VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO ORGANIZACI BĚŽECKÝCH ZÁVODŮ

A SYSTEM FOR ORGANIZING RUNNING RACES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JIŘÍ KALUS**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/20268/2016/xkalus02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Kalus Jiří, Bc.**
Obor: Informační systémy
Téma: **Systém pro organizaci běžeckých závodů**
A System for Organizing Running Races
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s požadavky na systém pro organizaci běžeckých závodů s podporou registrace závodníků a možností více stanovišť v průběhu závodu.
2. Prostudujte technologie MVC .NET a Windows Presentation Foundation pro tvorbu aplikací s architekturou klient-server.
3. Navrhněte architekturu systému pro organizaci běžeckých závodů s možností většího množství online a offline stanovišť.
4. Implementujte navržený systém na dané platformě.
5. Proveďte testování systému v reálných podmínkách.
6. Zhodnoťte dosažené výsledky a navrhněte možné další pokračování projektu.

Literatura:

- Petzold, C.: Mistrovství ve Windows Presentation Foundation, Computer Press, 2010
- Dále dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce se zabývá analýzou, návrhem a implementací systému pro organizaci běžeckých závodů s podporou registrace a možností více traťových stanovišť v průběhu závodu. Systém je komponován ze dvou webových aplikací a klientské desktopové aplikace. Administrační webová aplikace slouží k vytvoření a správě závodu. Za publikaci závodu s podporou registrace nese zodpovědnost prezentační webová aplikace. V průběhu závodu je možné snímat RFID čipy závodníků na více traťových stanovištích pomocí klientské aplikace. RFID čipy jsou snímány RFID čtečkou, zapojenou do cílovém zařízení. Je-li klientská aplikace připojena k internetu, jsou zaznamenané časy zobrazeny na webovém prezentačním rozhraní v aktuálním čase. Webové aplikace jsou implementovány technologií ASP.NET MVC. Autorizaci a autentizaci zprostředkovává ASP.NET Identity. Klientská aplikace je implementována technologií WPF.

Abstract

This master's thesis deals with the analysis, design and implementation of a system for organizing running races with the support of racers registration and the possibility of multiple online and offline checkpoints during the race. The system is composed of two web applications and a client desktop application. An administration web application is used to create and manage the race. A presentation web application with optional registration is responsible for publishing the race. It is possible to scan RFID chips of the racers on multiple track stations during the race using a client application. The RFID chips are scanned by an RFID reader connected to host device. If the client application is connected to the internet, the scanned times of racers are displayed on the presentation application in real time. Web applications are implemented with the ASP.NET MVC technology. Authorization and Authentication is provided by ASP.NET Identity. The client application is implemented using the WPF technology.

Klíčová slova

Organizace běžeckých závodů, registrace závodníků, tvorba kategorií a tras, konfigurovatelný registrační formulář, zobrazení real-time výsledků, RFID čipy, ASP.NET MVC, ASP.NET Identity, Entity framework, SQLite, WPF, Bootstrap, C#.

Keywords

Organization of running races, racers registration, creation of categories and routes, custom registration form, real time results, RFID chips, ASP.NET MVC, ASP.NET Identity, Entity framework, SQLite, WPF, Bootstrap, C#.

Citace

KALUS, Jiří. *Systém pro organizaci běžeckých závodů*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Systém pro organizaci běžeckých závodů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Další informace mi poskytli organizátoři závodů pan Ing. Petr Volný a Ing. Lukáš Tomčík. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Jiří Kalus
24. května 2017

Poděkování

Tímto bych rád poděkoval panu Radkovi Burgerovi za vstřícný přístup k vedení mé diplomové práce.

Obsah

Kapitola 1

Úvod

Cílem této práce je navrhnout a realizovat systém pro organizaci běžeckých závodů s podporou registrace závodníků a možností více online i offline stanovišť v průběhu závodu. Celý systém se skládá ze dvou webových a jedné desktopové aplikace. Administrační webová aplikace slouží pro vytvoření a správu závodu. Prezentační webová aplikace pro prezentaci závodu veřejnosti. Účelem desktopové aplikace je zaznamenávání časů závodníků na jednotlivých stanovištích závodu.

Účelem systému je webová prezentace výsledků běžeckých závodů. Prezentaci výsledků předchází vytvoření závodu, jeho správa a registrace závodníků. Fundamentálními kroky pro vytvoření závodu je stanovení kategorií, pravidel, stanovišť na trase nebo konfigurace položek v registračním formuláři. Prezentační webová aplikace nabízí veřejný detail závodu s možnou registrací a důrazem na vhodné zobrazení výsledků závodníků. Primárním účelem klientské aplikace je kolekce časů závodníků bez závislosti na internetovém připojení. Zaznamenané časy jsou v ideálním případě synchronizovány se serverem a zobrazeny na webových rozhraních. Zaznamenání času je provedeno sejmutím RFID čipu závodníka. RFID čipy jsou snímány RFID čtečkou, zapojenou do cílovém zařízení.

Následující text seznámí čtenáře s vývojem výše popsaného systému. V úvodní kapitole práce lze nalézt analýzu požadavků jednotlivých částí systému a zařazení produktu na trhu. Podrobnější popis použité technologie pro implementaci systému a snímání pomocí čipů je uveden v kapitole *Vybrané technologie*.

V kapitole *Návrh řešení systému pro organizaci závodů* je čtenář seznámen s architekturou a detailnějším popisem vybraných částí systému jako je databázové schéma, snímání čipů, autorizace, případy užití či grafické rozložení aplikací. Implementačně netriviální témata, mezi která patří autorizace uživatelů, objektově relační mapování či dynamická validace formulářových položek, jsou uvedena v kapitole *Implementace navrženého systému*. Následující kapitola obsahuje výsledky testování desktopové aplikace v reálných podmínkách. Pokračování a vhodné rozšíření systému je uvedeno v závěrečné kapitole *Rozšíření*.

Kapitola 2

Analýza požadavků

V této kapitole jsou shrnuty požadavky na systém pro organizaci závodů. Nejprve jsou uvedeny fundamentální požadavky na administrační a prezentační webové aplikace. Následují požadavky na klientskou aplikaci pro snímání čipů závodníků. Poslední sekce kapitoly se zabývá studií konkurenčních řešení a zařazení produktu na trh.

2.1 Definice požadavků

Tato diplomová práce se zabývá realizací webového informačního systému pro organizaci běžeckých závodů, přičemž součástí řešení je i klientská aplikace. Systém nejprve umožní vytvoření závodu hlavním organizátorem a poté je nakonfigurována klientská aplikace pro snímání čipů. Běžecké závody mají různou, často unikátní organizaci a je nutné vytvořit systém pro univerzální správu závodů. Časy jednotlivých závodníků jsou typicky u dlouhých běžeckých závodů zaznamenávány v průběhu na tzv. časových branách a je vhodné tyto výsledky zobrazovat v reálném čase ostatním uživatelům pomocí webového rozhraní.

Zaměříme-li se na hlavní požadavky při vytváření závodu, pak by každý závod měl mít určené kategorie a trasy pro závodníky. Konkrétním požadavkem je registrace závodníků a jejich správa. Druhým konkrétním požadavkem, založeným na závodě Hostýnská osma [1], je možnost vytvoření časových bran (stanovišť) na trase. Na časových branách se provede časová kontrola závodníků. V případě zájmu organizátora jsou závodníkům rozdistribuovány fyzické RFID čipy, jenž jsou na časových branách snímány. Časové brány jsou rozmístěné podél trasy, přičemž tato místa nemusí být nutně pokryta mobilním či jiným připojením na internet. Je tedy nezbytné, aby čipy mohly být snímány bez připojení a po připojení synchronizovány. Za snímání čipů a synchronizaci je zodpovědná klientská aplikace.

Po úspěšném vytvoření závodu nabídne webové rozhraní správu novinek a propozic. V momentě, kdy organizátor považuje závod za dostatečně zkompletovaný, je možné jej zveřejnit. Zveřejněním dochází ke zpřístupnění závodu všem uživatelům, kteří se mohou v daném termínu registrovat a stát se závodníky. Závodníci si později mohou zobrazit výsledky na webovém rozhraní. Součástí rozhraní je i galerie pro fotografie. Platba za registraci je v plné moci organizátora a systém za ni není nikterak zodpovědný.

V souhrnu musí systém umožnit webovou administraci závodu, webové rozhraní pro registraci závodníků a klientskou aplikaci pro snímání čipů. Požadavky na zmíněné části jsou podrobněji popsány v jednotlivých sekcích této kapitole.

[1] http://www.hostynskaosma.cz/

2.2 Požadavky na webové rozhraní

Webové rozhraní slouží k administraci a prezentaci závodu. Organizátor nejprve založí závod, přičemž uvede základní propozice (název, místo konání, limit závodníků, ...) a poté pokračuje definicí několika klíčových sekcí. Mezi klíčové sekce patří kategorie, trasy a registrační formulář pro případné závodníky. Po zveřejnění závodu je možná registrace závodníků. Každý zaregistrovaný závodník má možnost vyhledat všechny své závody. Jsouli vloženy do fotografie závodu, pak je uživatelům zpřístupněna galerie daného závodu. Součástí funkcionality systému je i vkládání formátovaných novinek pro závodníky.

2.2.1 Kategorie závodníků

Kategorie slouží k rozčlenění závodníků do skupin dle požadavků organizátora. Systém umožní vytvoření dvou základních typů kategorií, a to kategorie pro jednotlivce a kategorie pro týmy. V rámci každé kategorie je nezbytné uvést její název, určit akceptované pohlaví (muž/žena) a volitelně i věkové omezení. V týmu je vyjma zmíněných údajů také nutné uvést maximální počet členů, přičemž u pohlaví je možné zvolit mix (tým může být složen z obou pohlaví).

2.2.2 Trasy závodu

Trasa definuje běžeckou dráhu závodníků. Organizátorovi je umožněno vytvořit libovolné množství na sobě nezávislých tras. Ke každé trase je možné definovat několik stanovišť. Účel stanovišť není determinován a záleží na organizátorovi, zda bude na stanovišti probíhat občerstvení, časová kontrola či jiná činnost. Základní verze bude disponovat rozhraním pro vložení časových bran bez mapového podkladu. Žádoucí je také uvedení celkové vzdálenosti a výškového převýšení pro celou trasu.

2.2.3 Registrační formulář

Registrační formulář je velmi důležitá část realizace závodu. Organizátorovi je umožněna částečná konfigurace podoby formuláře dle jeho požadavků. Hlavní položky, mezi které patří *jméno*, *příjmení*, *ročník*, *pohlaví*, a *kontakt*, jsou předem definované a nelze je modifikovat. Pod statické položky je možné přidat položky vlastní. Nakonfigurovaný formulář poté vyplňují uživatelé při registraci.

Položka ve formuláři může nabývat různého datového typu. Prvním typem je obyčejný text. Rozšířením obyčejného textu je textové pole, které je vhodné pro delší texty. Dalším typem je celé číslo. Číselné vstupy lze upřesnit na desetinné číslo. Výlučný typ (ano/ne) je předposledním možným typem. Konečně posledním typem jsou možnosti, kdy organizátor nadefinuje několik možností a uživatel poté vybírá právě jednu z nich. Organizátor si u vytvořených položek stanovuje, zda je vstupní údaj povinný či volitelný. Některé položky mohou být chápany jako doplňkové služby/produkty, a proto by mělo být možné uvést u každé položky i její cenu, která se projeví v celkovém součtu startovného.

2.2.4 Správa čipů

Čip v informačním systému má fyzickou realizaci. Každý čip obsahuje unikátní identifikátor EPC, viz kapitola [3.7], který je asociovaný se startovním číslem závodníka. Tato relace může nabývat řádově stovky prvků, a proto je nezbytný import hodnot do informačního systému. Akceptovaný typ souboru je Excel, jelikož se jedná o nejběžnější formát uložení strukturovaných dat. Zároveň je však umožněno manuální vložení jednotlivých čipů.

Čipy jsou rozděleny do pojmenovaných sérii. Důvodem zařazení čipů do sérií může být více typů čipů nebo nutnost rozdělení do několika skupin dle definice organizátora. Příkladem jsou nízká startovní čísla (k nim asociované identifikátory čipů) pro jednotlivce a vysoká pro týmy.

Kombinace identifikátoru čipu a startovního čísla musí být v rámci série unikátní. Série s čipy jsou perzistentně uloženy v databázi a volitelně přidány do závodů.

2.2.5 Zanesení a vyhodnocení výsledků

Pro vyhodnocení výsledků je nutné přiřadit závodníkovi viditelný identifikátor. Identifikátory jsou čipy, startovní čísla nebo kombinace obojího. Organizátor manuálně přiřadí čísla či čipy nebo zvolí automatické možnosti přiřazení identifikátorů. První automatickou možností je vygenerování startovních čísel pro každého závodníka. Druhou možností je automatické přiřazení čipů (spolu se startovními čísly) k závodníkům.

Výsledky závodníků jsou implicitně seřazeny dle času, kdy nejrychlejší čas je ten nejlepší. Závodník si může pomocí grafického i tabulkového zobrazení porovnat svůj výsledek se soupeřem a vidět tak průběh závodu v závislosti na čase. V tabulce výsledků je možné vyhledávat a filtrovat závodníky podle kategorií. Na základě cílového času je vypočítáno celkové pořadí i pořadí v rámci kategorií. Stejné výpočty jsou uplatněny i pro ztrátový čas na nejlepšího závodníka/tým.

Závodníci mohou nabývat stavů *dokončil*, *neodstartoval*, *diskvalifikován* nebo *nedokončil*. Má-li na konci závodu kdokoliv jiný status než *dokončil*, je to indikace neúspěšného dokončení závodu a jeho výsledek není brán v potaz.

Uložit výsledek závodníka lze dvěma způsoby. Prvním je sejmutí čipu závodníka pomocí klientské aplikace a přeposlání na server. Limity a možné problémy tohoto způsobu zanesení jsou podrobněji rozepsány v kapitole [2.3.2]. Druhým způsobem je zanesení výsledků přímo v administračním rozhraní. Zde je možné uvést číslo nebo čip závodníka, jeho stav, případné stanoviště (časovou bránu) a čas.

2.2.6 Noviny a galerie

Zobrazení novinek a propozic v prezentačním rozhraní patří mezi sekundární požadavky. Novinky mohou být formátovaně vytvářeny v administračním rozhraní. Součástí výstupu jsou i fotografie. Náhled novinek je poté zobrazen v prezentačním rozhraní, kde uživatelé mohou přejít na detail novinky. Propozice disponují stejnou funkcionalitou, tedy jsou formátované, avšak je vytvořena pouze jedna instance.

Posledním požadavkem na webové rozhraní je galerie pro fotografie ze závodů. Fotografie je možné uložit a v případě potřeby i smazat. Základní verze, která je implementována v rámci této práce, umožňuje správu fotografií pouze pro jedno album a počet nahraných fotografií není limitován.

2.3 Požadavky od klientské aplikace

Základní funkcionalita klientské aplikace je snímání čipů během závodu. Sekundárně se vyžadují možnosti zobrazení seznamu přihlášených závodníků, vytvořených bran a všech nahraných čipů. Aplikace musí být zabezpečena proti nežádoucímu použití a její činnost nesmí být závislá na internetovém připojení. Přístup k aplikaci je povolen pouze organizátorem pověřeným osobám.

2.3.1 Zabezpečení aplikace

Aplikace je stáhnutelná jen z administračního rozhraní, které je zpřístupněno pouze organizátorům závodů. Stáhnutím a instalací na cílové zařízení ovšem vzniká riziko neoprávněného použití. Cílová zařízení, notebooky, jsou často pouze zapůjčena a jejich majitelé mohou, byť nedopatřením, zaslat neplatné časy. Aplikace je tedy chráněna formulářem, kdy přihlašovací údaje nastavuje organizátor závodu v administračním rozhraní.

2.3.2 Zanesení výsledků

Časy závodníků jsou snímány na časových branách, viz kapitola [2.2.2]. Uživatel nejprve vybere konkrétní bránu a poté započne samotné zanesení časů závodníka. Každý závodník obdrží při akreditaci svůj RFID čip. Časová brána je vybavena bezkontaktní čtečkou. Čip je typicky fyzickou realizací podobný hodinkám. Závodník přiloží čip ke čtečce a dojde k sejmutí čipu. Aplikace sejmutí čipu detekuje, přečte jeho přijatá data, zaznamená čas sejmutí, uloží jej databáze a pokračuje zasláním času na webové rozhraní. Primární čas je čas v administračním rozhraní. Každý klient však může mít odlišný čas. Organizátor proto musí provést manuální synchronizaci s primárním časem. Sejmuté informace jsou vhodnou formou zobrazeny závodníkům. Dále popsáno v kapitole [4.6.1].

2.3.3 Synchronizace

Pro úspěšnou činnost aplikace je nutné ji alespoň jednou připojit k internetu. V době průběhu závodu musí být v aplikaci staženy základní propozice závodu, především identifikátor závodu a časových bran. Změna údajů závodu v administračním rozhraní by měla být v co nejkratším čase reflektována v aplikaci. Aplikace je synchronizována při každém spuštění. Časy nově zaregistrovaných závodníků, kteří nejsou synchronizováni s aplikací, mohou být zaznamenány a korektně přeposlány na webové rozhraní.

Po úvodní inicializaci závodu může aplikace fungovat bez připojení k internetu. Časy závodníků jsou zaznamenávány a uloženy. Při opětovném připojení k internetu proběhne synchronizace a neodeslané časy závodníků jsou odeslány.

Všechny zaznamenané časy jsou paralelně ukládány do souboru umístěného na ploše operačního systému. Obsahem souboru je záloha výsledků, která může být do aplikace kdykoliv znovu nahrána. Takto nahrané časy lze opět odeslat na webové rozhraní. Pozdější synchronizaci lze omezit v administračním rozhraní, kde lze zakázat příjem výsledků z klientských aplikací.

2.4 Existující řešení a zařazení produktu na trhu

V současné době existuje několik společností na měření a správu výsledků, avšak žádná nenabízí vytvoření a správu samotného závodu. Dle mého názoru patří mezi přední zástupce portály Sport-base [2], Sportsoft timing [3] a Championchip [4]. Každá ze společností nabízí profesionální měření časů závodníků. Výsledky jsou typicky převedeny do PDF nebo jednoduše zformátovány pro snadné vložení na cílovou stránku závodu. Součástí služeb je i registrace závodníků a jejich správa. Tato řešení jsou často vhodná pro závody většího rozsahu, kde je klíčové zaznamenání přesného času závodníků.

Koncept je vždy takový, že organizátor požádá o spolupráci zmíněnou společnost, která zajistí měření časů a v případě poptávky umožní registraci a správu závodníků. Dnes považujeme za standard mít pro veřejné závody i webové stránky. Organizátor tak musí mít vlastní webové stránky, kde zpřístupní registraci, propozice, trasy, výsledky a další části náležící k závodu.

Vyvíjený produkt v této práci minimalizuje čas potřebný k vytvoření elektronické prezentace běžeckých závodů. Produkt cílí na pořadatele lokálních závodů, kteří potřebují administrativní rozhraní pro vytvoření a reprezentaci samotného závodu. Rozpočet takových závodů bývá často velmi nízký a měření probíhá individuálním způsobem. Individuálním způsobem je například ruční časomíra, kdy pořadatelé v časové bráně zapisují časy závodníků.

[2] http://www.sport-base.cz/
[3] http://sportsoft.cz/
[4] http://www.championchip.cz/

Kapitola 3

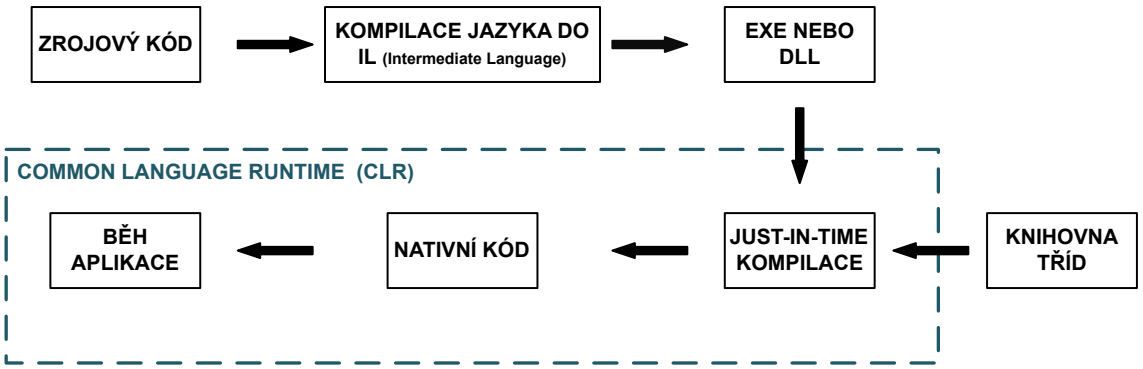
Vybrané technologie

Obsahem kapitoly je studie technologií a jejich uplatnění v rámci zadané diplomové práce. Nejprve jsou popsány technologie pro vývoj informačního systému. Podrobněji pouze technologie na straně serveru. Následně je čtenář seznámen s technologiemi pro vývoj desktopové aplikace na operačním systému Windows. Celý systém využívá platformu .NET. Obsahem závěrečné sekce je sumarizace použitých technologií.

3.1 Platforma .NET

Platforma .NET [15] je složena ze dvou hlavních částí. První částí je běhové prostředí Common Language Runtime (CLR) pro vykonávání kódu. Druhou částí je knihovna tříd Framework Class Library (CLS), což je knihovna tříd a rozhraní pro přístup k funkcím systému.

Prostředí CLR je virtuální stroj, ve kterém probíhá překlad do nativního jazyka. Průběh překladu je zobrazen na obrázku [4.11] níže.



Obrázek 3.1: CLR - kompilace a spuštění

Součástí prostředí je správa paměti, bezpečnost, správa výjimek, vláken apod. CLR je obdoba virtuálního stroje programovacího jazyka Java. Zdrojové kódy tedy nejsou kompilovány přímo do nativního jazyka, ale do intermediálního jazyka (Intermediate Language - IL). Jazyk C# kódu, ve kterém je tato práce implementována, je v počáteční fázi kompilován do tzv. managed code, který je součástí assembly. Výsledkem první fáze překladu je knihovna s příponou `.dll` nebo spustitelný kód s příponou `.exe`. V případě

požadavku na spuštění managed code systém detekuje, že se jedná o objekt zkompilovaný do jazyka IL a spustí Just-In-Time (JIT) kompilátor. JIT kompilátor vygeneruje skutečné instrukce cílové platformy.

Důležitým prvkem CLR je podpora společného typového prostoru (Common Type System - CLS). CLS vymezuje deklaraci, použití a správu typů. Poskytuje objektově orientovaný model, definuje pravidla, která zaručí interakci objektů napsaných v odlišných jazycích, a poskytuje také knihovnu primitivních typů jako například Boolean, Char, Byte, Int32 a další. Níže je uveden seznam základních součástí platformy .NET:

- ASP.NET - serverová webová platforma pro vývoj a provozování webových aplikací.
- ADO.NET - množina tříd pro přístup k datům a tvorbu databázových aplikací.
- LINQ - technologie pro unifikované dotazování nad různými daty (databázové rozhraní, kolekce, XML atd.).
- Windows Forms - starší knihovna tříd pro tvorbu grafického rozhraní desktopových aplikací.
- WPF - novější knihovna tříd pro tvorbu grafického rozhraní desktopových aplikací.
- WCF - framework pro komunikaci mezi aplikacemi a vytváření servisně orientovaných aplikací.

Knihovna CLS je základní pilíř, na kterém jsou postaveny aplikace platformy .NET, komponenty i ovládací prvky. Jsou zde knihovny jako WPF, Windows Forms, ASP.NET a další.

Primární prostředí pro běh platformy .NET je prostředí Windows. V současné době je však možný běh i v modulární verzi .NET Core [1], který je podporován na operačním systému Linux, Windows i Max OsX. Dalším z multiplatformních projektů je Mono [2], který nepoužívá běhové prostředí CLR, ale má vlastní kompilátor. Například prostředí Xamarin, umožňující vývoj mobilních aplikací pro Android i iOS, podporující jazyky z rodiny .NET, je založeno na prostředí Mono.

3.2 ASP .NET Framework

Jádrem platformy ASP .NET [15] pro webové aplikace je platforma .NET, viz sekce [3.1]. ASP. NET je serverová webová technologie [7] určená pro vývoj a provozování webových aplikací, webových služeb a webových stránek. Jedná se o nástupce technologie Active Server Pages (ASP). Základním elementem je .NET Framework, který vytváří prostředí pro běh webových aplikací a nabízí potřebné knihovny. Firma Microsoft, jenž vyvíjí tento framework, zpřístupnila jeho zdrojový kód, čímž se stal veřejně dostupný (open source). Pro vývoj webového produktu je možné zvolit takový jazyk, který je kompatibilní s Common Language Runtime (CLR, viz sekce [3.1]) prostředím, což je například: C#, VisualBasic či C++. ASP .NET. Framework disponuje následujícími aplikačními rámci:

- WebForms - technologie založená na modelu řízeném událostmi. Webová aplikace je tvořena ovládacími prvky a jejich událostmi. Logika vývoje webové aplikace je podobná technologii Windows Forms, viz kapitola [3.5].

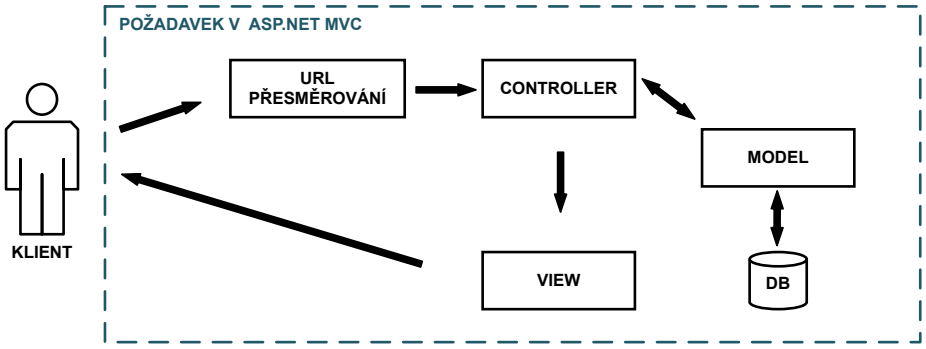
[1] https://docs.microsoft.com/en-gb/dotnet/articles/core/index
[2] http://www.mono-project.com/

- WebPages - alternativa skriptovacích jazyků jako je například PHP. Tato technologie kombinuje HTML kód se serverovým jazykem (kompatibilním s CLR).
- MVC - technologie pro vývoj webové aplikace respektující návrhový vzor MVC. Detailnější popis je uveden níže v této kapitole [3.2.1].
- Single Page Application - celá webová aplikace je tvořena jednou stránkou. SPA minimalizuje použití serveru a klade důraz na klientskou část, která je tvořena jazykem Javascript. Server je implementován prostřednictvím Wep API [3]

Webová aplikace této diplomové práce je vyvíjena aplikačním rámcem ASP. NET MVC.

3.2.1 ASP. NET MVC

Platforma ASP.NET MVC [6] je jedním z aplikačních rámců ASP.NET. Tato platforma implementuje architektonický vzor MVC, jenž se skládá ze tří logických částí: Model-View-Controller. Jednotlivé části spolu se schématem jsou podrobně popsány níže:



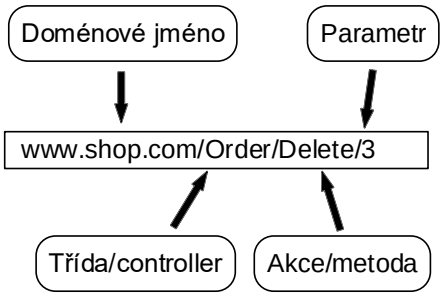
Obrázek 3.2: Zpracování požadavku v ASP.NET MVC architektuře

- Model - objekty v této části architektonického vzoru implementují logiku pro aplikační datovou doménu. Typicky jsou zde definována validační omezení formou anotací a vybrané operace nad daty.
- View - česky pohled, je část, která zobrazuje výstup uživateli. Tato část neslouží jen k zobrazení uživatelského rozhraní, ale zároveň s uživatelem interaguje. Uživatelovy příkazy přeposílá na logickou část jménem controller, která příkaz zpracuje a vrátí výsledek uživateli. Pohled je nejčastěji reprezentován HTML stránkou, do které lze vkládat příkazy v jiném jazyce (například C#). Stránka je zkompilována na serveru a vrácena ve standardním tvaru, jehož zobrazení podporuje webový prohlížeč.
- Controller - česky řadič, je komponenta, která přijímá uživatelské vstupy přeposlané z view a pracuje s modelem. Je to tedy mezivrstva mezi view a modelem. Detailnější informace o této logické části jsou uvedeny níže.

[3] Web API - REST rozhraní pro komunikace a zpracování dat v serializačním jazyce XML nebo JSON.

3.2.2 Controller analýza

Komunikace mezi logickými vrstvami view a controller nemusí být na první pohled zřejmá. Vrstva controller je zastoupena třídami, jejichž implementované metody jsou nazývány *akce (action)*. Pro vykreslení webové stránky předá view požadavek vrstvě controller. Zaměřme se na požadavek, který je i s implementací uveden níže:




```
public class OrderController: Controller
{
    ...
    public ActionResult Delete(int id)
    {
        ...
        return View();
    }
}
```

Obrázek 3.3: Rozložení adresy a implementace v ASP.NET MVC architektuře

První částí je doménové jméno webového serveru. Druhá část za lomítkem je controller název, přičemž třetí část je název metody/akce, která se má zavolat. Konečně poslední (libovolnou) částí je parametr pro upřesnění dotazu a předání další informace. Typicky se jedná o identifikátory entit z modelu. Webový server pomocí reflexe nalezne odpovídající controller a jeho akci, ve které předá identifikující parametr. Rozpoznání akce je určeno jejím názvem, anotacemi, počtem nebo typem parametrů.

Platforma ASP.NET MVC nabízí rozsáhlou funkcionalitu a výše zmíněné aspekty patří mezi nejzákladnější. Tento typ architektury poskytuje rozdělení hlavní logiky vzhledové části a modelu dat. Benefitem je u rozsáhlejších aplikací nesporně přehlednější zdrojový kód. Snadnější je tak vývoj i údržba a díky odlučitelnosti logických částí i testování. Negativem je delší učící křivka.

3.3 ASP.NET Identity

Pro autentizační a autorizační činnosti platformy ASP .NET byl vyvinut framework ASP.NET Identity. Do verze .NET 4.5.1 bylo možné pracovat pouze s mechanismy Forms Authentication [4] a Membership and Role provider [5], dále jen Membership. Vzhledem ke komplexnosti a diferenci možností implementace je vhodné uvést stručnou historii a návaznost na aktuální Identity 2.0 framework [5].

První verze ASP 1.0 implementovala Forms Authentication, což je mechanismus na uchovávání informace o identitě přihlášeného uživatele pomocí tiketů nebo url. Databázové napojení je nutné implementovat separátně. Navrhnuto je primárně pro Web Forms a na ostatní technologie je nasazení komplikovanější. S novou verzí ASP 2.0 byl vyvinut framework Membership, který byl ve své základní verzi určen především pro práci s Forms Authentication. Membership slouží ke zpřístupnění databáze uživatelů a členství ve skupinách. Verze ASP 4.5 obsahuje novou generaci dvou výše zmíněných technologií.

[4] https://msdn.microsoft.com/en-us/library/7t6b43z4.aspx
[5] https://msdn.microsoft.com/en-us/library/aa354509(v=vs.110).aspx

Forms authentication jsou nahrazeny technologií OWIN security. Tato nová technologie funguje se všemi částmi ASP.NET (Web Forms, ASP.NET MVC, Web Pages, ...). Podporuje otevřené standardy pro autentizaci, jako je například OAuth 2.0 nebo OpenID, a je platformě nezávislá.

Membership je nahrazen Identity, které definuje rozhraní pro přístup k informacím o uživatelích, o jejich rolích, externích identitách apod. Identity se tedy nestará o autentizaci. Je to jen rozhraní, které zpřístupňuje údaje. Autentizace a autorizace má na starosti OWIN security. Ačkoliv je Membership téměř deset let stará technologie a především má již svého nástupce, je i dnes používána a nasazována. Domnívám se, že důvodem je zřejmě velké množství návodů a relativní jednoduchost implementace (v porovnání s Identity). Mezi hlavní výhody Identity patří lepší rozšiřitelnost. Například uživatelský profil technologie Membership obsahuje pouze jméno a heslo. Ostatní vlastnosti musejí být spravovány externě. Membership je principiálně založen na relační databázi. Identity podporuje různá úložiště (relační databázi, ravenDB [6], Azure Table Storage [7]). Jedním z praktických rysů Identity je zabudovaná podpora externích autorit (Facebook, Google, Twitter, ...). V neposlední řadě je zde podpora vícefaktorové autentizace, jednorázová hesla nebo kvalitnější ukládání hesel. Následkem modularity je komplikovanější nasazení a rozšiřitelnost. Vše se inicializuje v kódu, nikoliv v konfiguračních souborech, jak tomu bylo u minulých mechanismů. Je také nutné si vytvořit vlastní grafické rozhraní pro přihlášení a správu uživatelů.

Po analýze, jenž je popsána výše, byly zvoleny modernější technologie OWIN security a ASP.NET Identity verze 2.0. Tyto mechanismy mají sice delší učící křivku, avšak reflektují aktuální potřeby pro autorizační a autentizační činnosti jako je podpora externích autorit, vícefaktorové přihlášení nebo jednodušší rozšiřitelnost uživatelského profilu o předem definované vlastnosti.

3.4 Entity framework

Pro práci s databází je možné využít technologii Entity framework (EF). EF [12] je open-source a zajišťuje objektově relační mapování. Vyvojáři mohou komunikovat jen s modelem za použití objektově orientovaných technik. Manipulace s daty je tak jednodušší, protože EF převede dotaz z objektového rozhraní (typicky dotaz za pomocí technologie LINQ [8]) na dotaz nad relační databází. V rámci EF můžeme u objektově orientovaných aplikací zvolit dva přístupy:

- Code first - na základě objektů EF vygeneruje databázové schéma. Výhodou je snadná manipulace s relacemi. Nevýhodou mohou být složitá integritní omezení.
- Database first - na základě databázového schématu EF vygeneruje odpovídající třídy v modelu aplikace.

V rámci této diplomové práce je použit Entity Framework s Code first přístupem.

[6] https://ravendb.net/
[7] https://docs.microsoft.com/en-us/azure/
[8] https://msdn.microsoft.com/en-us/library/bb308959.aspx

3.5 Windows Presentation Foundation

Windows Presentation Foundation (WPF) [14], jenž je součástí .NET platformy od verze 3.0, je platforma pro tvorbu desktopových aplikací. Jedná se o logického nástupce technologie Windows Forms [9].

WPF zavádí nezávislou jednotku délky Device Independent Pixel (DIP) a všechny elementy vytváří vektorově. Díky tomuto aspektu je výsledná aplikace nezávislá na rozlišení zobrazovacího zařízení. Původní grafické rozhraní Graphics Device Interface(GDI) [17] ve Windows Forms je nahrazeno a pro vykreslování grafického rozhraní je využit DirectX. V důsledku akcelerované grafiky má aplikace menší výpočetní režii na procesor.

Významná technologie, jenž je součástí WPF, je Data binding [9]. Data binding umožňuje navázání a synchronizaci dat mezi zdrojem dat (modelem) a grafických uživatelským rozhraním. V rámci WPF technologie je možné rozlišit 4 módy a synchronizace dat:

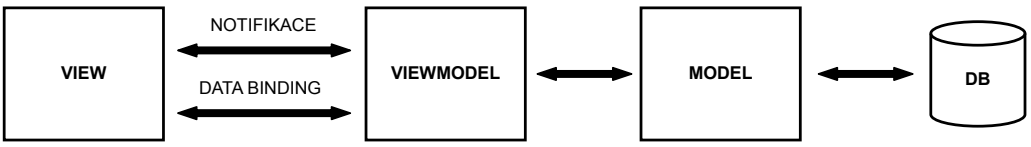
- One way - uživatelské rozhraní se pouze synchronizuje se změnami hodnot ve zdroji dat. Hodnoty tedy nelze z uživatelského rozhraní modifikovat.
- Two way - uživatelské rozhraní a hodnoty ve zdroji dat jsou synchronizovány navzájem. Změní-li uživatel hodnotu v uživatelském rozhraní, pak se změní i hodnota ve zdroji dat a naopak.
- One time - uživatelské rozhraní přečte hodnotu ve zdroji dat pouze jednou a poté jsou z pohledu uživatelského rozhraní veškeré změny ve zdroji dat ignorovány.
- One way to source - jedná se o protiklad módu One way. Pouze uživatelskému rozhraní je povoleno modifikovat hodnotu ve zdroji dat.

Pro uživatelské rozhraní je ve WPF používán jazyk XAML, který je odvozený od značkovacího jazyka XML. Všechny prvky, které definujeme v jazyce XAML, lze zapsat i v jazyce C# či Visual Basic. V kombinaci s Data binding a událostmi lze docílit oddělení implementace uživatelského rozhraní od modelu dat, což vede k možnosti poměrně nezávislého vývoje obou vrstev.

[9] https://msdn.microsoft.com/en-us/library/dd30h2yb(v=vs.110).aspx

3.5.1 MVVM vzor

Model-View-Viewmodel [2] je návrhový vzor pro WPF aplikace, jehož hlavním cílem je oddělit logiku aplikace od uživatelského rozhraní. Tento návrhový vzor vychází z principů vzoru MVC a navrhl jej softwarový architekt John Gossman[18]. Jednotlivé logické části vzoru jsou společně se schématem popsány níže [3.4].



Obrázek 3.4: Schéma návrhového vzoru MVVM pro WPF aplikace

- View - reprezentuje uživatelské rozhraní. Definice toho, co se zobrazí uživateli. Všechny ovládací prvky jsou přes datový kontext (zdroj dat) namapovány na vlastnosti ve viewmodel. Tato vrstva také obsluhuje efekty, validace či animace.
- Model - popisuje data, se kterými aplikace pracuje, jejich logiku a případně validaci.
- Viewmodel - logická vrstva mezi view a Model. Představuje prostředníka, který adaptuje Model pro potřeby view. Vrstva viewmodel je zodpovědná za stav view avšak bez přímé interakce mezi jednotlivými prvky ve view. Ovládací prvky jsou pomocí Data binding [9] propojeny a synchronizovány s viewModel. Pro validní propojení vrstev musí být implementováno rozhraní `INotifyPropertyChanged`, které zajistí propagaci změn hodnot.

Klientská aplikace pro snímání čipů využívá návrhový vzor MVVM.

3.6 Knihovna SQLite

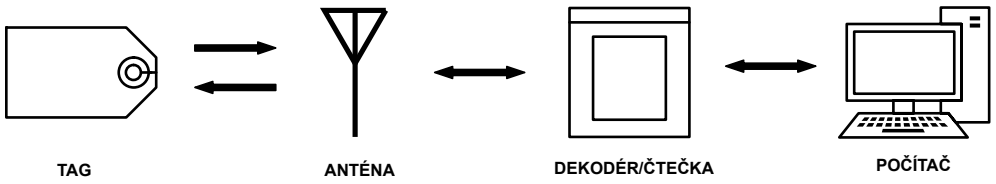
SQLite je knihovna menšího rozsahu [19], která je napsána v jazyce C a obsahuje téměř celý relační databázový systém. Hlavním rysem je přímé připojení knihovny k aplikaci, nikoliv oddělený běh, jak je to typické u architektury klient-server. V praxi je databáze tvořena souborem, nad kterým lze provádět relační databázové dotazy. Podpora standardu SQL-92 není kompletní a některé funkce nejsou implementovány [20]. Celá knihovna je distribuována pod licencí public domain a podporována ve většině nejpoužívanějších programovacích jazycích (C#, C++, Java, C, Python, PHP a další). Výhodou je lokální nenáročná instalace a použití, čímž se knihovna stala jedním ze standardů pro jednoduchou lokální databázi. Jako zástupce použití můžeme jmenovat prohlížeče Chrome a Safari, komunikátor Skype nebo velké množství aplikací pod operačním systémem Android.

Knihovna SQLite byla použita v klientské aplikaci pro lokální uložení dat o závodu.

3.7 RFID technologie

Radio Frequency Identification (RFID) je obecně používaný pojem pro identifikaci objektů pomocí elektromagnetických vln na rádiové frekvenci. Informace je uložena na mikročipu,

který je připojen k anténě a zalit do substrátu. Dohromady tyto složky tvoří tzv. RFID tag, který je schopen uloženou informaci vyslat do čtečky. Vysílač čtečky vysílá elektromagnetické vlny. Anténa RFID tag přijme signál a usměrněný elektrický proud dobije kondenzátor. Výsledkem je odeslání informace zpět do čtečky. Nejběžnější informací je unikátní standardizovaný 96-bitový kód - Electronic Produkt Code (EPC). Čtečka přemění vyslané radiové vlny přijaté od RFID tagu na formu, která může být dále zpracována.



Obrázek 3.5: Princip komunikace RFID technologie

Na obrázku [3.5] je uveden základní princip komunikace RFID tagu s řídícím počítačem. Hardwarová realizace může být však různá a často se můžeme setkat s případy, kdy je anténa součástí čtečky.

3.7.1 Frekvence

Systém RFID podporuje několik frekvencí [21]. Frekvence silně ovlivňuje vlastnosti čipu (dosah, rychlost čtení a zápisu, použitelnost v různém prostředí apod.), a proto je nutné ji stanovit již v návrhu systému. Frekvenci ovlivňuje typ a velikost antény RFID tagu. S rostoucí frekvencí jsou typicky nutné menší antény, ovšem s vyšší frekvencí narážíme na problém s elektromagnetickým rušením a vzniká tak nepřesnost především při snímáním z kovových materiálů a tekutin. Rozlišujeme čtyři typy frekvencí. Nízkofrekvenční, 125 KHz a 135 KHz, jsou používány například pro kontrolu přístupu nebo imobilizéry v automobilech. Rychlost a přenesená data jsou velmi nízká. Nespornou výhodou je jejich vyšší odolnost proti rušení a nízká cena. Vysokofrekvenční, 13.56 MHz, jsou používány například pro platební brány nebo označování zavazadel při přepravě. Pro vyšší čtecí dosah se jedná o nejrozšířenější typ používané frekvence. Ultrafrekvenční, 860 MHz až 960 MHz, jsou používány například pro elektronické mýtné, parkovací karty nebo současnou identifikaci více produktů. Nevýhodou je obtížné čtení na kovových podložkách. Pro úplnost uvedeme i mikrovlnou frekvenci, 2.45 GHz a 5.8 GHz, která dokáže bezdrátový záznam a přenos dat v reálném čase.

3.7.2 Využití RFID u běžeckých závodů

Čipy (RFID tagy) jsou velmi rozšířené na sportovních akcích, především běžeckých závodech s větším počtem účastníků. U takových závodů typicky není vhodné používat manuální stopky, ale čipy, které mají závodníci připevněné na sobě. Mezi dva nejčastější případy uchycení patří integrace čipu přímo do závodníkova čísla nebo je čip zapuštěn do umělohmotného náramku.

U časových bran je umístěno čtecí zařízení, které snímá procházející závodníky. Kvalitnější a finančně náročnější je průchozí brána se zabudovanými čtecími zařízeními. Obvykle se jedná o vysokofrekvenční či ultrafrekvenční řešení. Závodník bez jakékoliv interakce projde branou a jeho čas se zaznamená. Další možností je použití levnější, nízkofrekvenční varianty. U některých, převážně dlouhých vytrvalostních přechodů, není přesnost

na sekundy vyžadována. Závodník na časových branách přistoupí přímo ke čtecímu zařízení a na blízkou vzdálenost dojde k sejmutí čipu.

Tato práce je navrhnuta pro druhou, nízkofrekvenční variantu. Závodníci mají připevněný čip formou náramku na ruce a jednotlivě přistupují ke čtecím zařízením. Pro zefektivnění snímání je na jedné časové bráně umístěno několik čtecích zařízení.

3.8 Sumarizace použitých technologií

Pro informační systém je využita technologie ASP.NET MVC. Objektově relační mapování zajišťuje Entity framework. Jako relační databáze byla zvolena databáze MS-SQL. Pro autentizaci a autorizaci uživatelů byly vybrány technologie OWIN security a ASP.NET Identity. Klient pro stolní počítače využívá technologii WPF, kdy pro code-behind je zvolen jazyk C# a pro vzhled jazyk XAML. Lokální databáze je realizována pomocí knihovny SQLite.

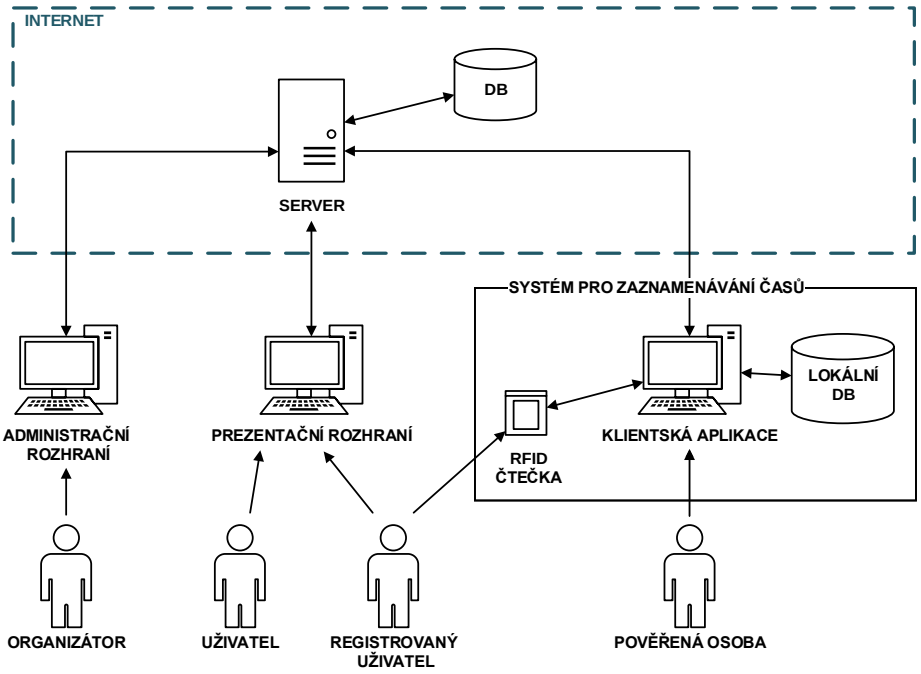
Kapitola 4

Návrh řešení systému pro organizaci závodu

Tato kapitola nejprve čtenáře seznámí s celkovým schématem systému a popisem jednotlivých částí. Dále jsou podrobněji popsáni aktéři a jejich činnost. Následující sekce čtenáře seznámí s význačnými návrhy jako je databázové schéma, snímání RFID čipů nebo autorizace klientské aplikace. Závěrem je uvedeno základní grafické rozložení administračního a desktopového rozhraní.

4.1 Schéma a popis jednotlivých elementů

Celý systém se skládá ze sedmi částí: serverové databáze, samotného serveru, administračního a prezenčního rozhraní, lokální databáze, klientské aplikace a RFID čtečky. Jednotlivé části a jejich vztah je zobrazen na schématu níže [4.1].



Obrázek 4.1: Schéma navrhovaného systému pro organizaci závodu

Celý systém organizace závodů s podporou registrace závodníků je tedy rozdělen do dvou hlavních částí. Administrační rozhraní, anglicky backend, slouží k vytvoření a správě závodu. Organizátor vytvoří a zveřejní závod, který se tak zobrazí široké veřejnosti v prezenčním rozhraní, anglicky frontend. Prezenční rozhraní můžeme chápat jako alternativu osobních webových stránek konkrétního závodu.

Klientská aplikace je doplňková služba pro snímání časů závodníků. Po vytvoření závodu je služba zpřístupněna v administračním rozhraní. Organizátor aplikaci stáhne a poté nainstaluje na cílové zařízení. Jakmile proběhne inicializační synchronizace se serverem, je aplikace připravena k použití. V případě zaznamenávání časů pomocí čipů je připojena RFID čtečka k cílovému zařízení a může dojít ke snímání čipů a přeposílání výsledků na server. Klientská aplikace s lokální databází a RFID čtečkou je součásti systému pro zaznamenávání času.

4.2 Případy užití systému

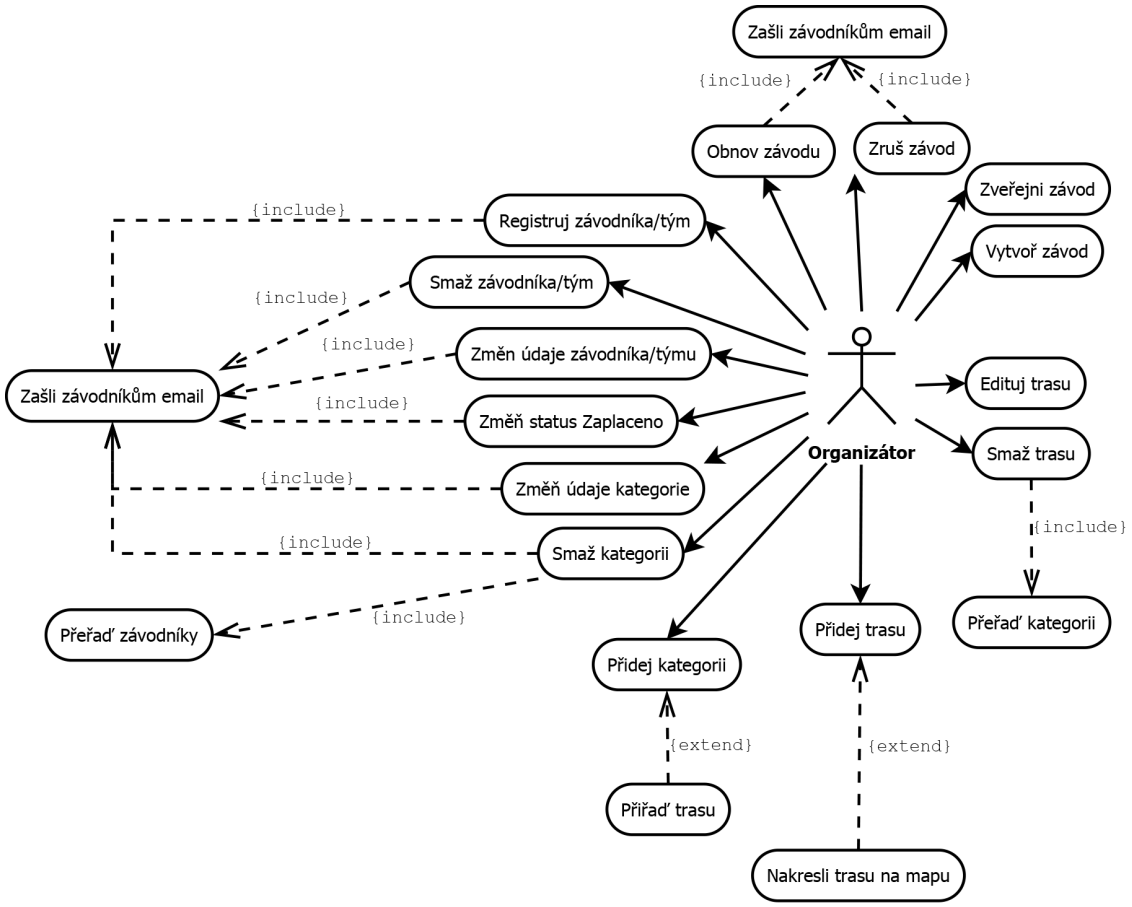
Diagram případů užití se používá k popisu chování systému z hlediska uživatele a zachycuje, které typy uživatelů se systémem pracují a jaké činnosti v rámci systému vykonávají [1]. Popisem interakce mezi uživatelem a systémem umožňuje znázornění funkčních požadavků na systém. Popis rolí uživatelů a jejich činnost je uvedena níže v sekci [4.2].

Obrázek 4.2: Výčet uživatelů systému

V celém systému pro organizaci běžeckých závodů účinkují čtyři uživatelé. Role Organizátora zastupuje uživatele, který vytváří závod a je zodpovědný za celkový průběh závodu. Ke správě závodu v administračním rozhraní má přístup pouze organizátor závodu. Pověřená osoba zastupuje uživatele, který pracuje s klientskou aplikací pro zaznamenávání výsledků. Taková osoba je určena Organizátorem a typicky se nachází v průběhu závodu na některém ze stanovišť (časových bran). V prezenčním rozhraní vystupuje role Uživatel a Registrovaný uživatel. Uživatelé si mohou prohlížet závody a případně se zaregistrovat.

Organizátor

Činnost organizátora je pro zpřehlednění rozdělena do dvou diagramů případů užití. V prvním jsou popsány základní akce pro správu závodu a správu závodníků. Druhý diagram pak doplňuje kompletní činnost organizátora.



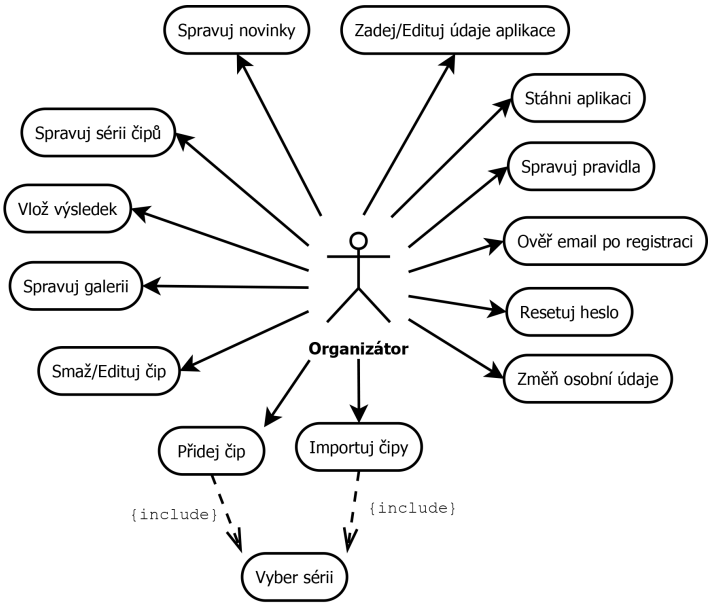
Obrázek 4.3: Diagram případu užití systému - základní činnosti Organizátora

Základní operace nad závodem jsou vytvoření, zveřejnění, skrytí, zrušení a obnovení. Závod je bezprostředně po vytvoření skryt uživatelům. Organizátor má možnost vše zkontrolovat, přidat či editovat další dílčí části (propozice, čipy, pravidla, novinky atd.) a až poté závod zveřejnit.

Další důležitá činnost je práce s kategoriemi. Kategorii může být přiřazena trasa. Před smazáním trasy musí být změněny trasy kategorií, jenž jsou s kategorií asociovány. Tato činnost je v diagramu případu užití nazvána „Přiřaď kategorii“.

Poslední činností je správa závodníků. Jakákoliv modifikace závodníkových dat je zaslána na jeho email. Organizátor může registrovat závodníky nebo týmy do nastavené kapacity závodu. Každému závodníkovi je po registraci doručen email s výzvou k zaplacení startovného. Jakmile organizátor obdrží platbu nebo jinak klasifikuje zaplacení, lze změnit status závodníka na zaplaceno.

Doplnění kompletní činnosti organizátora je znázorněno na obrázku [4.4].



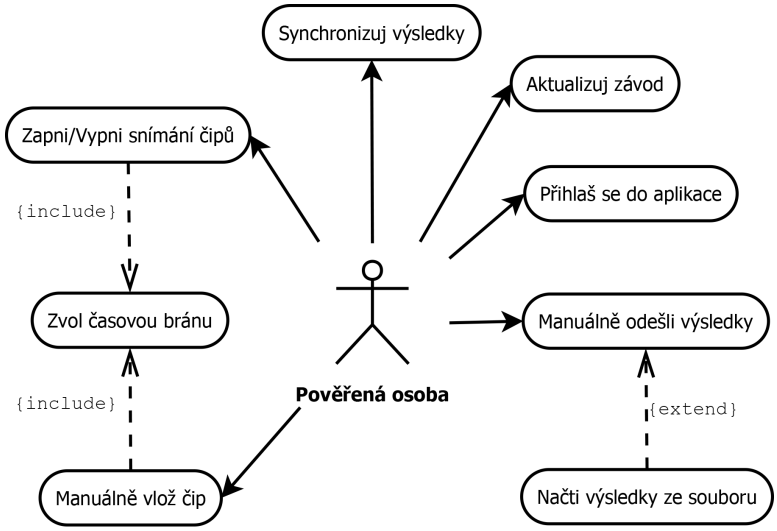
Obrázek 4.4: Diagram případu užití systému - doplňující činnosti organizátora

Do této sekce činností spadá práce s novinkami, pravidly, čipy a manipulace s osobními údaji. Novinky a pravidla mohou být neomezeně přidávány v administračním rozhraní, přičemž výsledky se promítnou v prezenčním rozhraní. V prezenčním rozhraní si je mohou prohlédnout závodníci či běžní uživatelé. V administraci lze inicializovat a stáhnout klientskou aplikaci. Inicializací je míněno nastavení jednoznačného identifikátoru a hesla pro přístup do aplikace. Úspěšná registrace je podmíněna ověřením emailu. Je-li email úspěšně ověřen, je uživateli umožněno přihlásit se do systému. Série čipu je pojmenovaný seznam, jehož obsahem je kombinace čísla čipu a startovního čísla.

Dosud popsané činnosti jsou specifické pro daný závod a s každým vytvořením závodu vznikají nové instance. To však neplatí pro čipy, které zastupují reálný fyzický produkt. Čipy jsou persistentní a mohou být použity v libovolném závodě.

Pověřená osoba

Diagram případu užití systému pro činnosti pověřené osoby je zobrazen níže [4.5].

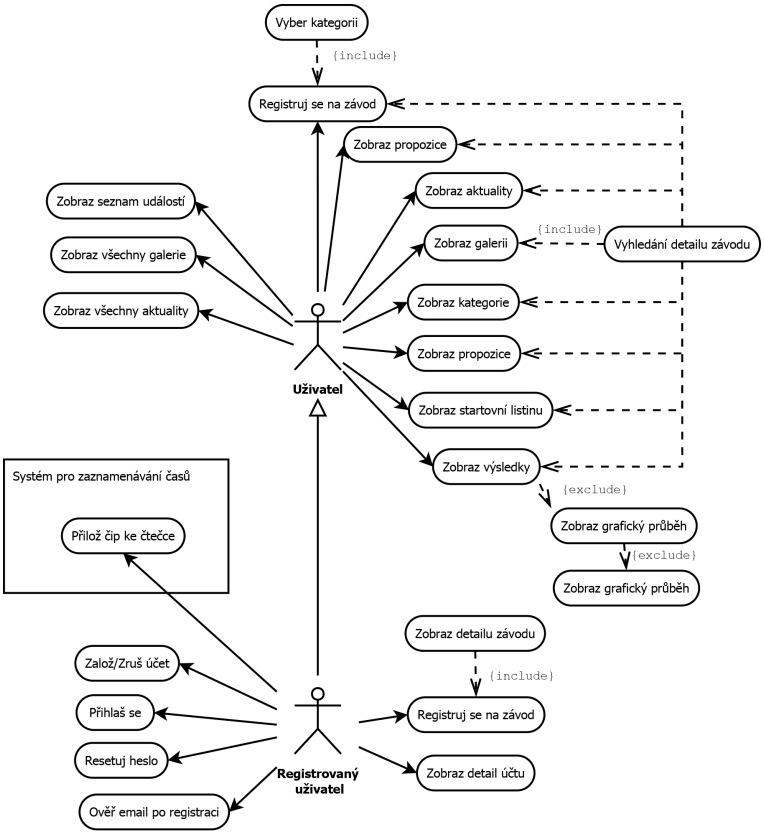


Obrázek 4.5: Diagram případu užití - pověřená osoba

Úspěšným přihlášením do aplikace je uživateli zpřístupněna kompletní funkcionalita. Aplikace provede aktualizaci dat bezprostředně po spuštění, avšak uživatel může kdykoliv provést novou aktualizaci. Se zaznamenaným časem je propojena brána, která musí být vybrána předně. Výsledky jsou při vložení do aplikace automaticky zasílány na server. Jedním z možných scénářů je výpadek internetu, a proto je Pověřené osobě umožněno zaslat výsledky později. V případě potřeby jsou výsledky načteny ze souboru, kam jsou v době závodu automaticky ukládány.

Uživatel a Registrovaný uživatel

Diagram případu užití systému pro činnosti uživatele a registrovaného uživatele [4.6] je zobrazen níže. Akce aktérů jsou vztaženy na prezentační webové rozhraní.

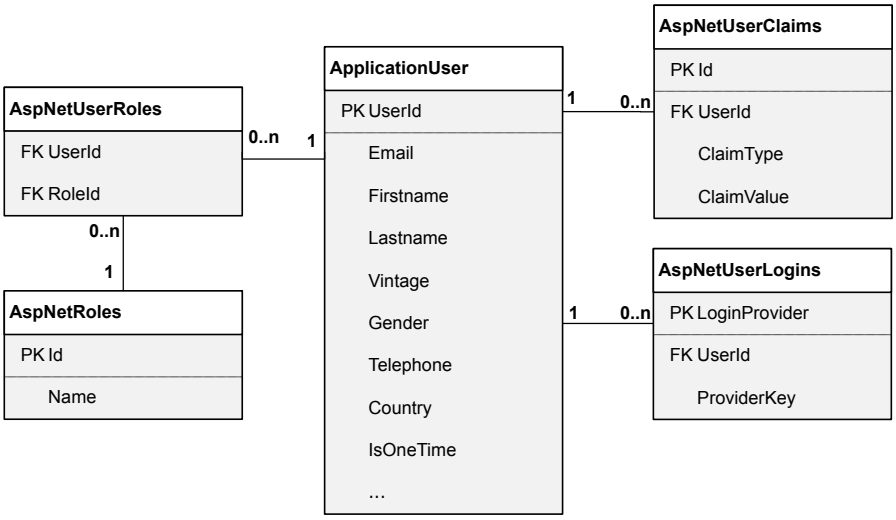


Obrázek 4.6: Diagram případu užití - uživatel a registrovaný uživatel

Uživateli je umožněno procházet téměř celý obsah prezentačního rozhraní. Je-li vybrán závod, jsou zobrazeny pravidla, novinky, propozice kategorie, výsledky a další sekce náležící k danému závodu. Přihlášení a registrace má totožný průběh a operace, jako je tomu u Organizátora v administračním rozhraní. Registrovanému závodníkovi jsou zobrazeny aktuálně dosažené výsledky a veškerá historie závodů, kterých se zúčastnil. V průběhu závodu může Uživatel i Registrovaný uživatel přikládat své čipy k RFID čtečce, která je součástí systému pro zaznamenávání.

4.3 Databázové schéma

Databáze je fundamentálním elementem každého informačního systému pracujícím s daty. Tvorba databázového schématu pro rozsáhlejší informační systémy patří k realizačně náročnějším úkolům. Na základě požadavků byl vytvořen konceptuální návrh ERD (entity relationship) v nástroji pro tvorbu diagramů Microsoft Visio. Poté byly implementovány třídy (entity) v jazyce C#, které Entity framework pomocí objektově relačního mapovaní transformoval na databázové schéma. Původní databázové schéma bylo modifikováno technologií Identity. Tabulka zastupující uživatele, `User`, byla nahrazena tabulkou `ApplicationUser`, která je zobrazena níže [4.7].



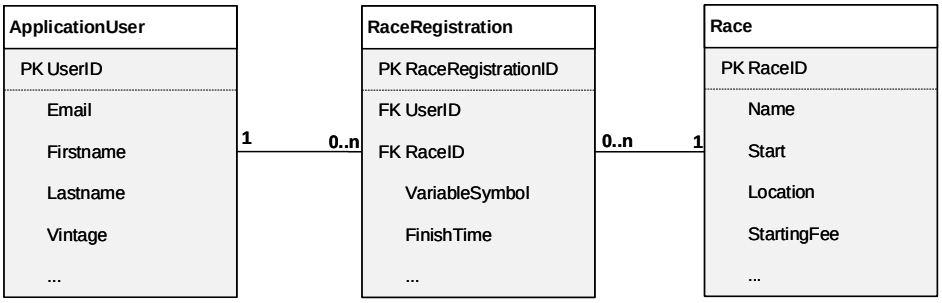
Obrázek 4.7: Databázové schéma ASP.NET Identity uživatele

Asociované tabulky AspNetRoles/Claims/Logins slouží k pokročilé manipulaci s uživatelským účtem jako jsou role nebo přihlašování přes externí autority. S těmito tabulkami systém nepracuje, ale pro správnou funkcionalitu Identity a případné budoucí rozšíření jsou součástí databázového schématu.

Registraci závodníka smí provádět Organizátor závodu, Registrovaný uživatel i běžný Uživatel. Z hlediska aplikační logiky se tyto registrace liší a je potřeba je rozpoznat na úrovni databáze.

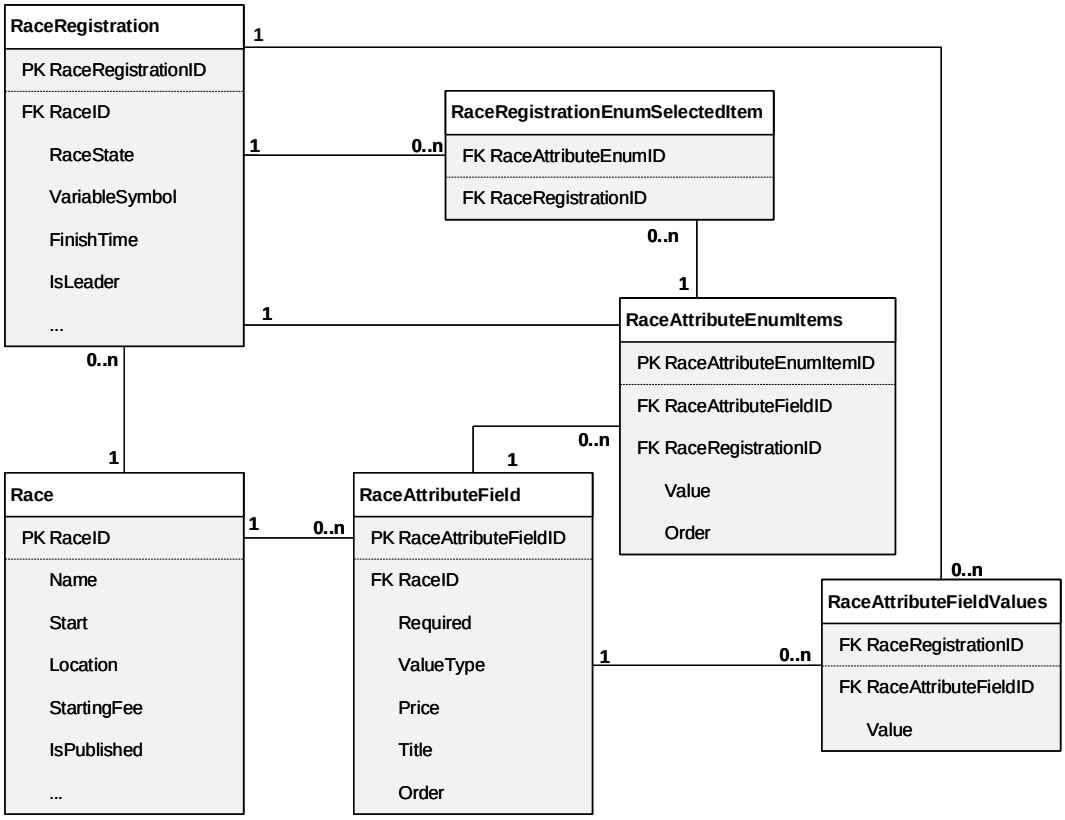
Tabulka Identity uživatele, ApplicationUser, obsahuje uživatelovy osobní informace. Tabulka registrací, RaceRegistration, obsahuje informace pro konkrétního uživatele v konkrétním závodě. Tedy například startovní číslo, variabilní číslo nebo výsledný čas. Registrace obsahuje cizí klíč na uživatele a jejich kardinalita je N:1. Registrace také obsahuje cizí klíč na závod a jejich kardinalita je opět N:1. Výsledkem těchto vztahů je registrace jednoho Identity uživatelského účtu na více závodů bez duplicitních informací v databázi. Uživatelský atribut `IsOneTime` je nastaven na `true`, pokud registraci provádí organizátor nebo je provedena bez přihlášení do systému.

Databázové schéma zobrazující uživatele a registrace je zobrazen níže [4.8].



Obrázek 4.8: Databázové schéma registrace do závodu

Poslední skupinou detailněji popsaných elementů jsou entity pro dynamickou tvorbu formulářových položek. Položka je zastoupena tabulkou `RaceAttributeFields`. Zde je definován datový typ položky, název nebo cena. Vyjma datového typu *možnosti* jsou vstupní hodnoty uživatele uloženy do tabulky `RaceAttributeFieldValues`, jenž je asociována s registrací, `RaceRegistration`. Skládá-li se položka z více možností, pak jsou uloženy v tabulce `RaceAttributeEnumItems`. Konečně zvolené hodnoty možností jsou přístupné z tabulky `RaceRegistrationEnumSelectedItems`. Schéma dynamiky tvořených položek je vloženo níže [4.9].



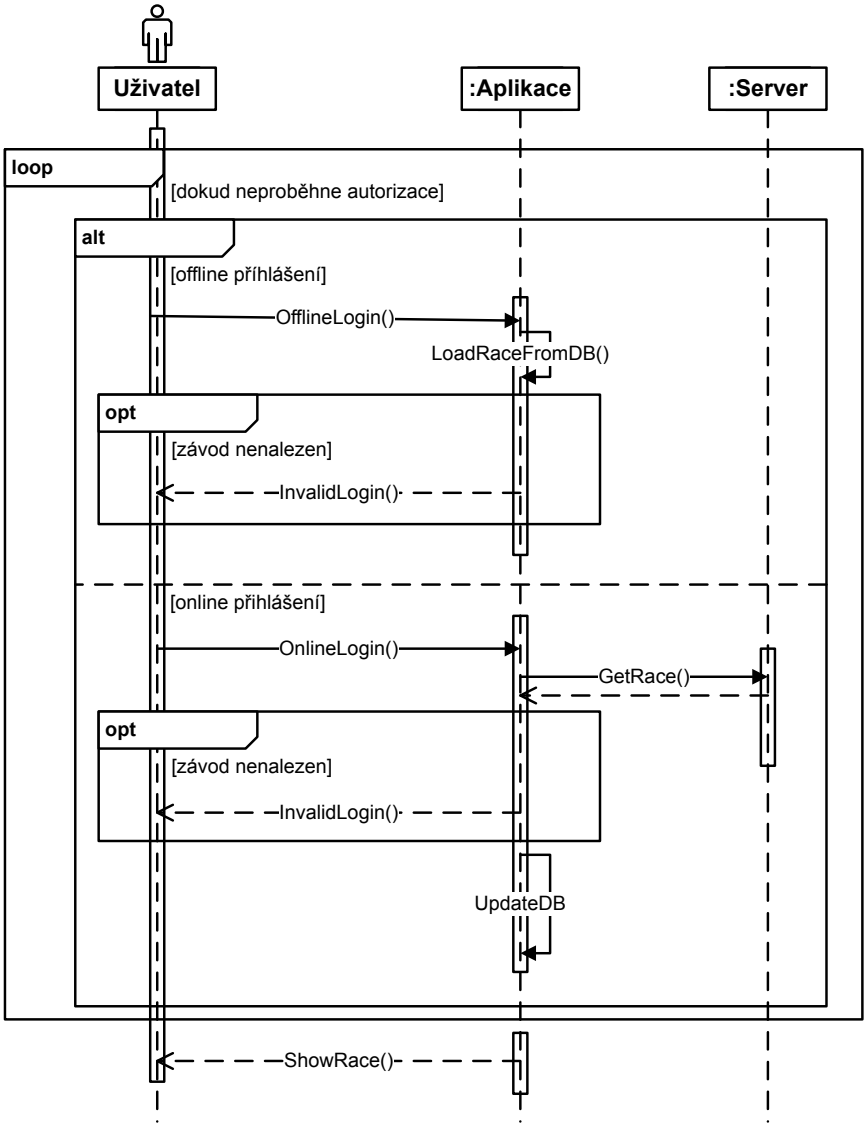
Obrázek 4.9: Databázové schéma dynamicky tvořených položek

4.4 Autorizace klientské aplikace

Klientská aplikace pro snímání časů se autorizuje vůči autorizační části administračního rozhraní, ze kterého byla aplikace stažena. Organizátor závodu nastaví v administrační části jméno a heslo závodu, které později použije Pověřená osoba pro přihlašovací údaje.

Primární způsob autorizace je online ověření. Po úspěšném ověření v autorizační části administračního rozhraní obdrží aplikace kompletní data pro daný závod. V závislosti na pořadí přihlášení jsou data v lokální databázi vložena či aktualizována. V případě prvního přihlášení je do lokální databáze vložen nový závod. V ostatních případech je závod výhradně aktualizován.

Sekundárním způsobem přihlášení je offline ověření. Podmínkou úspěšného přihlášení je dřívější online autorizace, při které došlo ke stažení a úspěšnému uložení závodu spolu s přihlašovacími údaji do lokální databáze. Přihlášením je načten závod a zpřístupněné rozhraní pro snímání závodníků. Z výše uvedeného popisu tedy vyplývá, že aplikaci nelze použít bez minimálně jednoho online ověření. Níže je uveden sekvenční diagram autorizace [4.10].



Obrázek 4.10: Autorizace klientské aplikace

4.5 Snímání RFID čipů

Pro snímání RFID čipů byla zvolena nízkofrekvenční, 125 kHz, RFID čtečka s USB výstupem. Vzdálenost sejmutí čipů integrovanou anténou je dle specifikace 5 až 8 cm [21]. Zařízení je typu *Plug and Play*, přičemž čtečka je napájena přímo z USB kabelu, který je připojen ke zdroji. Zdrojem je v našem případě zařízení, na kterém je možné spustit klientskou aplikaci pro snímání čipů.

Po přiložení čipu je jeho unikátní hodnota přenesena jako ASCII stream 12 bytů, kde poslední dva byty jsou znaky `CR,LF` pro konec řádku na operačním systému Windows. Výstupem je tedy desetimístné dekadické číslo, které operační systém zapíše tam, kde je logická aktivita[1]. Čtečka dočasnou změnou barvy led diodou a krátkým zvukovým pípnutím upozorní na sejmutý čip.

Obrázek 4.11: Fyzická realizace sejmutí čipů

Po kliknutí na zahájení snímání v klientské aplikaci se logická aktivita přesune na textové vstupní pole, do kterého OS zapíše dekadické číslo čipu. Aplikace číslo přečte, zpracuje a setrvá pro další vstup. Současně na sejmutím upozorní hlasitým pípnutím. Výsledky budou pro rychlou kontrolu zobrazeny v aplikaci formou tabulky. S každým zapnutím aplikace a prvním sejmutím čipu dojde v předem určené složce k otevření souboru, na jehož konec se vloží data skládající se z identifikátoru závodu, brány, čipu a časového razítka. Z výše popsaného postupu snímání čipů vyplývá, že pro bezproblémový průběh závodu je vysoce doporučené kontrolovat zaznamenaný čip pověřenou osobou, protože OS může například přesunout logickou aktivitu na nově zobrazené systémové modální okno.

[1] OS se pokusí zapsat číslo do logicky aktivního prvku, například do otevřeného poznámkového bloku

4.6 Grafické rozložení webového a aplikačního rozhraní

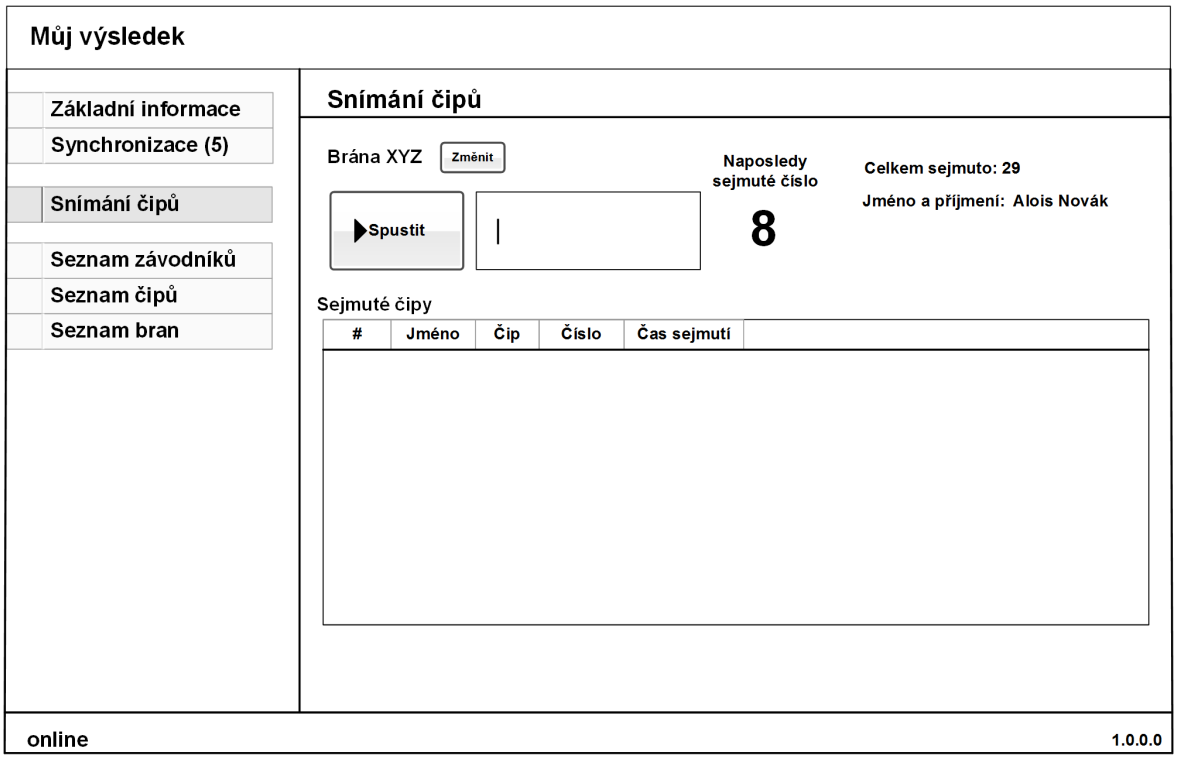
V této sekci jsou uvedena grafická rozložení administračního a desktopového rozhraní. Jsou vybrány demonstrační obrazovky, na kterých lze popsat účel jednotlivých sekcí a v nich se vyskytujících prvků.

4.6.1 Aplikační rozhraní

Základním funkčním elementem je snímání čipů závodníků, na kterém je demonstrováno rozložení a funkcionalita aplikace. Celá aplikace je graficky rozložena do dvousloupcového rozložení. Levý sloupec zobrazuje menu položky, na jejichž základě se přepisuje obsah pravého sloupce. Závodní stanoviště, kde bude aplikace spuštěna, je typicky umístěno na volném prostranství s měnícím se osvětlením. Je tedy vhodné zvolit vysoký kontrast mezi ovládacími prvky.

Po zvolení brány klikne pověřená osoba na tlačítko pro spuštění snímání čipů. Vzhled tlačítka se mění tak, aby byl zřejmý zapnutý a vypnutý režim snímání. Na obrazovku mohou nahlédnout i samotní závodníci, kterým je po sejmutí čipu zobrazeno několik základních informací jako je čas, číslo a jméno závodníka nebo jeho celkové pořadí.

Dalším z důležitých požadavků je informovanost uživatele aplikace o připojení k internetu a s tím spojená synchronizace se serverem. K tomuto účelu slouží především spodní lišta, která mění barvu v závislosti na připojení k internetu (zelená/online, červená/offline). Při offline režimu nejsou sejmuté čipy poslány na server a každý takový výsledek navýší číslo v levém menu u položky „Synchronizace“, kde lze neodeslané výsledky manuálně odeslat. Wireframe popsaného rozložení je uveden níže.

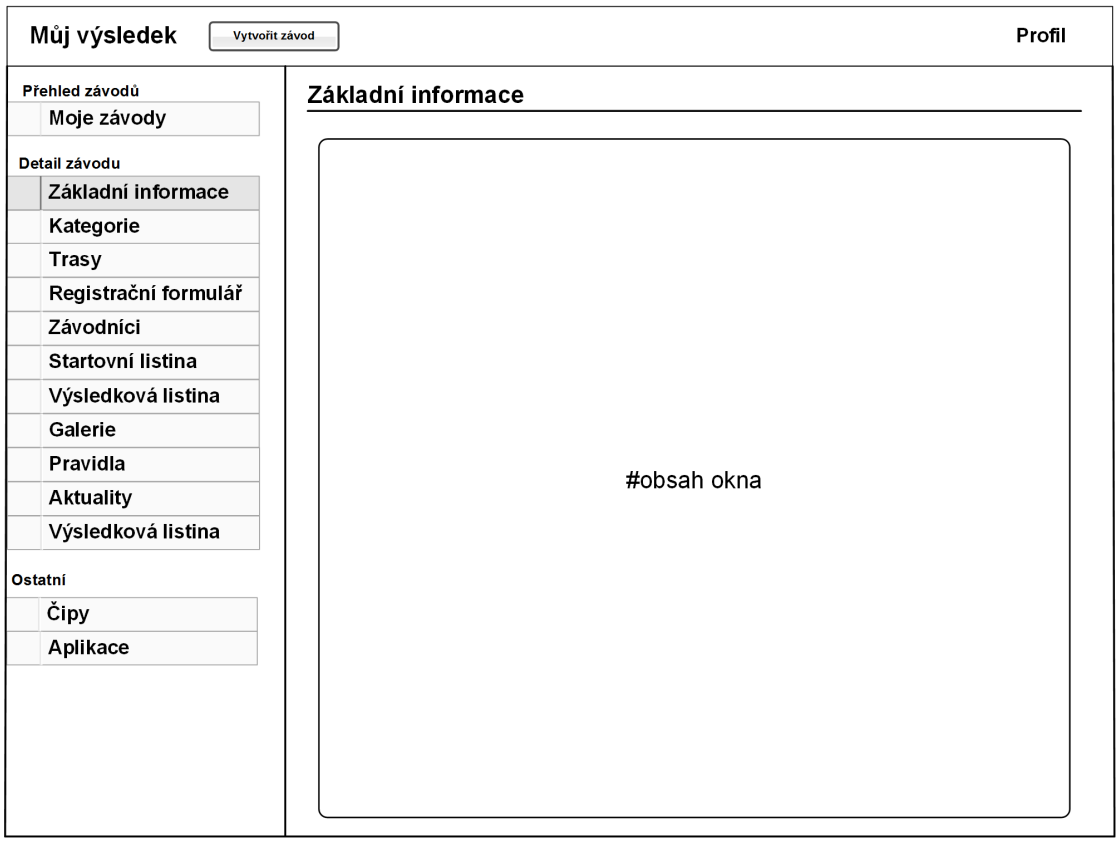


Obrázek 4.12: Uživatelské rozhraní aplikace pro snímání čipů

4.6.2 Webové rozhraní

Administrační rozhraní je určeno pro organizátory, kteří zde daný závod vytvoří a spravují. Jedná se o jádro celého systému. Vzhledem k jeho důležitosti je kladen vyšší důraz na funkcionalitu na úkor grafického zpracování.

Byla provedena kolekce několika šablon, určených pro administraci informačních systémů, a následná selekce nejvhodnější. Pro svou kompatibilitu, zpracování, rozšířenost a podporu byla vybrána open-source šablona Admin LTE [2]. Základní verze levé strany šablony obsahuje položky relevantní ke všem závodům (moje závody, čipy, aplikace, ...). Vytvořením či přesměrováním na detail závodu dojde k rozšíření levého menu pro správu závodu (kategorie, trasy, závodníci, ...). Horní lišta zobrazuje přímé odkazy na často konané uživatelovy akce a přístup ke svému účtu. Wireframe popsaného rozložení je uveden níže.



Obrázek 4.13: Grafické rozložení administračního rozhraní

[2] AdminLTE - https://almsaeedstudio.com/

Kapitola 5

Implementace navrženého systému

Obsahem této kapitoly jsou netriviální implementační témata navrženého systému. Čtenář je nejprve seznámen s implementací informačního systému, serveru, a poté s význačnějšími realizačními a implementačními sekcemi desktopové aplikace. Závěrem kapitoly je uveden komunikační protokol mezi klientem a serverem. Implementační témata byla zvolena na základě své komplexnosti a časové náročnosti.

5.1 Server

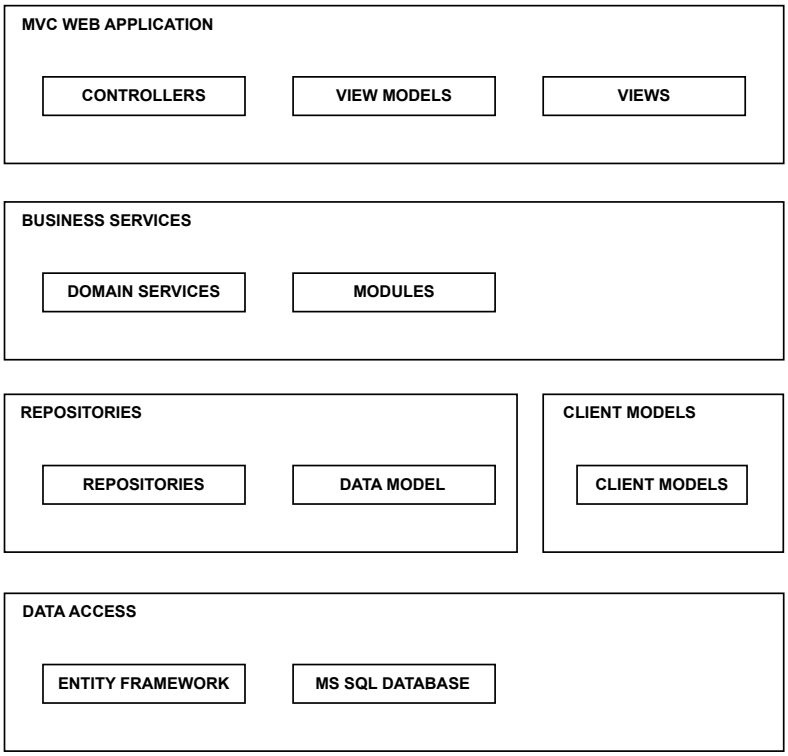
Server zastupuje informační systém, který je složený z administračního a prezentačního rozhraní. Vyjma podkapitoly *Porovnání výsledků* jsou zmíněná architektonická a implementační témata uplatněná v obou rozhraních. Porovnání výsledků je zpřístupněno pouze v prezentačním rozhraní.

5.1.1 Struktura serveru

Implementace serverové části je postavena na vícevrstvé architektuře. Vytvoření vícevrstvé koherentní architektury, která splňuje základní požadavky, jako je například udržovatelnost, testovatelnost, bezpečnost, je komplexní problematika vzájemných vztahů, zodpovědností a logického rozdělení tříd a projektů. Následující popis je strukturován od fyzického úložiště po části s vysokou abstrakcí. Dle referenčního řešení [8] bylo sestaveno základní rozložení logických elementů architektury, které bylo přizpůsobeno pro řešený systém. Uplatněné schéma [5.1] je zobrazeno níže.

Datová vrstva

Zodpovědností této vrstvy je manipulace s daty formou komunikace s databází a zpřístupnění dat vyšším vrstvám architektury. Do datové vrstvy můžeme zařadit vrstvy Repositories, Data Access a Client models ze schématického obrázku [5.1]. Vrstva Data Access je všeobecně označována zkratkou DAL (Data Access Layer). Tato vrstva zprostředkovává rozhraní pro CRUD operace (Create, Read, Update, Delete). Jádrem je ORM (Object Relation Mapping) Entity Framework, viz kapitola [3.4], který provede namapování relačních dat z databáze na doménové objekty. Veškerá komunikace, včetně dotazů, je v rámci této práce přenechána na Entity framework. Jak je popsáno v návrhu systémů, je využit Code first přístup. Doménové objekty, entity, byly nejprve definovány v rámci doménového modelu a na základě anotací a sémantiky bylo pomocí EF vytvořeno databázové



Obrázek 5.1: Vícevrstvá architektura serveru

schéma. Typickým řešením při manipulaci dat mezi Entity framework a vyššími vrstvami architektury je uplatnění návrhového vzoru Repository. Návrhový vzor Repository je základním kamenem doménou řízeného návrhu. Pro každou entitu je vytvořen repositář, který definuje CRUD operace a manipuluje s daty. Repositář můžeme chápat jako obal entity, který nám umožňuje manipulaci s jejími daty. S využitím generických vlastností technologie ASP. NET byl vytvořen generický repozitář Generic Repository a další návrhový vzor Unit of Work, v rámci kterého je například sledována entita (z hlediska databázového kontextu), spravovány transakce nebo uplatňovány změny dat. Implementace Unit of Work a Generic Repository vychází ze článku od autora T. Dysktra [4]. Samostatnou částí datového modelu jsou třídy pro komunikaci mezi klientem a serverem. Třídy jsou implementovány v samostatném projektu administračního rozhraní, který je poté referencován jako knihovna DLL do desktopového klienta. Server i klient tedy sdílí stejné třídy, které jsou serializovány v serializačním jazyce JSON na serveru a deserializovány na straně klienta.

Logická vrstva

Logická vrstva, jenž je uvedena na obrázku [5.1] jako Business Services, obsahuje hlavní logiku realizovaných systémů. Některé entity fungují na jednoduchém principu CRUD operací a využívají čistě metody repositářů. Pak jsou zde entity, které jsou komplikovanější (složitější zachování databázové konzistence, integritních omezení, logické omezení) a je vhodné pro ně vytvořit samostatný repositář. V logické vrstvě je pro každou entitu vytvořena její třída Service. Tato třída obsahuje instanci třídy Unit of Work, aplikační logiku a další

metody vztahující se k asociované entitě. K volbě tohoto řešení vedlo několik důvodů. Hlavním důvodem je samotný princip vícevrstvé architektury, kdy je webová aplikace odlehčena od logiky. Mezi další důvody patří unifikovaný přístup a implementace specifických požadavků pro manipulaci s daty, což je například kontrola logických omezení, validace z hlediska integrity dat nebo vytváření databázových transakcí. Typickým požadavkem vyšší vrstvy architektury, tedy MVC aplikace, je získání dat z několika entit (tabulek). Pro jednodušší práci se servisními třídami byl implementován návrhový vzor Factory, který zpřístupňuje všechny entitní servisy. Entitní servisy jsou udržovány pomocí Factory na principu dalšího návrhového vzoru Singleton, a to pouze v rámci jednoho dotazu z vyšší vrstvy architektury. Důvod tohoto řešení je popsán níže v poslední vrstvě realizované architektury.

Vrstva webové aplikace

Poslední vrstvou vícevrstvé architektury je samotná MVC webová aplikace, která má přímou interakci s uživatelem. V řadiči se v rámci jednoho HTTP dotazu vytvoří instance Factory, která na základě požadavků vytvoří vždy jen jednu instanci několika různých entitních tříd.

Řadič tak jen přijme dotaz od uživatele, provede autorizaci uživatele, validaci dat, modifikuje dočasná data a předá řízení nižším vrstvám architektury. Výsledek nižších vrstev (například úspěšné uložení do databáze) je předán řadiči, který jej zobrazí uživateli. Mezi význačnější použité nástroje pro webovou aplikaci patří mimo technologie ASP. NET MVC také vizuální a logické komponenty společnosti Devexpress (například tabulka s výsledky) a technologie Bootstrap, JQuery nebo Morris.

5.1.2 Datový model pohledu

Pro každý pohled, jenž manipuluje nebo zobrazuje perzistentní data, je definován datový model. Ve schématu [5.1] je zastoupen jako viewmodels. Řadič zkopíruje data do datového modelu pohledu. Pohled je poté poslán do prohlížeče uživatele. Definice datového modelu pohledu není povinná a je možné uvést datovým modelem přímo entitní třídu. Tato technika je přímočařejší a není nutné kopírovat data z viewmodels do entitních tříd, jejichž data jsou ukládána do databáze. Uvedením entitních tříd jako datového modelu pohledu vzniká několik úskalí. Mezi přední patří nemožnost definice pouze dočasné proměnné, která je specifická pouze pro daný pohled. Dalším je komplikovanější validace, která je demonstrována na následujícím příkladu:

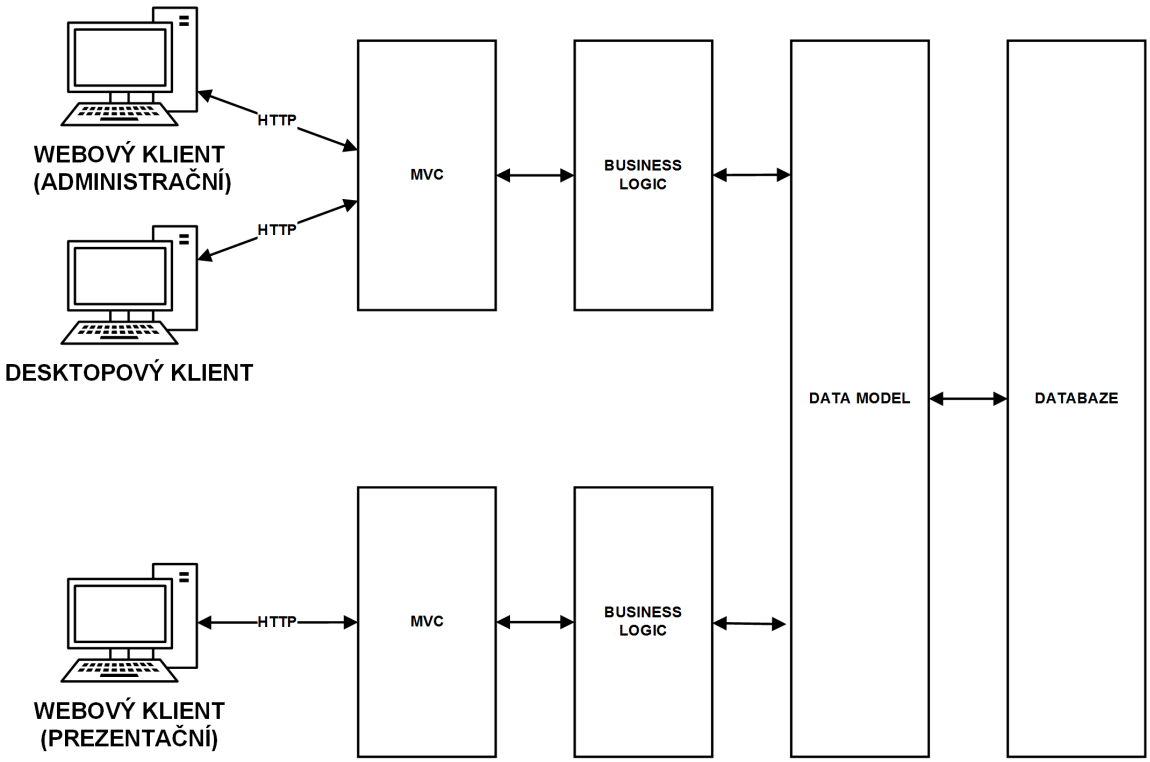
Mějme entitní třídu Uživatel obsahující atributy jméno, příjmení a heslo. Atributy jsou povinné a jsou nad nimi definovány příslušné anotace. ASP .NET validuje v řadiči celý datový model. Pohled zobrazující registraci uživatele bude v pořádku validován, neboť jsou ve formuláři použity a vyplněny všechny tři atributy (jméno, příjmení a heslo). Pohled pro aktualizaci uživatele použije pouze atributy jméno a příjmení. Uživatel korektně vyplní aktualizovaná data a odešle na server. Řadič ovšem vyhodnotí datový model jako nevalidní, protože atribut heslo není vyplněn (ačkoliv jsme ho vůbec nepoužili). Museli bychom vytvořit vlastní validační logiku, která by validovala pouze atributy použité v pohledu.

V řešeném informačním systému jsou využity oba přístupy k datovým modelům. Pokud nejsou s pohledem spjaty manipulační požadavky, ale slouží pouze k zobrazení dat, je

použit první přístup. Tedy entitní třída je datovým modelem pohledu. V opačném případě je vytvořen vlastní datový model pro každý pohled.

5.1.3 Sdílení vrstev mezi dvěma systémy

Mezi netriviální implementační řešení můžeme zařadit sdílení vrstev architektury mezi dva informační systémy. Jedním je administrační a druhým je prezentační systém, viz schéma [4.1]. V rámci každého systému byla vytvořena samostatná MVC webová aplikace spolu s aplikační logikou. V kontextu výše popsané vícevrstvé architektury byla pro každý systém vytvořena vrstva webové aplikace a logická vrstva. Datová vrstva je pak sdílena. Implementována je v administračním projektu samostatnou class library. Vzniklá DLL knihovna je přidána jako reference do prezentačního rozhraní. Původní koncept zahrnoval pouze jednu vrstvu webové aplikace a aplikační logiky. Zodpovědnost za rozdělení administračního a prezentačního rozhraní by nesly řadiče. Během implementace administračního rozhraní, které bylo realizováno jako první, vyplynulo, že koncept jednoho systému není vhodný. Hlavním důvodem byla komplikovanost a udržovatelnost kódu. Vznikly tedy dva systémy se sdílenými vrstvami. Negativním aspektem je místy duplicitní, avšak mnohem lépe udržovatelný kód. Mezi nesporné výhody tohoto řešení patří izolovanost obou řešení, kdy vizuální či logická změna jednoho systému neovlivní systém druhý. Popsané řešení je zobrazeno na schématu [5.2] níže.



Obrázek 5.2: Rozložení vrstev napříč řešenými částmi systému

Celá oblast správného rozložení vrstev, sdílení a zodpovědnosti je poměrně komplikovaná a vyžaduje delší časovou křivku učení a správného pochopení. Některé prvky se mohou

časem ukázat jako chybné, avšak v rámci časového rozmezí diplomové práce bylo vyvinuto zvýšené úsilí na prostudování principů a vzorů architektur informačních systému.

5.1.4 Výkonnost Entity framework

Práce s databází je úzce spjata s potencionálními výkonnostními výkyvy v závislosti na struktuře dotazu a zpracování dat.

Jako způsob sestavení a přeposlání SQL dotazu do databáze je používán mechanismus `DbSet.SqlQuery` [3]. `DbSet` reprezentuje kolekci všech entit, které se vyskytují v kontextu. Metoda `SqlQuery` vrací entitní objekty, jenž jsou sledovány, anglicky tracking, v rámci databázového kontextu. To znamená, že je pozorováno, zda jsou objekty v dočasné paměti stejné s těmi v databázi. Objekty v paměti fungují jako *cache* a jsou použity při aktualizaci entit.

Například pro výpis závodů v prezentačním rozhraní není nutné zaznamenávat změny entit, protože závody slouží pouze pro čtení. Přidáním metody `AsNoTracking` do databázového dotazu docílíme vypnutí sledování entit. Jedním z benefitů je navýšení výkonnosti během zpracovávání většího množství dat. Druhou výhodou je manipulace s dvěma stejnými objekty v jednom databázovém kontextu. Není možná aktualizovat entitu, pokud jsou v paměti dvě entity se stejnými primárními klíči. Popisovaný konflikt nastal aktualizací registrovaného závodníka, kde je z validačních důvodů nutné načíst dva stejné záznamy, které EF převede na dva různé objekty. Vložením metody `AsNoTracking` do dotazu pro validační objekt byl konflikt vyřešen.

Dalším z uplatněné výkonnostní optimalizace je použití typu `IQueryable` na objekty načítané z databáze. Dosažení podmnožiny záznamů typicky docílíme přidáním filtrační metody `Where` technologie LINQ. Je-li použit typ `IEnumerable`, jsou nejprve načteny všechny záznamy do dočasné paměti. V dočasné paměti jsou uplatněny filtry a teprve poté jsou záznamy vráceny do aplikace, což může být neefektivní. Při použití `IQueryable` je filtr nejprve převeden do klauzule WHERE a poté je odeslán do databáze.

5.1.5 Integrace ASP.NET Identity

Dokumentace firmy Microsoft [5] je vždy popisována vytvořením nového projektu s autentizací Individual User Accounts a Code first přístupem. Code first přístup je vysvětlen výše v sekci [3.4]. Volba Individual User Account automaticky vytvoří prostředí pro ukládání uživatelských profilů do SQL databáze pomocí již existujících nebo vytvořených autorit (Google, Microsoft, Facebook, ...). Vytvořením projektu získáváme předdefinovaný kód pro autentizaci i autorizaci. Informace o uživatelích jsou spravovány frameworkem Identity, který technologií Entity framework operuje nad nově vytvořenou databází. Databáze je automaticky vytvořena (default connection) při prvním spuštění projektu. Podrobnější analýzou zjistíme, že byla vytvořena třída `ApplicationUser`, která dědí z Identity User, a třída reprezentující databázový kontext, která dědí z kontextu Identity.

Komplikace nastávají, pokud již máme vytvořený projekt bez frameworku ASP.NET Identity. Úspěšná integrace frameworku do existujícího projektu a existující databáze vyžaduje pokročilejší znalost této problematiky. Uvažujeme-li projekt s Code First přístupem a existující databází včetně tabulky uživatelů, pak je nejsnadnější nechat Identity vygenerovat nové tabulky a do nich převést existující informace o uživatelích. Nejprve provedeme export existující tabulky uživatelů. Poté ve třídě `ApplicationUser` (vygenerovanou Identity) přidáme proměnné z naší existující třídy a provedeme aktualizaci databáze. Nakonec provedeme import exportovaných dat do nové tabulky, která byla vytvořena na zá-

kladě Identity při aktualizování databáze dle datového modelu. Tento postup se nicméně nepodařilo uplatnit v řešeném systému. Vzhledem k počáteční fázi vývoje byl vytvořen nový projekt typu Individual User Accounts a do něj byla vložena stávající implementace. Aplikační kontext nového projektu bylo ovšem nutné propojit s existujícím kontextem v datovém modelu. Možné řešení nabízí dědičnost, kdy existující kontext dědí z Identity kontextu. Díky dědičnosti jsou v existující databázi vytvořeny tabulky pro ukládání informací o uživatelích. Další významný problém může nastat, pokud potřebujeme explicitně nadefinovat některé vlastnosti v metodě `OnModelCreating`, podle které se řídí Entity framework při vytváření nebo modifikaci databáze. V těle funkce musíme zavolat i bázovou metodu `OnModelCreating`, jenž je definována Identity a jejíž aplikací budou při aktualizaci databáze vytvořeny potřebné tabulky pro informace o uživatelích. Implementace zmíněného problému je uvedena níže.

```
public class DbContext: IdentityDbContext<ApplicationUser>
{
    public override void OnModelCreting (DbModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
    }
}
```

Uplatněné řešení není jediné a s vysokou pravděpodobností existuje řešení optimálnější, kdy máme například dva kontexty. Jeden je aplikační, který spravuje operace nad uživatelskými profily a druhý je datový, který spravuje zbylý datový model.

5.1.6 Dynamická validace

Položky registračního formuláře jsou dynamicky vytvářeny organizátorem . U každé položky lze definovat datový typ a povinnost vyplnění. Validační omezení je definováno anotacemi nad deklarací proměnné. Uložení dat do databáze typicky předchází tři validační kontroly. První je pomocí knihovny `JQuery` v prohlížeči klienta. Formulář není odeslán, dokud nejsou všechny položky v pořádku. Druhá validace se provede v řadiči serveru. ASP .Net zkontroluje příchozí data dle anotací a nastaví *boolean* proměnnou `IsValid` třídy `ModelState`. V řadiči jsou také kontrolována složitější omezení. Příkladem je unikátnost registračního emailu. Poslední validace je opět povedena dle anotací Entity frameworkem před uložením dat do databáze.

Validační anotace není vhodné definovat nad proměnnou zastupující dynamickou proměnnou ve formuláři. ASP.Net i EF mohou vyhodnotit položku jako nevalidní, ačkoliv je v pořádku (nepovinná).

Uplatněné řešení deklaruje na datové vrstvě pomocnou proměnnou. S touto proměnnou jsou spjaty anotace `required` a `notmapped`, což zajistí povinné vyplnění položky, avšak proměnná je frameworkem EF ignorována.

Ukázkový příklad demonstruje průchod nepovinného, dynamicky vytvořeného, textového vstupu. Třída `RaceAttributeField` obsahuje proměnnou `Required` určující povinnost vyplnění. Implicitně je každá položka anotována jako povinná. Validační omezení jsou generována dle proměnné `Required` a přeposlána na webový prohlížeč. Fragment popsaného kódu je uveden níže.

```
if (RaceAttributeField.Required)
{
    Html.TextboxFor(m => m.DynamicInput)
    Html.ValidationMessageFor(m => m.DynamicInput)
}
else
{
    Html.EnableClientValidation(false);
    Html.ValidationMessageFor(m => m.DynamicInput)
    Html.EnableClientValidation(true);
}
```

Nevyplněná nepovinná textová položka je v řadiči vyhodnocena jako nevalidní (anotace `required`). Před kontrolou validity je tedy nutná redukce validačních chyb. Třída `ModelState` obsahuje kolekci `Keys`, kde jsou uloženy názvy příchozích proměnných, a kolekci `Values`, kde jsou uvedeny jejich validační problémy. Redukcí validačních chyb se míní iterace proměnných v kolekci a odstranění chyb u nepovinných položek. Výsledkem je zachování validace pro povinné položky a odstranění validace pro nepovinné položky.

Korektně validovaná data jsou v řadiči nakopírována do datových proměnných, které jsou mapovány Entity frameworkem. Jak již bylo zmíněno, proměnné datového modelu pro dynamické položky formuláře nejsou anotovány validací a nedochází tak k validační kontrole EF před uložením do databáze.

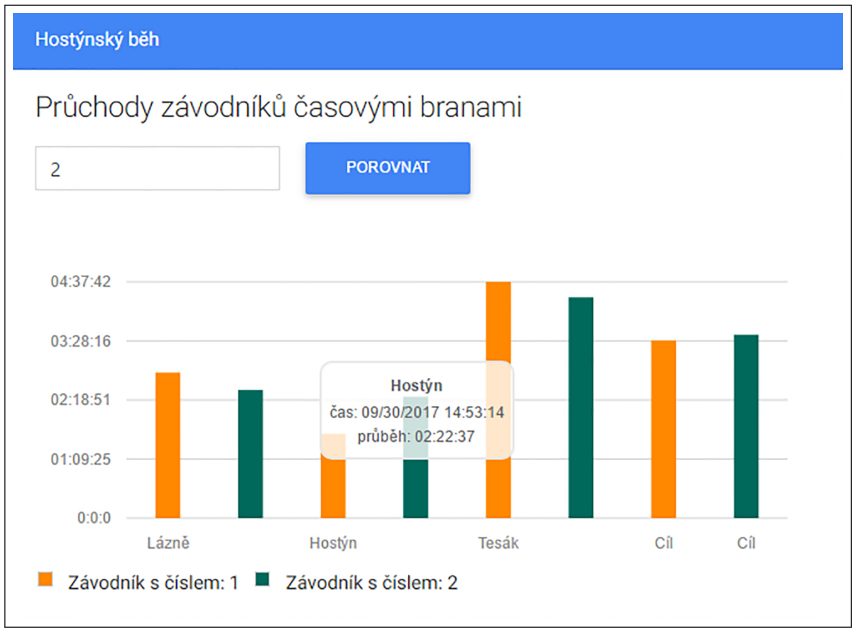
Jiný způsob dynamické validace je implementován u textového vstupu, který musí být unikátní v rámci celé databáze. Takovým vstupem je například registrační email nového účtu. Unikátnost nejsme schopni detekovat na straně klienta. ASP .Net poskytuje mechanismy pro vzdálenou validaci konkrétního textového pole [13]. Nad proměnnou v datovém modelu, namapovanou ve formuláři, se definuje anotace:

```
[Remote("IsEmailAvaiable", "Account")]
public string Email { get; set; }
```

První parametr je název metody řadiče. Druhý určuje řadič. Vyplněním textového pole ve formuláři dojde k ajaxovému odeslání hodnoty na řadič. Asynchronní metoda `IsEmailAvaiable` vyhodnotí unikátnost emailu a serializovanou odpověď odešle zpět na klienta.

5.1.7 Porovnání výsledků

V prezentačním rozhraní lze zobrazit graf vykreslující časovou prodlevu mezi jednotlivými branami. Pro práci s grafy byla zvolena knihovna Morris.js [1]. Data jsou nejprve zpracována v řadiči, kde probíhá selekce a následná serializace serializačním jazykem JSON. Serializovaná data jsou předána klientské funkci pro vykreslení grafu `Morris.Bar()`. Textové pole nad grafem umožňuje vložit číslo závodníka, jehož výsledky mají být porovnány. Potvrzením formuláře dochází k ajaxovému odeslání a poté k aktualizaci grafu s nově příchozími daty. Grafická podoba [5.3] porovnání časových prodlev mezi závodníky je vložena níže.



Obrázek 5.3: Grafická reprezentace časových prodlev mezi branami

5.2 Klient

Klient zastupuje desktopovou aplikaci pro snímání RFID čipů. Aplikace je vyvíjena na operační systém Windows technologií WPF.

5.2.1 Aktualizace klienta

Aktualizací klientské aplikace je míněno vydání nové verze programu. Existuje několik způsobů publikace nové verze aplikace. Nejjednodušší variantou je manuální zkopírování adresáře, do kterého byl publikován projekt, na cílový počítač. Adresář obsahuje potřebné knihovny pro spuštění. Aplikace je spuštěna bez nutnosti instalace. Hlavní nevýhodnou je nekomfortní aktualizace aplikace.

Další dva způsoby nabízejí možnost online publikace a aktualizace [10]:

- ClickOnce - přímočaré řešení publikace jednodušších aplikací. Aplikace je spuštěna v prostředí sandbox s nastavenými právy, což přispívá k vyšší kompatibilitě a zabezpečení. Existují tři způsoby zveřejnění: z webové stránky, sdíleného internetového

[1] http://morrisjs.github.io/morris.js/index.html

úložiště nebo přenosného média (cd-rom). ClickOnce aplikace podporuje automatickou aktualizaci. Po spuštění se aplikace pokusí kontaktovat server a případnou reinstalací aktualizuje pouze změněné soubory.

- Windows Installer - komplexnější řešení publikace typicky složitějších aplikací. Windows Installer nabízí podrobnější nastavení instalace. Oproti ClickOnce řešení je možné nakonfigurovat potřebné ovladače, sdílené soubory, práva uživatelů, atd. Nevýhodou může být dodatečná implementace aktualizace.

Pro naši práci bylo vybráno ClickOnce řešení, a to pro svou jednoduchost a vestavěné aktualizaci aplikace. Jako způsob zveřejnění bylo vybráno sdílené internetové úložiště, odkud si organizátoři v administračním prostředí stáhnou klientkou aplikaci. Nevýhodou může být instalace do specifických adresářů (ClickOnce application cache) definovaných OS Windows, protože uživatelé jsou zvyklí hledat soubory nainstalovaných programů v adresáři Program files.

5.2.2 Struktura klientské části aplikace

Klientská desktopová aplikace je stejně jako server rozdělena do několika vrstev rozdělených dle funkcionality a zodpovědnosti. Fundamentální rozdělení je na základě návrhového vzoru MVVM pro WPF aplikace, viz kapitola [3.5.1]. Nejnižší vrstva reprezentuje vrstvu datovou, kde je implementován datový model. Dále je zde vrstva Services pro manipulaci s daty. Zde je implementováno vytvoření a správa databáze, deserializace dat či čtení/zápis ze/do souboru. Pro deserializaci dat je stejně jako na serveru využita knihovna Newtonsoft [2]. Databáze je realizována pomocí knihovny SQLite, viz kapitola [3.6]. Další vrstvou je ViewModel, což je obdoba řadiče v návrhnu MVC. Ve vrstvě ViewModel je obsažena logika aplikace, definice Commands a navigace mezi okny. Základním prvkem ViewModel vrstvy je třída ViewModelFactory, která je navrhnuta podle návrhového vzoru Factory. Každá třída ViewModel zastupuje jednu stránku ve View vrstvě. Tyto třídy jsou v rámci celé aplikace vytvořeny jako Singletons a zpřístupněny právě přes ViewModelFactory. Poslední vrstva View zprostředkovává grafické rozhraní pro uživatele a validaci vstupních dat. Je tvořena logicky propojenými UserControls, které využívají technologie data binding pro namapování dat a propagování změn ve vrstvě ViewModel. Vyjma vlastních stylů byla použita knihovna Material Design [3] obsahující styly (vzhled) základních komponent WPF a definici vlastních komponent jako je například Card, Layout či ColorPicker.

5.2.3 WPF datagrid

WPF datagrid je vestavěná komponenta WPF. Zastupuje strukturované zobrazení dat formou tabulky.

Virtualizované zobrazení

Jedním z obecných požadavků na aktuální aplikace je přizpůsobení zobrazení na obrazovkách různých velikostí. U webově založených produktů je responzivní či adaptivní zobrazení téměř standardem. Zcela rozdílná situace je u WPF desktopových aplikací, kde je pro renderování vzhledu použit značkovací jazyk XAML. Existuje zde komponenta `Wrappanel`, která

[2] http://www.newtonsoft.com/json
[3] http://materialdesigninxaml.net/

v případě nedostatku zobrazovací plochy umístí horizontálně umístěné prvky vertikálně. V praxi se ale využívají nastavené limity výšky a šířky jednotlivých komponent v kombinaci s posuvníky, anglicky scrollbars, scrollviewers. Příkladem mohou být produkty balíku MS Office od firmy Microsoft.

V počátcích implementace byl obsah aplikace vložen do horizontálního a vertikálního posuvníku, které se zobrazovaly při zmenšení okna na minimální výšku či šířku. Výkonnostní problémy nastaly při vykreslení většího počtu položek v komponentě `datagrid`. WPF datagrid podporuje virtualizované zobrazení [11]. Každý řádek komponenty datagrid je vykreslen až tehdy, kdy je to vyžadováno. Řádky mimo zobrazovací plochu nejsou instanciovány. Popsané chování nefunguje, je-li datagrid umístěn do vertikálního posuvníku. V takovém případě je vyrendrován celý obsah datagridu, protože celý datagrid je virtuálně viditelný. Následkem je vysoká odezva hraničící s mrznutím aplikace. Řešením je vyjmutí vertikálního posuvníku a nastavení posuvníku uvnitř datagridu. Datagrid poté vyplní dostupnou zobrazovací plochu, což je plocha, kterou má k dispozici uživatel. Zbylé položky jsou rendrovány při posunutí vnitřního posuvníku datagrid.

Za zmínku také stojí nativní podpora virtualizovaného vykreslování pouze v posuvném módu item-by-item. Pohybem posuvníku jsou vykreslovány celé řádky datagridu. To je změna oproti režimu pixel-by-pixel, na který jsou uživatelé zvyklí například z webového zobrazení.

Asynchronní binding

Implicitně je při použití technologie data binding nastaven synchronní binding. Každá komponenta, během načítání celého okna s nastaveným data kontextem, načítá svůj datový obsah a nedovolí uživateli jakkoliv interagovat v rámci celého okna. Typicky je tato činnost vykonána velmi rychle a prodleva je neznatelná. Citelné omezení interakce s uživatelem vzniká, pokud se jedná o příliš rozsáhlá data nebo je limitována jejich dostupnost. Například data z webových služeb či databáze. V takovém případě je vhodné definovat parametr `IsAsync=true`. S asynchronním typem binding jsou data načítána v separátním vlákně, čímž je uživateli umožněna interakce se zbylými komponentami zobrazované plochy.

Popsané postupy a technologie, virtualizované zobrazení a asynchronní binding, byly uplatněny v desktopové aplikaci pro snímání čipů.

5.2.4 Zanesení výsledku

RFID čtečka implementuje rozhraní HID (Human Interface Device)[4] a je operačním systémem detekována jako externí klávesnice. Čtečka po přiložení čipu odešle EPC kód následovaný hexadecimální hodnotou `0x0D` (Enter).

Operační systém zapisuje číslo čipu do textového pole v aplikaci, které při každém stisknutí klávesy vykoná kód definovaný událostí `Key_Down`. Detekce klávesy Enter vyvolá asynchronní metodu `AddScannedChip` pro uložení a zaslání sejmutého čipu.

Reprezentace výsledného čísla čipu se liší podle jazykového rozložení klávesnice. Reverzním postupem bylo zjištěno, že například pro číslici nula systém zaznamená zmáčknutí klávesnice s označením `D0`, ASCII 4868. Výstupem systému s anglickým rozložením klávesnice je očekávaná nula, avšak u českého rozložení je zobrazeno písmeno *é*. Operační systém tedy vypíše na výstup vždy virtuální reprezentaci klávesy `D0` dle jazykového rozložení. Řešením je implementace převodu přijatých symbolů bez závislosti na stavu jazyko-

[4] http://www.usb.org/developers/hidpage/HID1_11.pdf

vého rozložení klávesnice. Číslice nula je u českého rozložení zobrazena jako písmeno *é*. Na pozadí je ale detekována hodnota stisknuté klávesy a písmeno *é* je přepsáno na číslici nula. Tento proces je velmi rychlý a uživatelem nepozorovatelný.

Nevyřešeným problémem zůstalo paralelní sejmutí čipů ze dvou a více čteček. Aplikace je schopna korektně zpracovat pouze jeden sejmutý čip v jednom časovém okamžiku. Jsou-li čipy současně přiloženy ke dvěma čtečkám zapojeným do jednoho počítače, pak jsou vyvolány události `Key_Down` a aplikace není schopna determinovat jejich zdroj. Obě čtečky se tedy ve vstupním textovém poli interferují. Možným řešením je zpracování vstupu pomocí knihovny LibUsbDotNet [5], která je schopna rozpoznat USB port, po kterém se přenáší sejmuté číslo čipu. Problematika bude řešena v rámci budoucího rozšíření.

5.3 Komunikace mezi klientem a serverem

Sekce popisuje komunikaci a obsah přenesených dat mezi klientem, aplikací pro snímání čipů a serverem. Pro úplnost je vybrán scénář s prvotním přihlášením a odesláním zaznamenaného výsledku.

Potvrzením přihlašovacích údajů se provede hash hesla a společně s přihlašovacím jménem dojde k odeslání dat na server. Pro odeslání je zvolena metoda GET protokolu HTTP. Server, naslouchající na řadiči endpoint, přijme zprávu a na základě jejího obsahu provede autorizaci. Je-li autorizace úspěšná, jsou data pro aplikaci serializována serializačním jazykem JSON a odeslána zpět aplikaci. Výsledek neúspěšné autorizace je prázdná zpráva s nastaveným HTTTP statusem 401 (unautorized). Struktura dotazu je uvedena níže:

```
https:.../Endpoint/GetData?code=$1&password=$2
```

- $1 - přihlašovací jméno
- $2 - přihlašovací heslo

Po úspěšném přihlášení dojde v aplikaci k uložení přijatých dat do databáze. Režie uložení všech čipů a závodníků není zanedbatelná. Data nemohou být ukládána na pozadí, protože by se mohla aplikace dostat do nekonzistentního stavu. Uživateli je tedy zobrazen indikátor postupu, anglicky progress bar, dokud nejsou všechna data uložena. Možným řešením je s každým online přihlášením smazat a znovu uložit nová data. Vzhledem ke zmíněné režii je tento scénář nevyhovující. Součástí přenesených informací je datum poslední aktualizace čipů a datum poslední aktualizace celého závodu. Shoduje-li se nově přijaté datum s datem z klientské aplikace, nedochází pak k aktualizaci dat v databázi.

Sejmutím čipu dojde k pokusu o odeslání zaznamenaných informací. Aplikace opět odešle metodou GET hodnoty pro zpracování na serveru. Struktura dotazu je uvedena níže:

```
https:.../Endpoint/GatePass?gateID=$1&time=$2&chipNumber=$3&raceID=$4
```

- $1, $4 - jednoznačné identifikátory časové brány a závodu
- $2 - čas, kdy byl sejmut čip
- $3 - EPC hodnota čipu

Validní přijetí a zpracování dat následuje odpověď s HTTP statusem 200 (OK). V opačném případě je odpovědí prázdná zpráva se statusem 400 (bad request).

[5] http://libusbdotnet.sourceforge.net/V2/Index.html

Kapitola 6

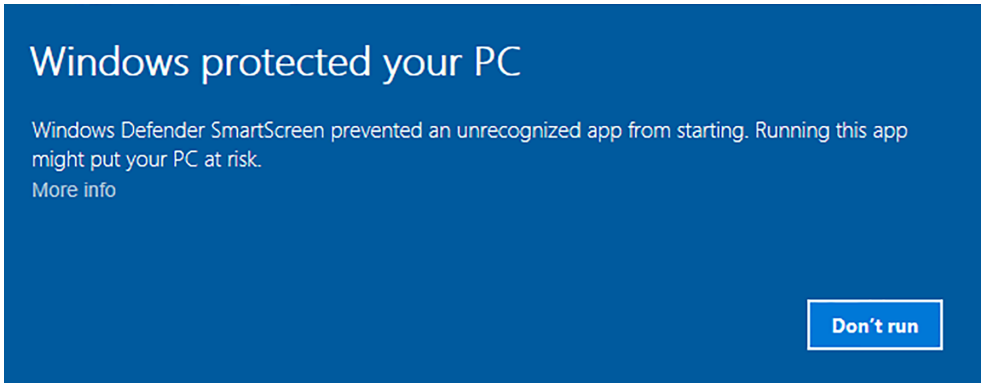
Testování a rozšíření systému

Kapitola je rozdělena do dvou sekcí. První seznamuje čtenáře s testováním klientské aplikace v reálných podmínkách. Obsahem druhé sekce je rozšíření celého systému pro běžecké závody.

6.1 Testování klientské aplikace

První testování proběhlo v srpnu 2016 při závodě Hostýnská osma 2016. Administrační, ani prezenční rozhraní nebylo v tomto období implementováno. Testována byla základní verze klientské aplikace, která komunikovala se serverem hlavního organizátora. Aplikace zaznamenala sejmutý čip a odeslala jej s časovým razítkem na server. Současně byly sejmuté čipy ukládány na pozadí pro případnou synchronizaci. Již při instalaci se objevil problém se zabezpečením. Aplikace nebyla a doposud není digitálně podepsána, což má za následek zablokování stažení a případnou instalaci.

Existuje velký počet autorit pro podepisování desktopových aplikací (Comodo, VeriSign, Thawte atd.). Nejlevnější roční certifikáty začínají na několika tisíci korunách. Optimálnějším řešením se jeví vytvoření vlastního certifikátu a jeho instalace na cílový počítač. Postup vytvoření vlastního certifikátu a jeho správa [16] se skládá z časově náročných a netriviálních kroků. Prozatím je tedy nutné vypnout v době stažení a instalace antivirové programy. Před instalací je navíc zobrazeno protekční okno operačního systému informující uživatele o neznámém vydavateli [6.1].



Obrázek 6.1: Ochrana Windows před neznámými vydavateli

Tento postup bude v případě úspěšného rozšíření aplikace eliminován zakoupením digitálního podpisu.

Zásadní problém nastal rychlým snímáním čipů. Vstupní textové pole nemělo nastavený data kontext a kontrolovala se délka zapsaného řetězce. Délka deseti zapsaných symbolů znamenala sejmutí čipu. Absence technologie binding a způsob kontroly sejmutí způsobil, že aplikace nestíhala zapisovat a ukládat čipy. Nastávaly případy, kdy během zapisování jednoho čipu přicházely vstupní události na zapisování druhého čipu. Výsledkem byla zaznamenaná hodnota skládající se ze dvou čipů. Optimalizace je popsána v podkapitole [5.2.4].

Další implementační nedostatek byl odhalen po nečekaném vypnutí notebooku, na kterém běžela klientská aplikace. Časová brána *Lázy okraj* závodu Hostýnská osma [1] měla problematické připojení k internetu. Pověřená osoba zapomněla připojit notebook do elektrické sítě a po průchodu 14 závodníků se notebook vypnul. Vzhledem k problematickému připojení k internetu byly odeslány pouze 3 výsledky a ostatní měly být poslány ze souboru v rámci synchronizace. Sejmuté výsledky se udržovaly v dočasné paměti aplikace a každou minutu byly zapsány do souboru. Zápisu do souboru předcházelo vymazání obsahu a následný zápis výsledků z dočasné paměti. Zřejmě v okamžiku smazání došlo k vypnutí notebooku a výsledky z dočasné paměti nebyly zaspány. Výsledkem byla ztráta jedenácti časů, které se nepodařilo obnovit.

Aktuální klientská aplikace ukládá časy do lokální databáze. Lokální databáze je umístěna do skrytého adresáře *AppData*, kde si typicky programy vytváří své vnitřní adresářové struktury a ukládají data. Neodeslané časy jsou tedy ukládány do této databáze a zobrazeny k odeslání v záložce *Synchronizace*. Současně jsou také periodicky vytvářeny soubory s neodeslanými časy. Celý obsah souboru se nyní nepřepisuje novějším, ale aplikace pouze zapíše nový záznam na konec obsahu souboru. Nečekané vypnutí vede v nejhorším případě ke ztrátě aktuálně posledně zapisovaného záznamu.

Snadněji řešitelný problém se objevil během snímání čipů. Aplikace sice při sejmutí čipu vydala krátké pípnutí, ale na některých stanovištích byl příliš velký hluk a pípnutí nebylo z notebooku slyšet. Nedostatek byl vyřešen dvouvteřinovou změnou pozadí aplikace. Pověřená osoba tedy kromě akustického indikátoru může zaznamenat i zřetelnější visuální upozornění.

6.2 Rozšíření systému

Během vývoje systému se objevily nové podněty a možnosti vylepšení cílového produktu. Ukázalo se, že vytvoření závodu a jeho vhodná reprezentace je nesmírně rozsáhlý úkol. Redukce funkcionality vede ke snížení potencionální nabídky, avšak produkt je lépe udržovatelný a pro koncové zákazníky pochopitelnější.

Razantní změnou je zrušení komplexního prezentačního rozhraní a zavedení vyšší modularity. Registrace, výsledková listina nebo galerie budou upraveny na samostatné moduly, které budou přes nově vytvořené webové API komunikovat se serverem. Výsledkem tak může být komponenta Galerie vložená do vlastních webových stránek.

Rozšiřujícími moduly může být diskusní fórum,rezervační systém pro čipy, platební brána nebo vykreslení trasy na mapovém podkladu.

[1] http://www.hostynskaosma.cz/results.html?raceId=31273

Kapitola 7

Závěr

V rámci této diplomové práce byl vytvořen systém pro organizaci běžeckých závodů s podporou registrace závodníků a možností více stanovišť v průběhu závodu. Systém se skládá ze dvou webových aplikací a desktopového klienta. Administrační webová aplikace slouží k vytvoření a správě závodu. Organizátorovi nabízí definování kategorií, trasových stanovišť, propozic, galerie či přidávání formátovaných novinek. Významným bodem je konfigurace registračního formuláře, což je vytvoření vlastních položek včetně sémantiky (text, číslo, možnosti). Registrační formulář je spolu s detailem závodu publikován na prezentační webové aplikaci. Přihlášením jsou závodníkovi zobrazeny závody, jichž se zúčastnil. V průběhu závodu je možné na více traťových stanovištích snímat RFID čipy závodníků pomocí klientské aplikace. RFID čipy jsou snímány RFID čtečkou, zapojenou do cílovém zařízení. Je-li klientská aplikace připojena na internet, jsou zaznamenané časy zobrazeny na webovém prezentačním rozhraní v aktuálním čase. V opačném případě jsou výsledky později synchronizovány.

Systém byl pravidelně konzultován s organizátory běžeckých závodů. Klientská aplikace byla několikrát úspěšně otestována v reálných podmínkách. Webové aplikace byly demonstrovány organizátorům a očekává se pokračující spolupráce. Vzhledem k měnícím se nárokům a novým nápadům je řešený systém brán jako prototyp, do kterého bude přidaná rozšiřující funkcionalita. Jako příklad můžeme uvést import závodníků nebo převedení galerie do samostatné webové komponenty, jenž mohou organizátoři využít bez nutnosti vytvoření závodu.

Literatura

[1] BUCHALCEVOVÁ, A.; PAVLÍČKOVÁ, J.; PAVLÍČEK, L.: *Základy softwarového inženýrství*. Vysoká škola ekonomická v Praze, 2007, ISBN 9878024512709.

[2] DAJBYCH, V.: *MVVM: model-view-viewmodel*. Duben 2009, [online]. [cit. 06.11.2016].
URL http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[3] DYKSTRA, T.: *Advanced Entity Framework Scenarios for an MVC Web Application* . [online]. [cit. 14.03.2017].
URL https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/advanced-entity-framework-scenarios-for-an-mvc-web-application

[4] DYKSTRA, T.: *Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application* . [online]. [cit. 27.02.2017].
URL https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

[5] GALLOWAY, J.; RASTOGI, P.; ANDERSON, R.; aj.: *Introduction to ASP.NET Identity* . [online]. [cit. 19.02.2016].
URL https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

[6] MSDN: *ASP.NET MVC Overview* . [online]. [cit. 17.10.2016].
URL https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

[7] MSDN: *ASP.NET Overview* . [online]. [cit. 17.10.2016].
URL https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx

[8] MSDN: *Chapter 11: Server-Side Implementation* . [online]. [cit. 27.02.2017].
URL https://msdn.microsoft.com/en-us/library/hh404093.aspx

[9] MSDN: *Data Binding Overview*. [online]. [cit. 27.10.2016].
URL https://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx

[10] MSDN: *Deploying a WPF Application (WPF)* . [online]. [cit. 25.03.2017].
URL https://msdn.microsoft.com/en-us/library/aa969776(v=vs.110).aspx

[11] MSDN: *EnableRowVirtualization Property* . [online]. [cit. 18.11.2016].
URL https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid.enablerowvirtualization(v=vs.110).aspx

[12] MSDN: *Entity Framework* . [online]. [cit. 20.10.2016].
URL https://msdn.microsoft.com/en-us/library/gg696172(v=vs.103).aspx

[13] MSDN: *Implement Remote Validation in ASP.NET MVC*. [online]. [cit. 28.12.2016].
URL https://msdn.microsoft.com/en-us/library/gg508808(VS.98).aspx

[14] MSDN: *Introduction to WPF*. [online]. [cit. 25.10.2016].
URL https://msdn.microsoft.com/en-us/library/mt149842(v=vs.110).aspx

[15] MSDN.: *Overview of the .NET Framework*. [online]. [cit. 15.10.2016].
URL https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx

[16] MSDN: *Signing and Checking Code with Authenticode*. [online]. [cit. 17.10.2016].
URL https://msdn.microsoft.com/en-us/library/ms537364(v=VS.85).aspx

[17] SAMARAS, C.: *Differences Between WPF And Windows Forms*. Srpen 2014, [online]. [cit. 25.10.2016].
URL http://www.myengineeringworld.net/2014/08/wpf-and-windows-forms-differences.html

[18] SMITH, J.: *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern* . [online]. [cit. 13.12.2016].
URL https://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[19] SQLite: *About SQLite*. [online]. [cit. 10.11.2016].
URL https://sqlite.org/about.html

[20] SQLite: *SQL Features That SQLite Does Not Implement*. [online]. [cit. 10.11.2016].
URL https://www.sqlite.org/omitted.htmll

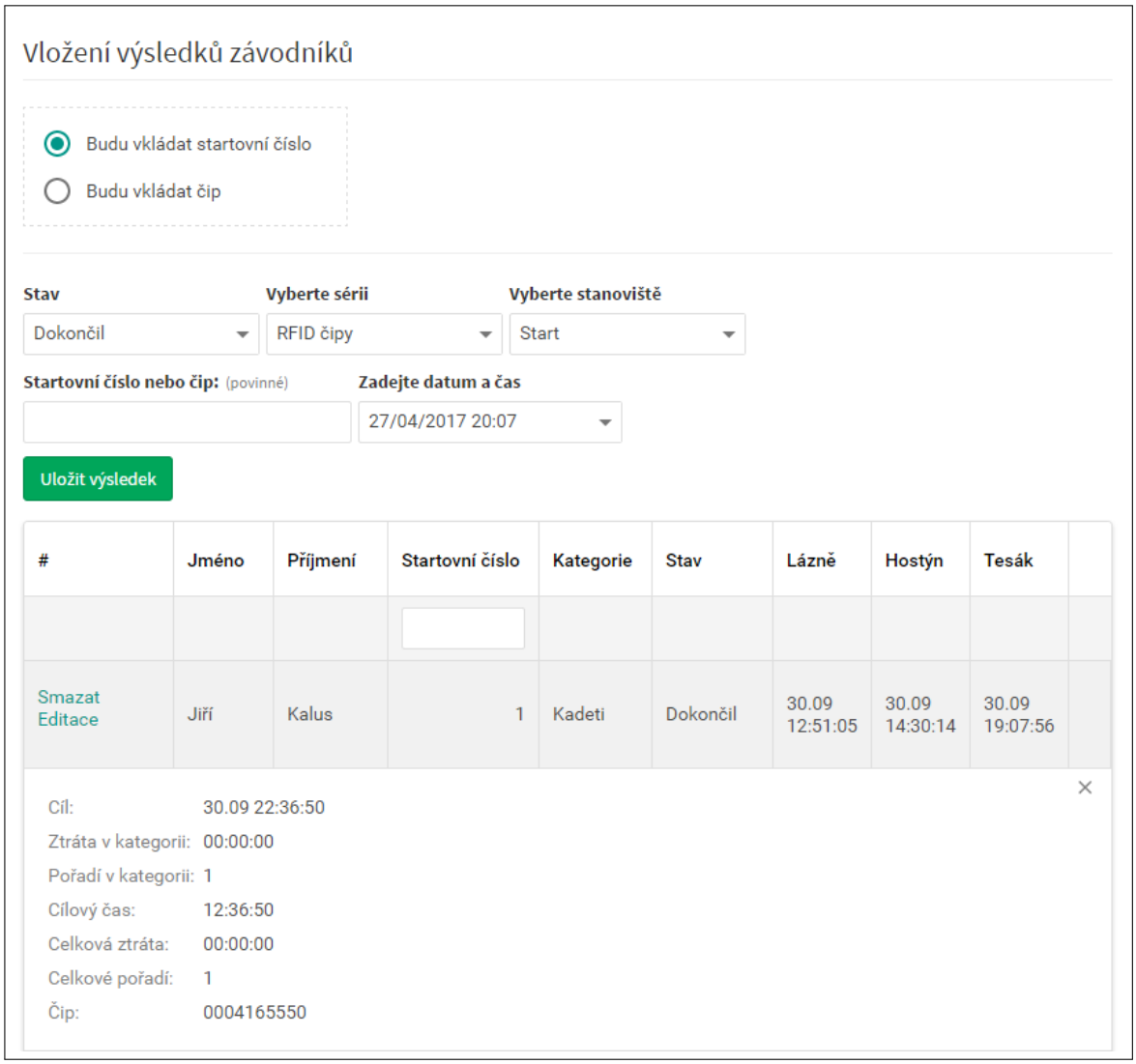
[21] VOJÁČEK, A.: *Používané RFID frekvence a jejich vliv na čtení a zápis tagu*. Září 2015, [online]. [cit. 13.09.2016].
URL
http://automatizace.hw.cz/komponenty-prumyslove-sbernice-a-komunikace/vice-i-mene-bezne-rfid-frekvence-a-jejich-vliv-na-vlastnosti-tagu.html

Přílohy

Příloha A

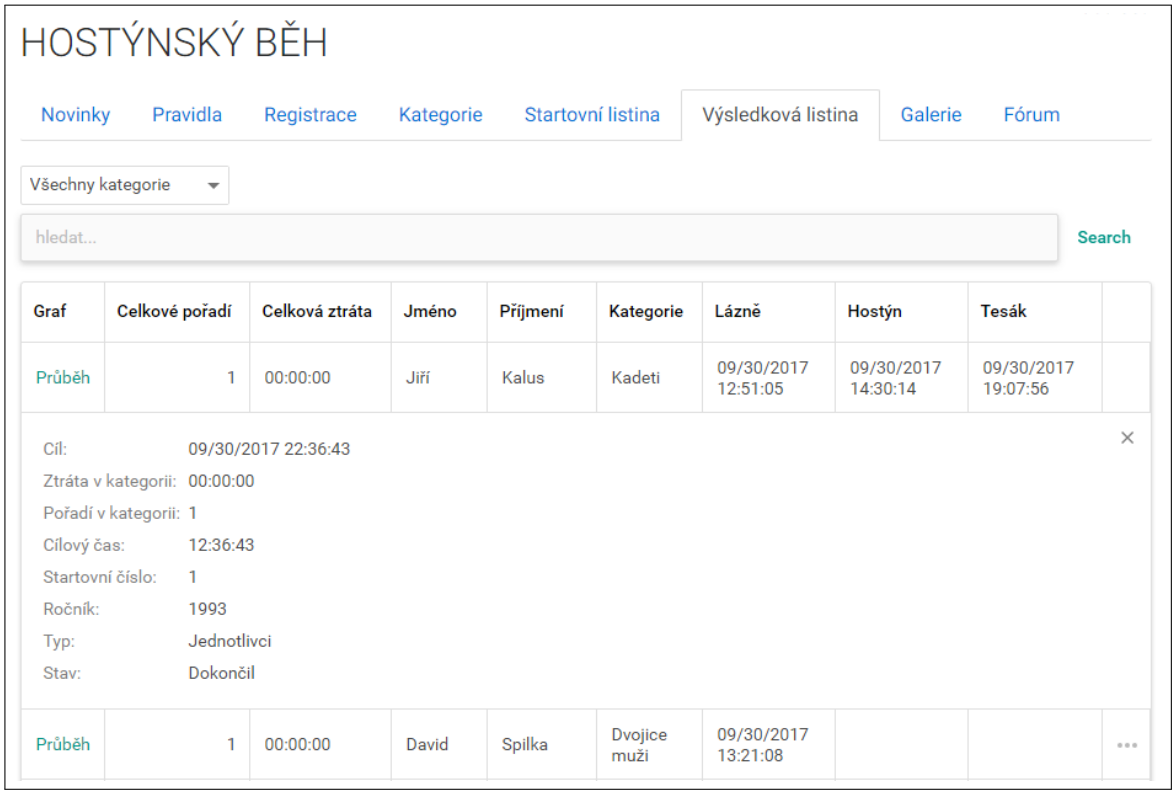
Obrazovky systému

Obrazovka administračního rozhraní pro vložení výsledku závodníka.



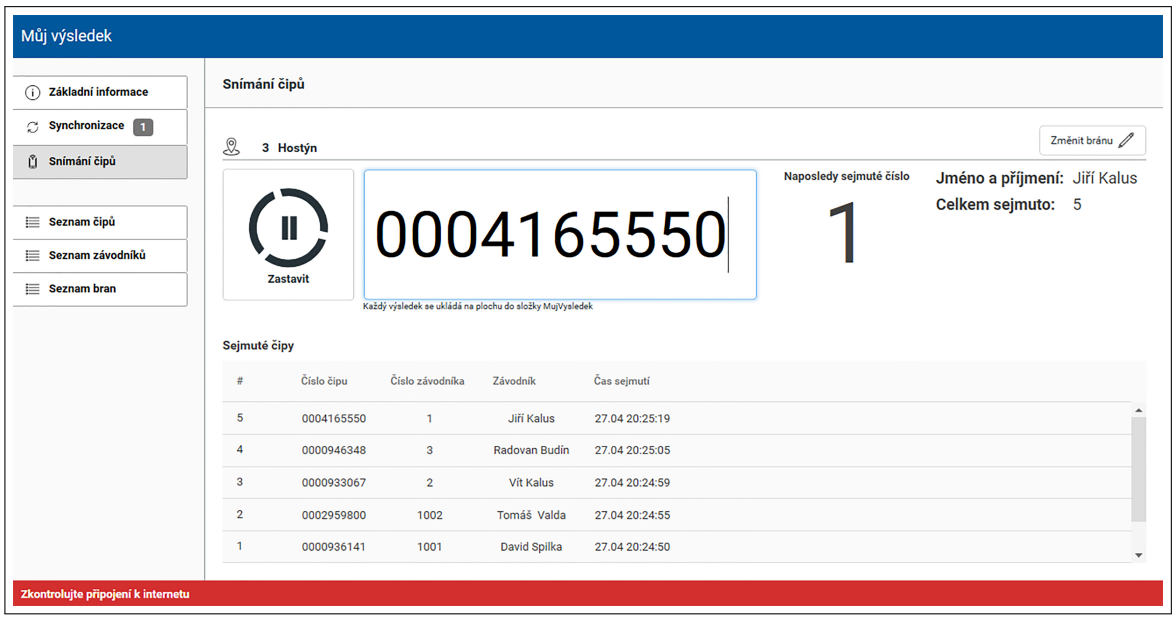
Obrázek A.1: Obrazovka pro vložení výsledku

Obrazovka prezentačního rozhraní zobrazující výsledky závodníků.



Obrázek A.2: Obrazovka pro zobrazení výsledku

Obrazovka desktopové aplikace pro snímání čipů závodníků



Obrázek A.3: Obrazovka pro snímání čipů