. 2021-4-12]. Dostupné z: https://neoteric.eu/blog/single-page-application-vs-multiple-page-application/?utm_source=medium.com&utm_medium=social&utm_content=neo&utm_campaign=blog

[17] ITnetwork: *UML - Doménový model [online]*. [cit. 2021-4-7]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-domenovy-model-diagram

[18] Django Software Foundation and individual contributors: *Model [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.djangoproject.com/en/3.2/topics/db/models/

[19] Real Python: *Django Migrations: A Primer [online]*. [cit. 2021-4-7]. Dostupné z: https://realpython.com/django-migrations-a-primer/

[20] Django Software Foundation and individual contributors: *Making queries [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.djangoproject.com/en/3.2/topics/db/queries/

[21] Johnson, A.: Django and the N+1 Queries Problem. *scoutapm [online]*, září 2020, [cit. 2021-4-12]. Dostupné z: https://scoutapm.com/blog/django-and-the-n1-queries-problem

[22] Thagana, K.: Introduction to Django Signals. *pluralsight [online]*, říjen 2020, [cit. 2021-4-12]. Dostupné z: https://www.pluralsight.com/guides/introduction-to-django-signals

[23] Django Software Foundation and individual contributors: *The Django template language [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.djangoproject.com/en/3.2/ref/templates/language/

[24] Grudl, D.: Escapování – definitivní příručka. *phpfashion [online]*, květen 2009, [cit. 2021-4-12]. Dostupné z: https://phpfashion.com/escapovani-definitivni-prirucka#comment-18832

[25] Amazon Web Services, Inc. or its affiliates: *Amazon Simple Email Servicee [online]*. [cit. 2021-4-7]. Dostupné z: https://aws.amazon.com/ses/

[26] Amazon Web Services, Inc. or its affiliates: *Amazon SES pricing [online]*. [cit. 2021-4-7]. Dostupné z: https://aws.amazon.com/ses/pricing/

[27] Python Software Foundation: *django-paypal 1.1.1 [online]*. [cit. 2021-4-7]. Dostupné z: https://pypi.org/project/django-paypal/

[28] PayPal: *Instant Payment Notification [online]*. [cit. 2021-4-7]. Dostupné z: https://developer.paypal.com/docs/api-basics/notifications/ipn/

[29] PayPal: *PayPal sandbox testing guide [online]*. [cit. 2021-4-7]. Dostupné z: https://developer.paypal.com/docs/api-basics/sandbox/

[30] ggspatial: *Distance on a sphere: The Haversine Formula [online]*. [cit. 2021-4-7]. Dostupné z: http://www.ggspatial.co.uk/distance-on-a-sphere-the-haversine-formula/

[31] PROJECT OSRM: *Open Source Routing Machine [online]*. [cit. 2021-4-7]. Dostupné z: https://github.com/Project-OSRM/osrm-backend

[32] Both, D.: An introduction to swap space on Linux systems. *opensource [online]*, březen 2020, [cit. 2021-4-12]. Dostupné z: https://opensource.com/article/18/9/swap-space-linux-systems

[33] Covassi, G.: The 6 Benefits of Docker Container. *kiratech [online]*, prosinec 2018, [cit. 2021-4-12]. Dostupné z: https://www.kiratech.it/en/blog/the-6-benefits-of-docker-container

[34] Docker, Inc.: *Dockerfile reference [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.docker.com/engine/reference/builder/

[35] Docker, Inc.: *Docker compose [online]*. [cit. 2021-4-7]. Dostupné z: https://docs.docker.com/compose/

[36] Sullivan, E.: pay-as-you-go cloud computing (PAYG cloud computing). *TechTarget [online]*, březen 2015, [cit. 2021-4-12]. Dostupné z: https://searchstorage.techtarget.com/definition/pay-as-you-go-cloud-computing-PAYG-cloud-computing

[37] Amazon Web Services, Inc. or its affiliates: *About AWS [online]*. [cit. 2021-4-7]. Dostupné z: https://aws.amazon.com/about-aws/

[38] Amazon Web Services, Inc. or its affiliates: *Amazon EC2 [online]*. [cit. 2021-4-7]. Dostupné z: https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc

[39] Amazon Web Services, Inc. or its affiliates: *Amazon Elastic Block Store [online]*. [cit. 2021-4-7]. Dostupné z: https://aws.amazon.com/ebs/?ebs-whats-new.sort-by=item.additionalFields.postDateTime&ebs-whats-new.sort-order=desc

[40] Hlava, T.: Fáze a úrovně provádění testů. *testovanisoftwaru [online]*, září 2011, [cit. 2021-4-12]. Dostupné z: http://testovanisoftwaru.cz/tag/testovani-programatorem/

[41] Nielsen, J.: 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group [online]*, listopad 2020, [cit. 2021-4-12]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[42] Nielsen, J.: Why You Only Need to Test with 5 Users. *Nielsen Norman Group [online]*, březen 2000, [cit. 2021-4-12]. Dostupné z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Příloha A

Seznam použitých zkratek

GPS Global Positioning System

API Application Programming Interface

DMS Dárcovská SMS

CSS Cascading Style Sheets

CMS Content Management System

URL Uniform Resource Locator

HTTP Hypertext Transfer Protocol

ORM Object Relational Mapping

DRY Don't Repeat Yourself

LESS Leaner Style Sheets

SASS Syntactically Awesome Style Sheets

ACID Atomicity Consistency Isolation Durability

XML eXtensible Markup Language

OSRM Open Source Routing Machine

AWS Amazon Web Services

EC2 Elastic Compute Cloud

EBS Elastic Block Storage

SES Simple Email Service

MPA Multi Page Application

SPA Single Page Application

AJAX Asynchronous JavaScript and XML

SEO Search Engine Optimalization

MVT Model View Template

MVC Model View Controller

DTL Django Template Language

JSON JavaScript Object Notation

IPN Instant Payment Notification

IOPS Input / Output Per Second

Příloha B

Příručka

B.1 Přihlašovací údaje

Jak je zmíněno v kapitole 4.5, v době psaní práce má aplikace status **SANDBOX** a tak je potřeba využívat pouze registrované emaily, které slouží výhradně pro testování této aplikace. Po úspěšném schválení aplikace službou Amazon SES budou tyto e–maily zrušeny.

Všechny tyto emaily mají heslo: **Autostopem1**

Pro aplikaci jako takovou, tz. ze které se odesílají emaily je založená e–mailová schránka: **mcra.dev@gmail.com**

Pro uživatele posádky je možné využít těchto e–mailových schránek:

- **mcra.test.user@gmail.com**
- **mcra.test.user2@gmail.com**
- **mcra.test.user3@gmail.com**
- **mcra.test.user4@gmail.com**

B.2 Nasazení

Aby mohla být aplikace nasazena s použitím zdrojových kódů aplikace, musí prostředí, kde se bude nasazovat, splňovat tyto podmínky:

- Paměť RAM min. 3GB
- Volné místo na disku cca. 2GB (je potřeba rezerva pro dynamicky nahrávaný obsah)
- Volné porty: 8000, 80, 6000 a 5000
- Mít k dispozici služby Docker a Docker-Compose

Nasazení potom probíhá podle těchto kroků:

1. Nakopírovat složku **mcra** na lokální uložiště
2. Přepnout se do složky **mcra/OSRM**
3. Stáhnout a zpracovat data pro OSRM směrovací stroj a to pomocí příkazů (v tomto případě jsou to data pro ČR):
   a) wget http://download.geofabrik.de/europe/czech-republic-latest.osm.pbf
   b) docker run -t -v "$PWD:/data"osrm/osrm-backend osrm-extract -p /opt/car.lua /data/czech-republic-latest.osm.pbf
   c) docker run -t -v "$PWD:/data"osrm/osrm-backend osrm-partition /data/czech-republic-latest.osrm
   d) docker run -t -v "$PWD:/data"osrm/osrm-backend osrm-customize /data/czech-republic-latest.osrm
4. Přepnout do složky **mcra**
5. Spustit příkaz **docker-compose up –build**, pokud je potřeba, aby server běžel v pozadí, stačí přidat přepínač **-d**
6. Aplikace je dostupná pod adresou **localhost**.

Když už server běží je potřeba naplnit základní data aby stránky ukazovaly to, k čemu jsou určeny. Proto je potřeba vytvořit některé objekty na stránce **localhost/admin**. Přihlašovací jméno stejně jako heslo je **admin**.

Je třeba vytvořit a spojit objekty v tomto pořadí:

- Založit hlavní závod **(Race)**
- Založit kvalifikační závod **(QualificationRace)** a spojit s ho s hlavním závodem.
- Vytvořit kvalifikační posádku **(QualificationCrew)**. To vyžaduje založení dvou účastníků, jejichž e–mailové adresy musí být ty z výše uvedených.
- Na nově vytvořenou posádku aplikovat akci **Qualify crew to the main event**. Tím se odešle na zadané e–mailové adresy e–mail s přihlašovacími údaji.
- Založit **Tracker** s názvem **ZIDEPERUHA** a přiřadit ho nově vytvořené posádce **Crew**. Toto zařízení posílá své souřadnice a jako jediná takto vytvořená posádka se bude zobrazovat na mapě.
- Založit úkol **(Task)** a spojit ho s hlavním závodem

- Vytvořit kontrolní bod **(Point)** a spojit ho s úkolem, z předchozího kroku.
- Vytvořit článek **(Article)** se zaškrtnutým políčkem **is_active** a spojit ho s hlavním závodem
- Spustit akci **Start race**
  Po splnění všech kroků by měla být aplikace nasazená a závod spuštěn.

Příloha C

Ukázky kódu

C.1 Dockerfile

```
FROM python:3.8-alpine
# Add scripts to path of the container
ENV PATH="/scripts:${PATH}"

# Install all requirements, that MCRA app has. (including
    node_modules)
COPY ./requirements.txt /requirements.txt
RUN apk update \
    && apk upgrade \
    && apk add --no-cache \
        freetype \
        libpng \
        libjpeg-turbo \
        freetype-dev \
        libpng-dev \
        jpeg-dev \
        libjpeg \
        libjpeg-turbo-dev
RUN apk add postgresql-dev \
    && apk add --update --no-cache --virtual .tmp gcc python3-dev
    musl-dev jpeg-dev libjpeg\
    && pip install -r requirements.txt \
    && apk del .tmp

# Copy actual data
RUN mkdir /mcra
COPY ./mcra /mcra
COPY ./scripts /scripts
WORKDIR /mcra

# Add executable rights to all scripts
RUN chmod +x /scripts/*

EXPOSE 8000
```

```
33
34 # Execution of starting script
35 CMD ["/scripts/entrypoint.sh"]
```

C.2 docker-compose.yml

```
1  version: '3.8'
2
3  services:
4    db:
5      image: postgres
6      env_file:
7        - database.env
8      ports:
9        - "6000:5432"
10     volumes:
11       - database_data:/var/lib/postgresql/data/
12
13   osrm:
14     image: osrm/osrm-backend
15     container_name: osrm_stage
16     ports:
17       - "5000:5000"
18     volumes:
19       - ./OSRM:/OSRM/data
20     command: "osrm-routed --algorithm mld /OSRM/data/czech-republic-
       latest.osrm"
21
22   nginx:
23     build: ./nginx
24     volumes:
25       - static:/mcra/static
26     ports:
27       - "80:80"
28     depends_on:
29       - mcra
30
31   mcra:
32     build:
33       context: .
34     env_file:
35       - .env
36     volumes:
37       - media:/mcra/media
38       - static:/mcra/static
39     ports:
40       - "8000:8000"
41     depends_on:
42       - "db"
43       - "osrm"
44     command: /scripts/entrypoint.sh
45
46 volumes:
```

```
  database_data:
  static:
  media:
```

C.3 base.html

```
{% load static %}

{% load i18n %}

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <!-- Font awesome -->
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.11.2/css/all.css">
        <!-- Google Fonts Roboto -->
        <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&display=swap">
        <!-- CSS -->
        <link rel="stylesheet" href="{% static 'css/main.css' %}" />
    {% block stylesheets %}
    {% endblock %}

    <title>{% trans "CR championship in hitchhiking" %}</title>
</head>

<body>
    <!-- Navbar -->
    {% url 'index' as introduction_url %}
    {% url 'experiences' as experiences_url %}
    {% url 'crews:crews' as crews_url %}
    {% url 'races:detail' as race_url %}
    {% url 'articles:index' as article_url %}

    <div class="w-100">
        <div class="container-md header-container py-2">
            <a class="navbar-brand my-auto" href="{% url 'index' %}">
                <img src="{% static 'img/logo.png' %}" alt="MCR logo" height="50px" />
            </a>
            <div class="my-auto header-links">
                <form action="{% url 'set_language' %}" method="POST">{% csrf_token %}
                    {% if redirect_to %}
                        <input hidden name="text" value="{{ redirect_to }} " />
```

```
41                      {% endif %}
42                      <select name="language" id"" class="form-select"
     onchange="this.form.submit()">
43                          {% get_current_language as LANGUAGE_CODE %}
44                          {% get_available_languages as LANGUAGES %}
45                          {% get_language_info_list for LANGUAGES as
    languages %}
46                          {% for language in languages %}
47                          <option data-thu value="{{ language.code }}"
     {% if language.code == LANGUAGE_CODE %} selected
48                              {% endif %}>
49                              <strong>{{ language.code }}</strong>
50                          </option>
51                          {% endfor %}
52                      </select>
53                  </form>
54                  {% if user.is_authenticated %}
55                  <a class="btn btn-white py-2" aria-current="page"
    href="{% url 'crews:edit_profile' %}">{{ user }}</a>
56                  <a class="btn btn-primary py-2 btn-big" aria-current
    ="page" href="{% url 'logout' %}">{% trans "Sign out" %}</a>
57                  {% else %}
58                  <a class="btn btn-white py-2" aria-current="page"
    href="{% url 'login' %}">{% trans "Sign in" %}</a>
59                  <a class="btn btn-primary py-2 btn-big" aria-current
    ="page" href="{% url 'crews:register' %}">{% trans "Sign up" %}<
    /a>
60                  {% endif %}
61              </div>
62              <button id="toggleNavbar" class="btn btn-primary" type="
    button" data-bs-toggle="collapse"
63                  data-bs-target="#navbarNav" aria-controls="navbarNav
    " aria-expanded="false"
64                  aria-label="Toggle navigation">
65                  <i class="fas fa-bars"></i>
66              </button>
67          </div>
68          <nav class="navbar navbar-expand-md navbar-dark bg-primary
    shadow-sm p-0">
69              <!-- Container wrapper -->
70              <div class="container-md">
71                  <!-- Collapsible wrapper -->
72                  <div class="collapse navbar-collapse" id="navbarNav"
    >
73                      <!-- Left nav items -->
74
75
76                      <ul class="navbar-nav navbar-nav-left me-auto ms
    -5-lg my-auto">
77                          <li class="nav-item">
78                              <a class="nav-link {% if request.path ==
     introduction_url %} active {%endif%}" aria-current="page" href=
    "{{ introduction_url }}">
79                                  <i class="fas fa-home"></i>
```

```
80                                  {% trans "Introduction" %}</a>
81                          </li>
82                          <li class="nav-item">
83                              <a class="nav-link {% if  'crews' in
    request.path %}active {%endif%}" aria-current="page" href="{{
    crews_url }}">
84                                  <i class="fas fa-car "></i> {% trans
     "Crews" %}</a>
85                          </li>
86                          <li class="nav-item">
87                              <a class="nav-link {% if  'tasks' in
    request.path %}active {%endif%}" aria-current="page" href="{%
    url 'races:task-list' %}">
88                                  <i class="fas fa-tasks"></i>
89                                  {% trans "Tasks" %}</a>
90                          </li>
91                          <li class="nav-item">
92                              <a class="nav-link {% if  'races' in
    request.path and 'tasks' not in request.path %}active {%endif%}"
     aria-current="page" href="{{ race_url }}">
93                                  <i class="fas fa-flag-checkered"></i
    >
94                                  {% trans "About the race" %}</a>
95                          </li>
96                          <li class="nav-item ">
97                              <a class="nav-link {% if 'experiences'
    in request.path%} active {%endif%}" aria-current="page" href="{{
     experiences_url }}">
98                                  <i class="fas fa-grin-tears"></i>
99                                  {% trans "Experiences" %}</a>
100                         </li>
101                         <li class="nav-item">
102                             <a class="nav-link {% if 'articles' in
    request.path%} active {%endif%}" aria-current="page" href="{{
    article_url }}">
103                                 <i class="fas fa-hands-helping"></i>
104                                 {% trans "We help!" %}</a>
105                         </li>
106                 </ul>
107                 <a class="btn btn-secondary btn-big" aria-
    current="page" href="{% url 'payments:crew_select' %}">{% trans
    "I want to contribute" %}</a>
108             </div>
109             <!-- Collapsible wrapper -->
110         </div>
111         <!-- Container wrapper -->
112     </nav>
113
114     {% block head_content %}
115     {% endblock %}
116
117     {% if messages %}
118     {% for message in messages %}
119     {%if message.level == 50 %}
```

```
120         <div name="message" class="container">
121             <div class="alert alert-warning alert-dismissible fade
    show mt-3" role="alert">
122                 <div class="text-center">
123                     <strong>
124                         {{ message.tags|title }}
125                     </strong>
126                     {{ message }}
127                 </div>
128                 <button type="button" class="btn-close" data-bs-
    dismiss="alert" aria-label="close"></button>
129             </div>
130         </div>
131         {% endif %}
132         {% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}
133         <div name="message" class="container">
134             <div class="alert alert-danger alert-dismissible fade
    show mt-3" role="alert">
135                 <div class="text-center">
136                     {{ message }}
137                 </div>
138                 <button type="button" class="btn-close" data-bs-
    dismiss="alert" aria-label="close"></button>
139             </div>
140         </div>
141         {% endif %}
142         {% if message.level == DEFAULT_MESSAGE_LEVELS.SUCCESS %}
143         <div name="message" class="container">
144             <div class="alert alert-success alert-dismissible fade
    show mt-3" role="alert">
145                 <div class="text-center">
146                     {{ message }}
147                 </div>
148                 <button type="button" class="btn-close" data-bs-
    dismiss="alert" aria-label="close"></button>
149             </div>
150         </div>
151         {% endif %}
152         {% endfor %}
153         {% endif %}
154
155
156
157         <!-- Navbar End -->
158     </div>
159     {% block content %}
160     {% endblock %}
161     <!-- Footer -->
162     <footer class="mt-auto footer-dark">
163
164         <div class="copyright">
165             <div class="container">
166                 <div class="copyright row">
167                     <div class="col-auto">
```

```
168                         <div class="align-middle">
169                             © 2020 Copyright - MCRA
170                         </div>
171                     </div>
172                     <div class="col-auto link-list">
173                             <div class="headline">{% trans "
    Transparent account" %}</div>
174                             <p>201323123/0100</p>
175                     </div>
176                     <div class="col-auto  link-list">
177                         <div class="headline">{% trans "Contacts" %}
    </div>
178                         <p>klarakuc@seznam.cz</p>
179                     </div>
180                     <div class="col-auto">
181                         <a href="https://www.facebook.com/
    mistrovstvivautostopu" target="_blank" rel="noopener"><i id="
    social-fb" class="fab fa-facebook fa-2x"></i></a>
182                         <a href="https://www.instagram.com/
    mistrovstvi_cr_v_autostopu/" target="_blank" rel="noopener"><i
    id="social-ig" class="fab fa-instagram fa-2x ms-2"></i></a>
183                     </div>
184                 </div>
185             </div>
186         </div>
187     </footer>
188 </body>
189
190 </html>
191     <script src="{% static 'js/main.js' %}"></script>
192     <script src="{% static 'js/bootstrap/bootstrap.js' %}"></script>
193     <script type="text/javascript" src="{% url 'javascript-catalog'
    %}"></script>
194 {% block js_block %}
195 {% endblock %}
```

Listing C.1: Základ kostry pro šablony webových stránek

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `app` ..... adresář se spustitelnou formou implementace
- `manual` ..... příručka pro nasazeni spustitelné formy implementace
- `src`
  - `mcra` ..... zdrojové kódy implementace
  - `OSRM` ..... Předzpracovaná data pro OSRM směrovací stroj
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF