Specifikace projektu NOHEC

1 Úvod

1.1 Motivace

Pražský nohejbalový svaz v současnosti řídí sedm dlouhodobých nohejbalových soutěží mužů a jednu soutěž mládeže. Řízení soutěží nemá v podstatě žádnou systémovou podporu. Rozlosování je připravováno pomocí tabulkového procesoru, ke komunikaci s týmy je využíván email, zápisy z utkání jsou pořizovány v papírové formě a pro zveřejnění je využíván obecný portál vysledky.lidovky.cz. Tento způsob řízení soutěží vyžaduje hodně manuální práce, často repetitivní či zbytečné (například přepisování výsledků z papírových zápisů) a je náchylný k chybám.

Nový informační systém ušetří zbytečnou manuální práci, zvýší kvalitu i kvantitu zachycených dat (například střídání během zápasu) a současně poskytne zájemcům rychlejší a přesnější přístup k výsledkům.

1.2 Organizace soutěží

Mužské soutěže sestávají z krajského přeboru a 1. až 6. třídy. V každé soutěži je zpravidla osm týmů, ale ve výjimečných situacích jich může být i méně. Herní sezóna probíhá v rámci jednoho kalendářního roku, hraje se od dubna do konce června, přes prázdniny je herní přestávka a v září se dohrávají zbývající zápasy. V rámci jedné sezóny se v každé soutěži utká dvakrát každý s každým, jednou na domácím hřišti a jednou na hřišti soupeře. Za vyhraný zápas získá tým dva body, za remízu jeden bod a za prohru body žádné. Pořadí týmů v soutěži je určeno součtem bodů případně dílčím skóre. Na konci sezóny postupují do vyšší soutěže dva nejlépe umístěné týmy a do nižší soutěže naopak sestupují dva nejhorší. Z šesté třídy se nesestupuje. Do krajského přeboru postupuje a z něj sestupuje jen jeden tým. Navíc se vítěz krajského přeboru může ucházet o kvalifikaci do druhé celostátní ligy.

Hráči jsou registrováni pod hlavičkou svých klubů. Každý klub na začátku sezóny umístí své hráče na soupisky týmů přihlášených v různých soutěžích. Hráči pak mohou nastoupit do utkání odpovídajících týmů. Každý tým si navíc na soupisku může připsat i hráče svého klubu z nižších soutěží na tzv. fluktuaci. Fluktuanti pak mohou nastoupit až do poloviny zápasů týmu ve vyšší soutěži. Hráči mladší 21 let, tzv. junioři, mohou fluktuovat do všech zápasů vyššího týmu na jehož soupisku jsou připsáni. Hráči mohou mezi kluby mimo sezónu nebo během prázdninové přestávky přestupovat. Navíc je možné kdykoliv během sezóny přestoupit dočasně na tzv. hostování. Hráče kteří nejsou na žádné soupisce, fluktuanty a juniory je možné na soupisky dopisovat téměř kdykoliv během sezóny.

V soutěži mládeže je v současnosti jen malé množství týmů a hraje se tedy jiným turnajovým způsobem. Tento systém je nad rámec tohoto projektu.

1.3 Průběh zápasu

Jeden zápas se skládá z devíti nebo deseti dílčích zápasů. Hra jednotlivců se totiž hraje jen v krajském přeboru. Každý dílčí zápas se hraje na dva hrané sety do deseti bodů. Celkový výsledek zápasu je pak dán součtem vítězství v dílčích zápasech.

Pořadí dílčích zápasů je následující:

Domácí	**Hosté**
1. dvojice	1. dvojice
2. dvojice	2. dvojice
1. trojice	1. trojice
2. trojice	2. trojice
3. dvojice	3. dvojice
(singl)	*(singl)*
1. trojice	2. trojice
2. trojice	1. trojice
1. dvojice	2. dvojice
2. dvojice	1. dvojice

Hráč který nastoupí do 1. dvojice nemůže nastoupit do 2. nebo 3. dvojice a obráceně. To samé platí i o trojicích. Minimální počet hráčů, kteří musí nastoupit do utkání je tedy šest. V dílčím zápase je navíc možné střídat, ale v každé dvojici se smí vystřídat nejvýše tři hráči a v každé trojici nejvýše pět hráčů. Ve hře jednotlivců se střídat nesmí. Každá dvojice (kromě 3. tzv. vložené) i trojice tak vlastně hraje dva dílčí zápasy. Hráči, kteří nastoupili do prvního takového dílčího zápasu, pak musí nastoupit i do odpovídajícího druhého dílčího zápasu. V průběhu každého setu si oba týmy mohou vzít nejvýše jeden třicetivteřinový oddechový čas.

V krajském přeboru, první třídě a druhé třídě dohlíží na průběh zápasu rozhodčí. V nižších soutěžích se po dílčích zápasech v pískání střídají oba týmy. Rozhodčí může ve výjimečných případech udělit hráčům žlutou nebo červenou kartu. V obou případech získává soupeř trestný bod. Pokud nějaký hráč dostane červenou kartu, pak je až do konce zápasu vyloučen.

2 Use cases

2.1 Přihlášení

Aktér: Libovolný nepřihlášený uživatel
Úspěšný scénář:

1. Uživatel otevře přihlašovací formulář
2. Uživatel zadá a potvrdí své přihlašovací údaje
3. Systém ověří správnost údajů
4. Systém uživatele přihlásí

Pokud systém v kroce 3. zjistí, že přihlašovací údaje nejsou správné:

1. Systém informuje uživatele o nesprávnosti zadaných údajů
2. Uživatel může do formuláře zadat přihlašovací údaje znovu
   (use case pokračuje krokem 3. výše)

2.2 Odhlášení

Aktér: Libovolný přihlášený uživatel
Úspěšný scénář:

1. Uživatel stiskne tlačítko pro odhlášení
2. Systém uživatele odhlásí

2.3 Zobrazení výsledkových tabulek

Aktér: Libovolný uživatel
Úspěšný scénář:

1. Uživatel zvolí soutěž, jejíž výsledkovou tabulku chce zobrazit
2. Systém uživateli zobrazí přehledovou stránku pro danou soutěž
3. Uživatel zvolí způsob zobrazení formou tabulky
4. Systém uživateli zobrazí výsledkovou tabulku pro danou soutěž

2.4 Zobrazení profilu týmu a klubu

Aktér: Libovolný uživatel
Úspěšný scénář:

1. Uživatel zvolí soutěž, v které hraje tým, jehož profil chce zobrazit
2. Systém uživateli zobrazí přehledovou stránku pro danou soutěž
3. Uživatel klikne na název nebo logo týmu
4. Systém uživateli zobrazí profil daného týmu

Pokud uživatel chce zobrazit i profil klubu:

5. Uživatel klikne na název klubu
6. Systém uživateli zobrazí profil daného klubu

2.5 Zobrazení odehraných a nadcházejících zápasů

Aktér: Libovolný uživatel
Úspěšný scénář:

1. Uživatel na úvodní stránce zvolí zobrazení všech odehraných zápasů
2. Systém uživateli zobrazí přehled všech odehraných zápasů

Pokud chce uživatel zobrazit nadcházející zápasy:

1. Uživatel na úvodní stránce zvolí zobrazení všech nadcházejících zápasů
2. Systém uživateli zobrazí přehled všech nadcházejících zápasů

Pokud chce uživatel zobrazit zápasy pro konkrétní soutěž:

1. Uživatel zvolí soutěž, jejíž zápasy chce zobrazit
2. Systém uživateli zobrazí přehledovou stránku pro danou soutěž
3. Uživatel zvolí zobrazení odehraných zápasů
4. Systém uživateli zobrazí přehled odehraných zápasů pro danou soutěž

Pokud chce uživatel zobrazit nadcházející zápasy pro konkrétní soutěž:

1. Uživatel zvolí soutěž, jejíž zápasy chce zobrazit
2. Systém uživateli zobrazí přehledovou stránku pro danou soutěž
3. Uživatel zvolí zobrazení nadcházejících zápasů
4. Systém uživateli zobrazí přehled nadcházejících zápasů pro danou soutěž

2.6 Zobrazení odehraných a nadcházejících zápasů hráče nebo rozhodčího

Aktér: Přihlášený uživatel v roli hráč a/nebo rozhodčí
Úspěšný scénář:

1. Uživatel na úvodní stránce zvolí zobrazení svých odehraných zápasů
2. Systém uživateli zobrazí přehled všech odehraných zápasů jeho týmu/týmů nebo zápasy, u kterých byl rozhodčím

Pokud chce uživatel zobrazit jen své nadcházející zápasy:

1. Uživatel na úvodní stránce zvolí zobrazení svých nadcházejících zápasů
2. Systém uživateli zobrazí přehled všech nadcházejících zápasů jeho týmu/týmů nebo zápasy, u kterých bude rozhodčím

2.7 Interaktivní vyplnění zápisu

Aktér: Přihlášený uživatel v roli hráč a/nebo rozhodčí
Úspěšný scénář:

1. Uživatel ve dne zápasu na úvodní stránce stiskne tlačítko *Zahájit zápas*
2. Systém ověří, zda již zápas nebyl zahájen jiným hráčem nebo rozhodčím
3. Systém uživateli zobrazí úvodní stránku pro vyplnění základních údajů
4. Uživatel vyplní kapitána obou týmů
5. Uživatel potvrdí vyplnění formuláře

6. Systém ověří správnost vyplněných dat
7. Systém uživateli zobrazí stránku pro vyplnění dílčího zápasu
8. Uživatel vyplní hráče obou týmů, případné střídání, oddechový čas v každém setu, případná napomenutí a skóre všech setů
9. Uživatel potvrdí vyplnění formuláře
10. Systém ověří správnost vyplněných dat
11. Pokud ještě zbývají nějaké nevyplněné dílčí zápasy, vrátí se scénář zpět ke kroku 7. s odpovídajícím následujícím dílčím zápasem
    Pokud šlo o poslední dílčí zápas, systém zobrazí stránku pro vyplnění podpisů
12. Uživatel nechá kapitány obou týmů a případně rozhodčího potvrdit zápis svým elektronickým podpisem
    (Pokud uživatel zastává jednu z uvedených rolí, pak podpis provede také sám)
13. Uživatel potvrdí vyplnění podpisů
14. Systém ověří validitu podpisů
15. Systém uloží všechna vyplněná data do persistentního úložiště

Pokud systém v kroce 2. zjistí, že zápas již byl zahájen jiným hráčem nebo rozhodčím:
1. Systém informuje uživatele o dané skutečnosti
2. Systém zajistí, že se uživateli již nezobrazuje tlačítko *Zahájit zápas*

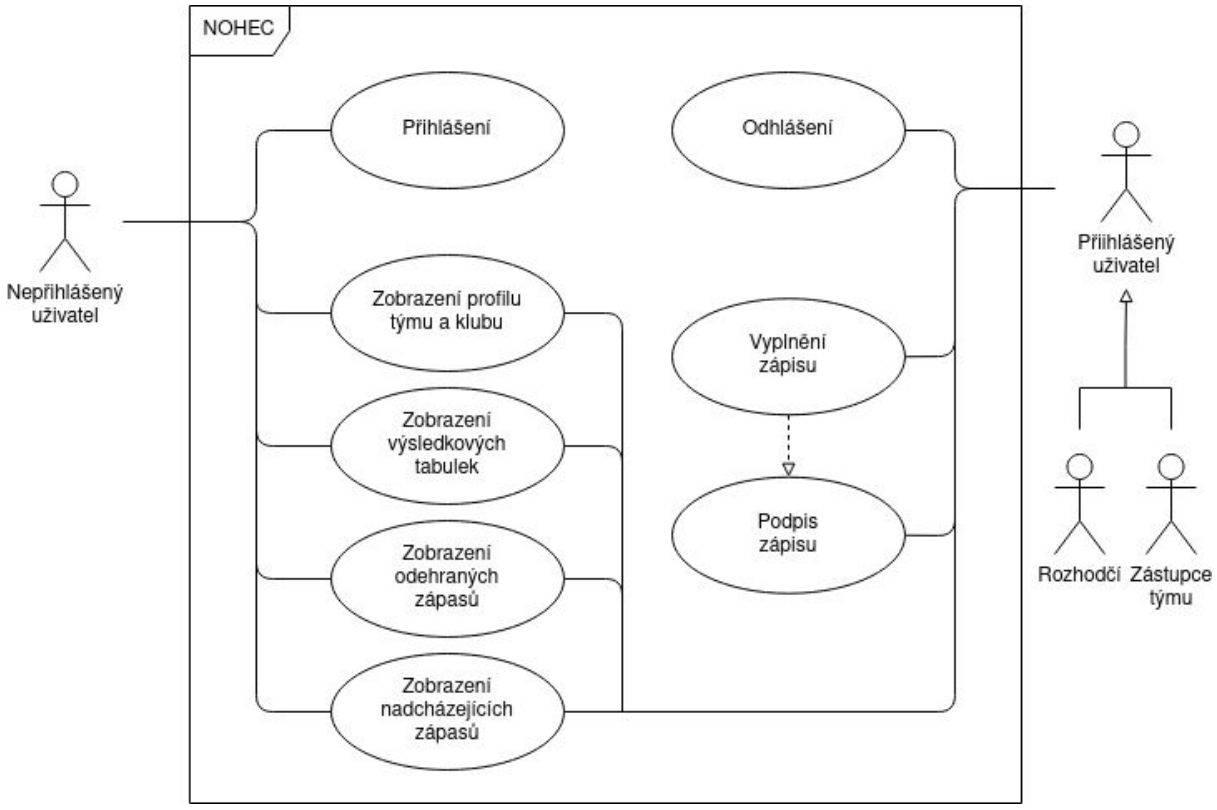
Pokud systém v kroce 6. zjistí, že je formulář špatně vyplněn:
1. Systém informuje uživatele o nesprávnosti zadaných údajů
2. Uživatel může formulář opravit
   (use case pokračuje krokem 5. výše)

Pokud systém v kroce 10. zjistí, že je formulář špatně vyplněn:
1. Systém informuje uživatele o nesprávnosti zadaných údajů
2. Uživatel může formulář opravit
   (use case pokračuje krokem 9. výše)

Pokud systém v kroce 14. zjistí, že podpisy nejsou validní:
1. Systém informuje uživatele o neplatnosti podpisu/podpisů
2. Uživatel může invalidní popis opravit
   (Případně opravit podpis sám, pokud se jedná o jeho podpis)
   (use case pokračuje krokem 13. výše)

2.8 Potvrzení zápisu

Aktér: Uživatel v roli hráč a/nebo rozhodčí
Úspěšný scénář:
1. Uživatel je v den zápasu po jeho ukončení požádán rozhodčím nebo zástupcem domácího týmu k zadání svého elektronického podpisu k vyplněnému zápisu
2. Uživatel zadá svůj elektronický podpis



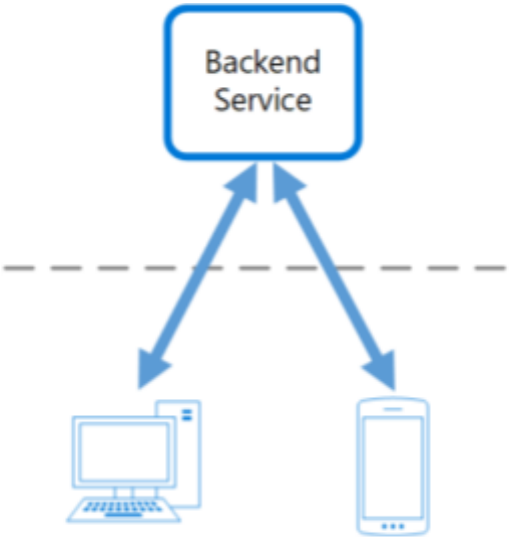
usecase diagram

3 Analýza

3.1 Přehled

Základním odrazovým můstkem při stavbě aplikace je volba použitého modelu a vymezení prostředí v nichž má aplikace operovat. Tato volba následně pomáhá vymezit programovací jazyky, technologie a metodologie použitelné pro tvorbu aplikace. Potřebami naší aplikace je zejména poskytování informací o proběhlých a nadcházejících utkání v nohejbale, autentizace uživatelů, a jejich autorizace pro zadávání výsledků právě hraných utkání. Aplikace by navíc měla být snadno přístupná z většiny přenosných digitálních zařízení jako jsou počítače, mobily a tablety pro umožnění zadávání výsledků utkání během samotné hry.

Uvažovali jsme dva modely které těmto požadavkům dokáží vyhovět, model s centralizovanou entitou schraňující data a spravující přístup k nim, a model s kopií dat na všech zařízeních a systémem propagace změn dat mezi nimi. Po zvážení implementační náročnosti jednotlivých řešení jsme zvolili centralizovaný model klient-server před distribuovaným modelem peer-to-peer. Z této volby přímo plyne požadavek na vytvoření společného serveru, na který se jednotlivá zařízení budou dotazovat pro data a jejich úpravy. Tomuto serveru dále budeme říkat *backend* a aplikaci, kterou uživatelé uvidí budeme říkat *frontend*. Frontend aplikace bude navíc rozdělena na samostatnou webovou, Android a iOS aplikaci pro zajištění maximálního pokrytí zařízení klientů.



Zdroj: https://docs.microsoft.com/cs-cz/azure/architecture/patterns/backends-for-frontends

3.2 Backend

Webový backend je komplexní systém který má za úkol odpovídat na požadavky frontendu. Ukládá kvůli tomu data povětšinou do nějaké databáze nebo přímo do souborového systému. Celý orchestr komponent běží na cloudu a musí být odolný alespoň vůči základním kybernetickým útokům. Zvolením správných technologií na vhodných místech si můžeme ulehčit mnoho práce.

V tomto projektu budeme používat takzvanou vrstvenou architekturu, ve které by se data měla přenášet jen mezi sousedními vrstvami a každá vrstva jim přidává abstrakci.

Jako většina ostatních webových aplikací, i u této bude potřeba ukládat různá data. Nejčastěji samozřejmě zápasy, jejich výsledky, hráče a podobně. Tyto data můžeme trvale uložit několika způsoby, my jsme pro pohodlnost a efektivnost zvolili relační databázi a konkrétně databázový systém PostgreSQL. Rozhodli jsme se pro něj proto, že to je svobodný a open-source produkt a dále je oproti jiným databázovým systémům poměrně vyspělý co se týče funkcionality. Django ORM tento systém taky podporuje.

Vzhledem k tomu, že nechceme naši business logiku psát v jazyku SQL, budeme data z databáze načítat a pracovat s nimi v tzv. business vrstvě. Existuje mnoho různých frameworků postavených na různých jazycích, které jsou k tomu vytvořené a my jsme se rozhodli pro framework Django postavené na jazyku Python. Pro Python jsme se rozhodli kvůli jeho jednoduché syntaxi, která je lehce čitelná i pro lidi, kteří jsou zvyklí na jiné jazyky. Samotné Django jsme zvolili kvůli jeho popularitě, kvůli jeho mnoho funkcím, které jsou dostupné hned po instalaci a také kvůli jeho nepřebernému množství volně dostupných balíčků a doplňků. Na jazyce python existuje i např. framework Flask, ale ten má v základu jen velice málo funkcí a proto je vhodný spíše pro menší aplikace.

Django narozdíl od něj už v základu poskytuje ORM (Objektově Relační Mapování), díky kterému bude náš backend zcela oproštěn od SQL dotazů. Všechny příkazy budeme psát v Pythonu a Django a nich bude za běhu generovat SQL. To je vhodné, když by bylo v budoucnu potřeba změnit databázový systém za jiný. Pokud ho bude Django podporovat, bude potřeba změnit jen pár řádků v nastavení Djanga. Další výhodou je i ochrana proti útokům typu SQL injection, které si Django samo ošetřuje.

Pro jednoduché psaní REST rozhraní v Djangu použijeme Django Rest Framework. Poskytuje extrémně strohé psaní nových pohledů (*ang. views*), poskytuje funkce, které dokážou z instance třídy použité pro definici entity v databázi automaticky vygenerovat JSON a opačně JSON dokážou namapovat na třídu. Často se budeme setkávat i s validací různých formulářů, na příklad pro vložení výsledku zápasu a i pro to má tento framework zjednodušení. Bude nám pak stačit definovat jednoduchou třídu pro formulář a všechny podmínky formuláře zapíšeme jako argumenty. Formuláře tedy ověřujeme jednoduše deklarativně na místo psaní těžce udržitelného a nepřehledného imperativního kódu.

Django sice poskytuje jednoduchý HTTP server, ale ten je vhodný jen pro vývoj na lokálním stroji. Na produkčním serveru jsme se rozhodli použít NGINX z vícero důvodů: vysoký výkon, nízké nároky na paměť, jednoduchá konfigurace, popularita a také je nám tento server dobře známý. Díky jeho popularitě se dá jednoduše najít konfigurace, která nás ochrání před DOS útoky.

V konfiguraci serveru budeme mít vzhledem k efektivitě nastavené dvojení požadavků. Pokud se bude klient dotazovat na statické soubory, odevzdá je přímo ze souborového systému sám nginx a požadavky na ně tedy nebudou směřovány na Django aplikaci.

3.3 Webový frontend

Jak již bylo naznačeno v dřívějších kapitolách, hlavní funkcionalitou aplikace bude zadávání výsledků a monitorování průběhu zápasů v nohejbalu. Možnosti přidávání hráčů do soupisek jednotlivých týmů, variabilní počet dílčích zápasů v rámci utkání dvou týmů a další doprovodné funkcionality kladou značná omezení a požadavky na schopnosti rychlého překreslování a dynamický obsah aplikace.

Při výběru technologie pro implementaci webového frontendu jsme zvažovali následující možnosti:

- React
- jQuery
- Vue.js
- Nuxt.js / Next.js
- Angular

Z těch jsme nakonec vzhledem ke kladeným nárokům a míře zkušeností s jednotlivými technologiemi v týmu zvolili technologii React. Ta je, stejně jako doposud stále oblíbené jQuery, šířena formou knihovny jako node package manager (npm) balíček. Narozdíl od jQuery, které poskytuje širokou paletu nástrojů a funkcionalit pro minimalizaci potřeby jejich manuální implementace, React přináší zcela novou metodologii tvorby responzivního uživatelského rozhraní (UI) postavenou na virtuálním DOM stromě a řízené aktualizaci menších podstromů, čímž umožňuje snadnou tvorbu responzivních webových frontendů.
Za zmínku stojí i fakt, že značná část funkcionalit poskytovaných jQuery byla za dobu existence této knihovny přidána i do samotného jazyka JavaScript. Pro plné využití potenciálu knihovny React je nutné použít rozšíření jsx, které vyžaduje transpilaci do starší verze jazyka JavaScript pro zajištění podpory ve všech současných webových prohlížečích. To nám navíc umožňuje volně užívat i funkcionality novějších verzí JavaScriptu od ES6 po ES10, které nejsou dosud podporovány prohlížeči a s pomocí transpilačního nástroje Babel vše před nasazením převést do legacy kódu.

Na podobném principu jako React pracuje, v určitém slova smyslu konkurenční, balíček Vue.js. Knihovna React však mimo převládající rozšířenost vyniká i v přímočařejším propojení JavaScriptu s HTML pomocí jsx a ve větší volnosti ve výběru doprovodných knihoven pro funkcionality jako state-management nebo routování, a to díky jejich vyššímu počtu.
Rozhodnutím použít React na místo Vue.js na frontendu a Python na místo Node na backendu jsme na jednu stranu zavrhli možnost použití výborných frameworků jako je Nuxt.js nebo Next.js, na druhou stranu máme ale stále možnost využít “framework” create-react-app (CRA).

CRA je bezpochyby jeden z nejjednodušších způsobů instalace knihovny React do webového frontendu aplikace. Mimo prvopočáteční instalaci však knihovna poskytuje i zjednodušené rozhraní pro práci s bundling utilitou Webpack, jejíž manuální konfigurace dokáže mnohdy vývoj komplexních aplikací značně zkomplikovat. Na druhou stranu, toto zjednodušené rozhraní poskytuje pouze omezenou podmnožinu funcionalit Webpack utility. Pro naše potřeby jsme se proto rozhodli CRA nevyužít a konfigurovat React i Webpack manuálně. Toto rozhodnutí nám umožní flexibilně reagovat na případné speciální či neočekávané požadavky plynoucí z rozdílných technologií používaných na frontendu a

backendu aplikace. Plyne z něho sice i zvýšené riziko konfiguračních komplikací, avšak vzhledem k očekávané velikosti webové aplikace pohybující se mezi malou až střední lze tyto komplikace považovat za triviální.

Poslední uvažovanou technologií vedle zvoleného Reactu je Angular. Po stránce výkonu jsou si tyto technologie vcelku rovné, na rozdíl od Angularu však React poskytuje výraznější volnost ve výběru použitých doprovodných technologii i samotné struktuře aplikace. Kvůli množství obsažených funkcionalit je navíc “boilerplate” Angular aplikací násobně větší v porovnání s výslednými aplikacemi postavenými na Reactu. Rozhodujícím faktorem při výběru však zůstávají míry zkušeností s Reactem a Angularem v týmu, kde React jednoznačně převládá.

Závěrem, dříve zmíněná transpilace se kterou pro náš webový frontend již počítáme nám také umožňuje JavaScript nahradit slabě typovaným jazykem TypeScript. Tento jazyk obsahuje veškeré funkcionality standardního JavaScriptu, rozšířené o možnosti statického typování. Tato jeho vlastnost vyniká především v test-driven vývoji kde pomáhá definovat jasný vzhled aplikace, nebo rozsáhlých aplikacích kde udržuje konzistenci a zvyšuje čitelnost kódu. Hlavní nevýhodou jazyka jsou však vyšší časové nároky na tvorbu aplikací, klesající právě až s rostoucí velikostí vyvíjené aplikace. Nabízející se kompromis použití funkcionalit z podmnožiny JavaScript spolu s využitím typování pouze na místech kde zvýšená čitelnost převažuje časové náklady je obecně považováno za špatnou praktiku. Po zvážení očekávaného rozsahu aplikace a omezeného časového fondu na její doručení jsme se proto rozhodli pro použití jazyka JavaScript.

3.4 Mobilní aplikace

Mobilní aplikace je alternativou k webovému rozhraní. Uživatelé v ní budou moci zapisovat průběh zápasu i sledovat výsledky již ukončených zápasů.

Mobilní aplikace lze psát dvěma způsoby. Buď nativně, nebo pomocí frameworku. Oba přístupy mají své výhody a nevýhody. Nativní přístup má tu výhodu, že aplikace bude mít přístup ke všem systémovým API, poběží rychleji, bude mít plynulejší animace a uživatelské rozhraní bude působit nativně - přirozeněji pro daný systém. Nevýhodou je svázanost s danou platformou a tedy aplikace nebude nijak přenositelná na další platformu.

U frameworku existují dva druhy přístupu. Buď se jedná v podstatě o webovou aplikaci, která běží ve WebView (např. Ionic) nebo se aplikace kompiluje do nativního kódu (např. React Native, Flutter, Kivy).
Varianta přes WebView umožňuje mít v podstatě stejný kód pro webovou aplikaci jako pro mobilní. Ovšem je s ním spjato celá řada problémů. Nedají se použít žádné funkce systémového API, aplikace běží znatelně pomaleji, nedá se použít offline, ukládání dat je výrazně složitější atd.
Frameworky kompilující aplikaci do nativního kódu některé z těchto problémů řeší. Prostřednictvím frameworku se dají použít některé systémové API (stále ne všechny - zejména ne ty, které jsou specifické pro jednu platformu), aplikace běží rychleji než ve WebView. Aplikace může stále sdílet část kódu, v našem případě buď s Javascriptovým

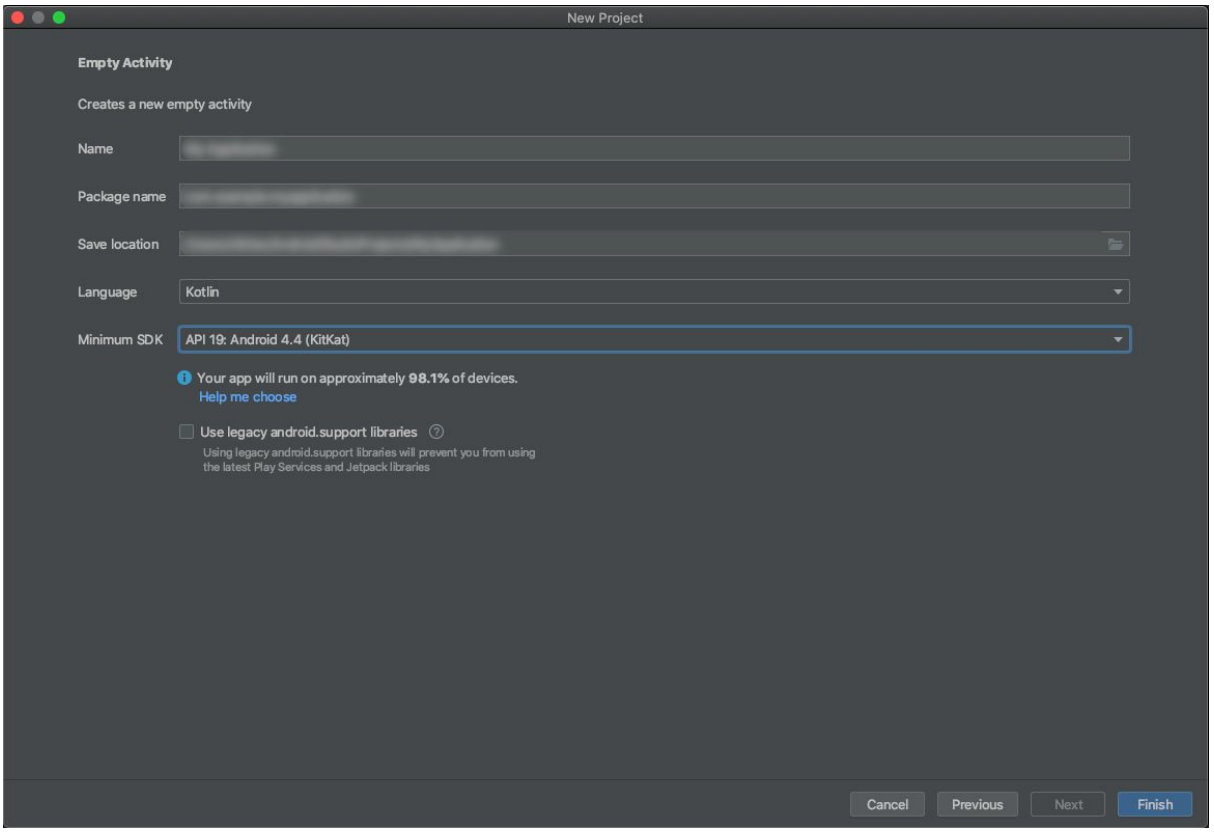
frontendem (React Native), nebo s Python backendem (Kivy). Kdyby se sdílel kód s frontendem, tak by šla použít stejná logika, nicméně samotné UI by bylo potřeba použít odlišné pro webovou verzi, která se používá typicky na počítači, a pro mobilní aplikaci, kde chceme, aby aplikace působila nativně a ne jako webová stránka. Kdyby se kód sdílel s backendem, tak by šla opět použít stejná logika a UI by se muselo udělat pro aplikaci zvlášť.

Vzhledem k tomu, že chceme používat moderní technologie daných platforem, které frameworky typicky implementují se zpožděním, cheme, aby aplikace působily přirozeně pro daný systém a zároveň fungovaly plynule i na slabších zařízeních jsme se rozhodli, že aplikace budeme psát nativně pro každou platformu zvlášť.

3.4.1 Android

Výběr technologií pro nativní Android aplikaci je poměrně přímočarý. Nativní aplikace se dají psát ve dvou jazycích, starší variantou je Java, novější a Googlem již několik let preferovanou variantou je Kotlin. Jelikož chceme v aplikaci používat moderní prvky, které jdou ruku v ruce s možnostmi a syntaxí Kotlinu, bude aplikace napsaná v Kotlinu.

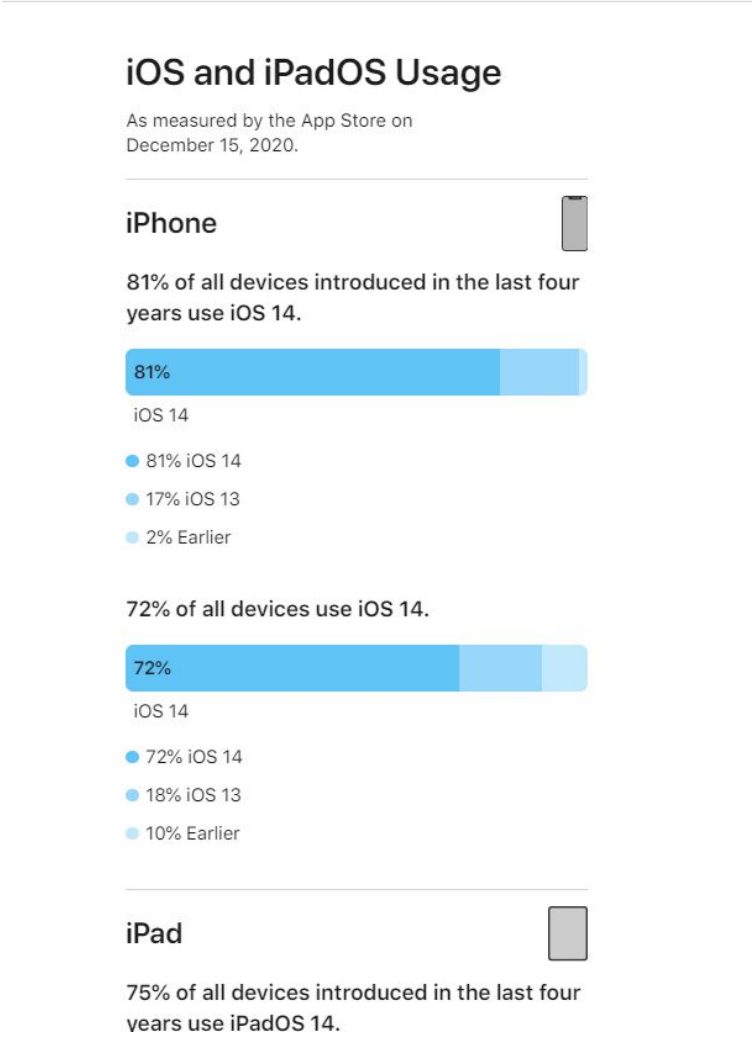
Od roku 2008, kdy spatřil světlo světa první Android, vyšlo už několik verzí. Nejnovější používaná verze je 11. Podporovat všechny verze je v současné době nesmyslné. Jednak by musely být aplikace v podstatě dvě, protože starší verze nepodporují některou novou funkcionalitu a hlavně starší verze Androidu už v podstatě nikdo nepoužívá. Google, v Android studiu, uvádí, že v podstatě 100% zařízení je Android 4.0 a výš. Nicméně Android od verze 4.0 do verze 11 ušel poměrně značnou cestu a proto aplikace bude podporovat zařízení od verze 4.4 (KitKat) a vyšší. Zde Google uvádí zastoupení zhruba 98,1% Android zařízení.



(Android Studio - 12.1.2021)

3.4.2 iOS

Aplikace na iOS bude napsána nativně. Nativní iOS aplikace můžou být napsané ve dvou jazycích: Objective-C, který je podstatně starší, a novější Swift, který byl světu představený v roce 2014 a představuje nový, moderní vývoj iOS aplikací. V této volbě u nás jasně zvítězil Swift, který je preferovaným variantem téměř ve všech ohledech, teda výkon, podpora, rychlost psaní. Co se týká výběru frameworku na tvorbu uživatelských rozhraní, rovněž se nabízejí dvě možnosti, a to: UIKit a SwiftUI. SwiftUI je novější přístup, byl představený v roku 2019 a nabízí nový deklarativní způsob tvorby UI. SwiftUI by měl představovat budoucnost iOS aplikací, a proto jsme se i my rozhodli pro SwiftUI. SwiftUI ale vyžaduje verzi iOS 13 a novější, co by ale neměl být problém, protože oficiální stránka uvádí, že v dnešní době 90% zařízení tuto požadavku splňuje a toto číslo samozřejmě denně roste.



(Apple appstore - 24.1.2021)

3.5 Komunikační protokol

Jak již bylo popsáno výše, backend se stará o ukládání a zpracování dat, které poté předkládá webové i mobilní aplikaci. Pro to, aby si mohli vzájemně vyměňovat data je potřeba zavést nějaký protokol, kterému budou všechny strany rozumět.
Nejzákladněji se protokoly rozdělují na stavové a nestavové. Stavové protokoly fungují tak, že se naváže spojení a to se drží po celou dobu běhu aplikace. To ovšem vyžaduje stabilní internetové připojení a výměnu většího objemu dat na režii připojení, tedy spotřebovává více dat. Pro náš typ aplikace je vhodnější bezestavový protokol. Frontend volá backend jen v případě když potřebuje nějaká data nebo chce data uložit do globálního úložiště.
Nejrozšířenějším takovým protokolem je HTTP (případně HTTPS - rozšíření o šifrování), které se běžně používá na webových stránkách a umí ho zpracovávat i Android.

Nad protokolem je ještě potřeba definovat rozhraní, kterému budou rozumět všechny strany. Možnosti rozhraní je v dnešní době několik, např. Simple Object Access Protocol (SOAP), Representational state transfer (REST), Remote procedure calls (gRPC), GraphQL. SOAP funguje nejlépe s daty v XML formátu, který je díky své zdlouhavé syntaxi náročnější na parsování a na přenos dat a proto je zejména pro mobilní aplikaci, kde typicky záleží na

objemu přenesených dat, méně vhodný než JSON, který se dobře používá s REST protokolem. Zároveň je JSON ekvivalentní Javascriptovému objektu, takže je pro webový frontend úplně ideální a v Androidu i iOSu se s ním dá také dobře pracovat. GraphQL je v podstatě rozšířený REST, kde rozhraní navíc nabízí, že volající si může nadefinovat jaké atributy se mu pro dané objekty vrátí. gRPC umožňuje volání funkcí na objektech na backendu jako by se jednalo o lokální třídy. Používá se zejména v architektuře microservices, kde pomáhá překlenout komunikaci mezi services, které jsou napsané v různých jazycích.

Vzhledem k tomu, že backend bude napsán v jednom jazyce a nebudeme používat architekturu microservices, použijeme raději REST protokol. Zároveň ve většině případů budeme přenášet ucelené objekty a tedy si vystačíme s klasickým REST protokolem bez GraphQL.

4 Architektura

4.1 Backend

Architektura jednotlivých komponent je zobrazena na následujícím diagramu s technologiemi diskutovanými v kapitole 3.1.

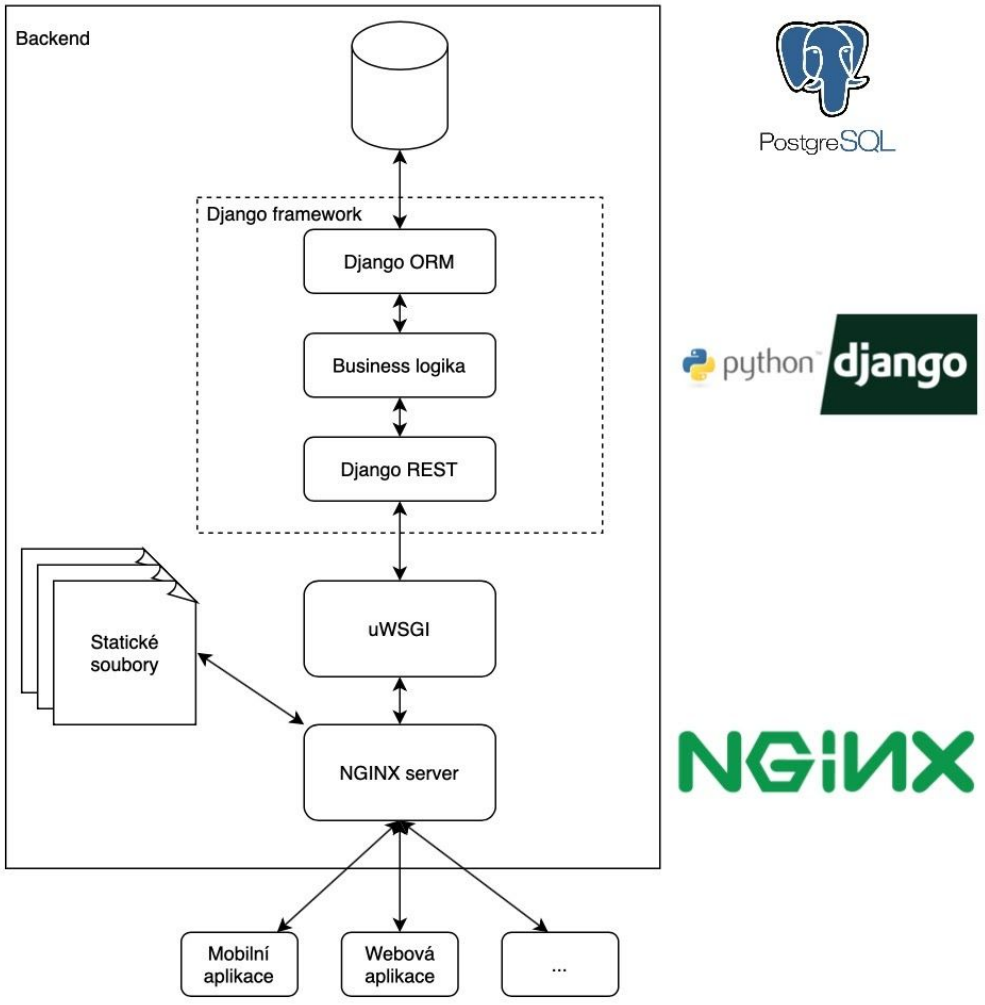


Diagram komponent backendu

Na diagramu komponent můžeme vidět vrstvenou architekturu, ve které každá vrstva komunikuje jen s dvěmi okolními. Tuto architekturu se budeme snažit respektovat i během psaní kódu. Mohou ale nastat výjimky, např. přístup do databáze přímo z Business vrstvy z výkonnostních důvodů.

Celý backend je přístupný přes aplikační HTTP rozhraní, které budou využívat všechny front-end aplikace.

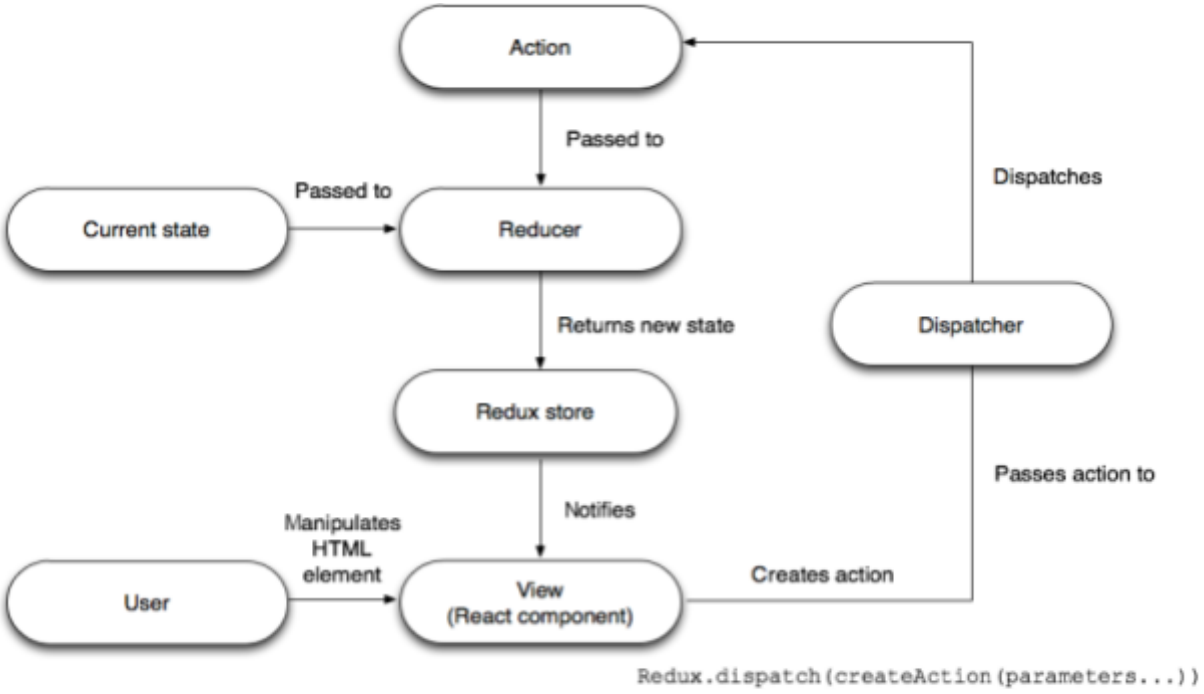
Všechny technologie jsou volně dostupné a poběží na nějaké volně dostupné linuxové distribuci dle nabídky poskytovatele serveru. Webový backend by bylo možné zabalit do nějakého containeru (např. Docker), ale vzhledem k jediné běžící instanci to je zbytečné.

4.2 Webový frontend

Pro správu knihoven použijeme npm package manager.

Aplikace bude fungovat jako single page s knihovnami react-router-dom a history, simulujícími routování na Frontendu.

Pro správu globálního stavu aplikace použijeme redux, společně s redux-saga pro izolaci asynchronních dotazů na server mimo komponenty do samostatných celků komunikujících s komponentami na bázi akcí. Výsledky asynchronních dotazů budou potom zpracovávat reducery a akualizovat příslušné redux stores. Tento klíčový proces je detailně znázorněn i na následujícím diagramu.



Zdroj: https://developer.ibm.com/tutorials/wa-manage-state-with-redux-p1-david-geary/

Pro napojení React aplikace do statického HTML použijeme výchozí react-dom knihovnu, která navíc umožňuje i snadný způsob pro renderování komponent mimo aktuální kontext pomocí techniky react portals.

Absenci typů a obecně jakýchkoliv informací o argumentech přijímajících jednotlivými React komponentami částečně nahradíme knihovnou prop-types která umožňuje ke komponentám nepovinně dodefinovat očekávané argumenty, jejich typy a případné výchozí hodnoty. Runtime následně provádí validaci volání React komponent vůči definovaným pravidlům, čímž lze snadno odhalit případné chyby bez zbytečného vynucování plošného typování.

Komunikaci mezi API a Webovým frontendem nám usnadní knihovna axios, fungující na bázi Promises, které jdou ruku v ruce s redux-saga knihovnou.

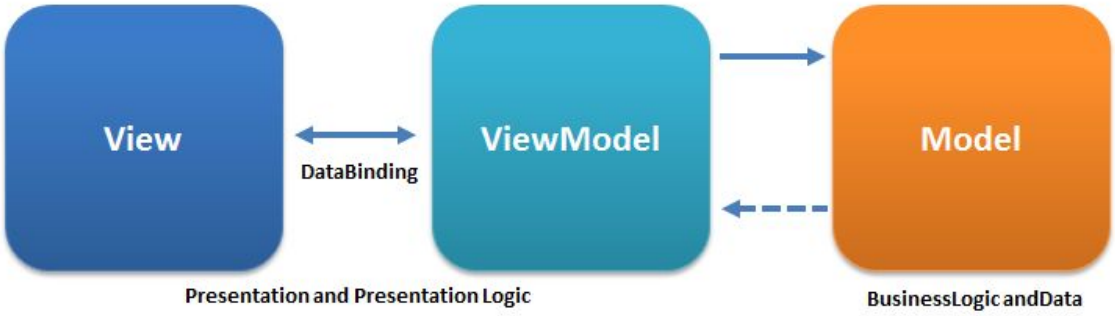
Pro kaskádové styly použijeme knihovnu styled-components která úzce propojuje CSS a Reat, čímž umožňuje definici kaskádových stylů rozšířit o hodnoty proměnných z kontextu React komponent v nichž jsou styly používány a provádět na jejich základě dynamické výpočty v JavaScriptu.

Vzhledem k volné povaze JavaScriptu pro stránce vzhledu kódu pro jeho sjednocení použijeme knihovnu eslint s vlastní dohodnutou konfigurací. Ta poskytuje CLI pro statickou analýzu JavaScript kódu a nástroje pro jeho semi-automatickou opravu, pro zajištění odpovídajícího stylu dle nastavené konfigurace. Tím zajistíme celistvost JavaScriptového kódu od velikosti odřádkování po mezery v destrukturalizaci objektů.

Poslední knihovnou je již zmiňovaný bundler webpack spolu s transpilační utilitou babel a doprovodnými loadery jako css-loader pro resolvování importovaných CSS souborů a jejich přidání do výsledného bundle, nebo filel-loader pro totéž s importovanými statickými soubory jako jsou obrázky.

4.3 Mobilní aplikace

V souladu s doporučením od Android vývojářů, bude v Android aplikaci použita architektura Model-View-ViewModel (MVVM) s databindingem. V prostředí iOS je konsenzus, že budoucnost patří SwiftUI jako deklarativnímu UI frameworku a Combine jako funkcionálnímu reaktivnímu programovacímu frameworku. Architekturou, která toto využívá nejlépe, je také Model-View-ViewModel (MVVM).



MVVM diagram
(Zdroj: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel)

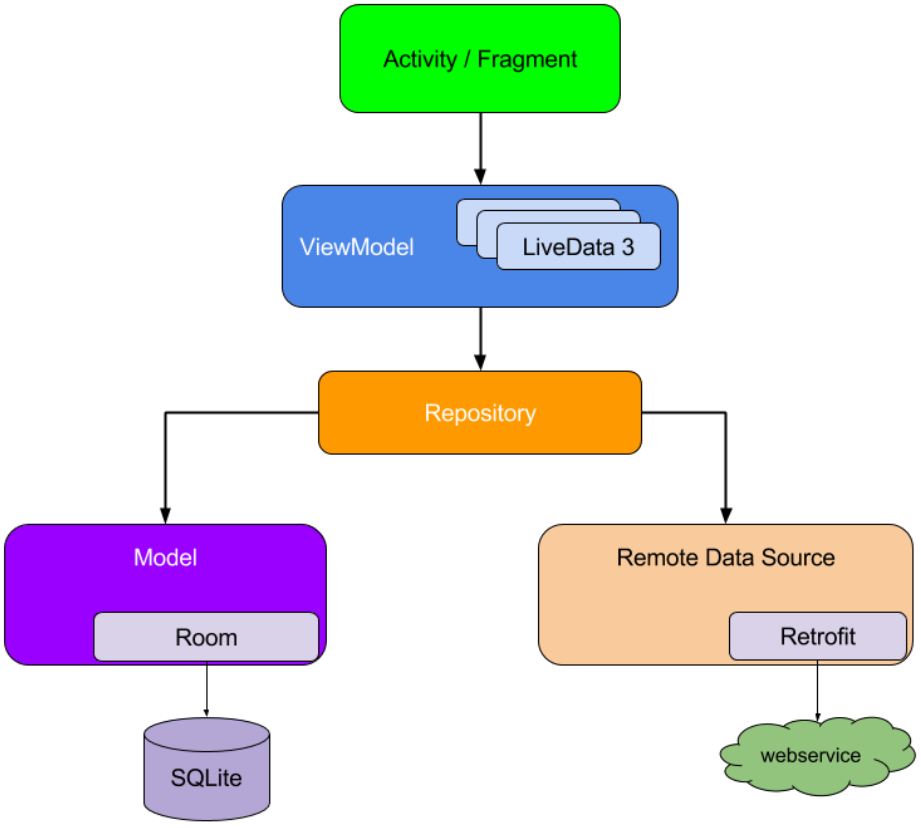
Architekturou se snažíme aplikaci rozdělit do celků podle funkcionality. To nám kromě lepší čitelnosti kódu dovolí jednotlivé menší komponenty znovu použít. Zároveň to udělá kód jednodušeji testovatelný a lépe se budou hledat případné chyby.

Architektura MVVM rozděluje každou obrazovku, nebo její komponentu, do tří základních prvků:

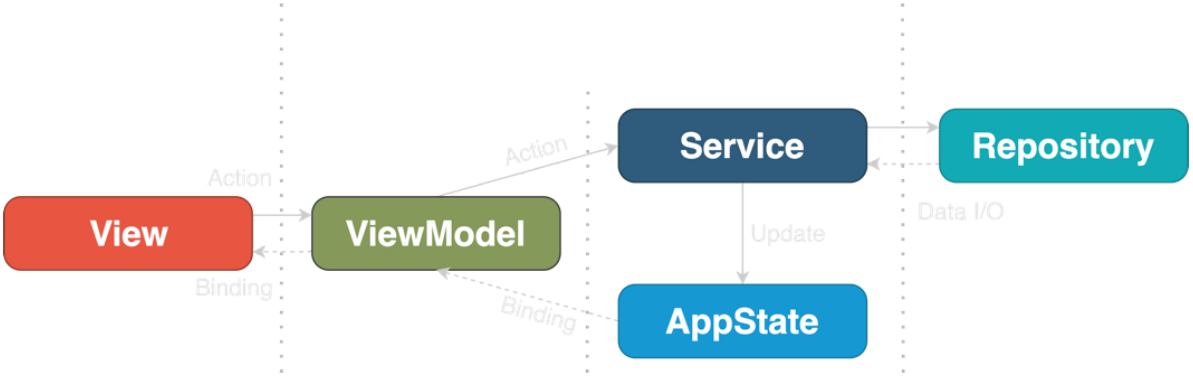
- **Model**
  Určuje strukturu a obsah dat. Data může načítat z lokálního úložiště nebo přes webový backend. Zároveň se stará o ukládání dat, jak do lokálního úložiště, tak na backend.
- **ViewModel**

Načte data z Modelu a zpřístupní je pomocí databindingu View. Zároveň se stará o veškerou business logiku dané komponenty.

- **View**
  Samostatný frontend komponenty, který uživatel vidí. Pouze zobrazuje data, poskytnutá přes databinding z *ViewModelu*.



Architektura Android aplikace
(Zdroj: https://developer.android.com/jetpack/guide)



Architektura iOS aplikace
(Zdroj: https://nalexn.github.io/clean-architecture-swiftui/)

4.4 Rozšiřitelnost

Frameworky, které v tomto projektu využijeme jsou dobře připraveny na pozdější změny.

Například upravení databázového modelu je možné dělat postupně tak, aby nic většího nezměnily.
Příklad úpravy databázového modelu:

1. Vytvoření/změna modelu definovaného v Djangu.
2. Vytvoření třídy, která bude tento model umět serializovat a deserializovat.
3. Použití těchto tříd.

Rozšíření webového frontendu je také bezproblémové. Kód v Reactu je velmi modulární a proto je v nutnosti úprav potřeba jen přidat nový modul, většinou bez zásahu do žádného jiného.

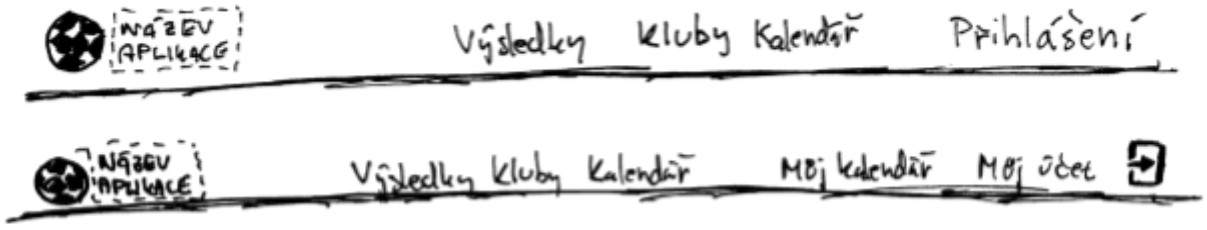
Co se týče rozšiřitelnosti UI o nový typ aplikace, tak to by mělo být udělatelné bez žádných nebo téměř žádných úprav na backendu. To proto, že oproti již stávajícím UI aplikacím pravděpodobně nebude potřeba dotazovat jiné data.

5 UI

Uživatelské rozhraní představuje tvář každé aplikace a je proto důležité, aby dostupné funkcionality aplikace uživatelům poskytovalo snadnou a přehlednou cestou která zároveň i zaujme. V tomto kontextu se často klade důraz na uživatelský zážitek, user-experience, zkráceně UX. Při tvorbě návrhu UI jsme se proto snažili uvažovat i UX cílené skupiny uživatelů aplikace. Jelikož mezi budoucí uživatele dva z členů týmu patří, lze hovořit o úzké spolupráci mezi týmem vývojářů a budoucími uživateli za účelem nalezení optimálního řešení.

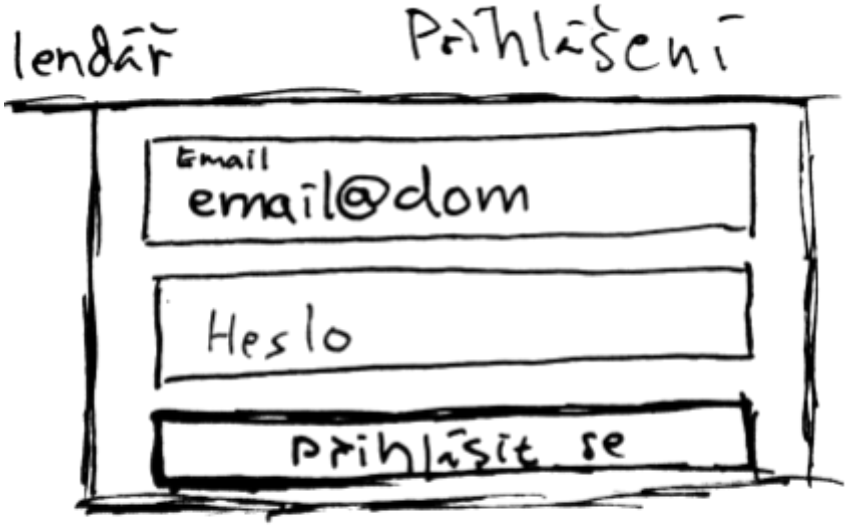
5.1 Webová aplikace

Hlavním ovládacím prvkem většiny webových aplikací je bezesporu navigační panel. Aplikace poskytuje dvě varianty tohoto panelu, jednu pro nepřihlášené uživatele a druhou pro přihlášené. Zároveň panel umožňuje rychlý přístup ke všem podstránkám jako je Kalendář nebo seznam klubů.



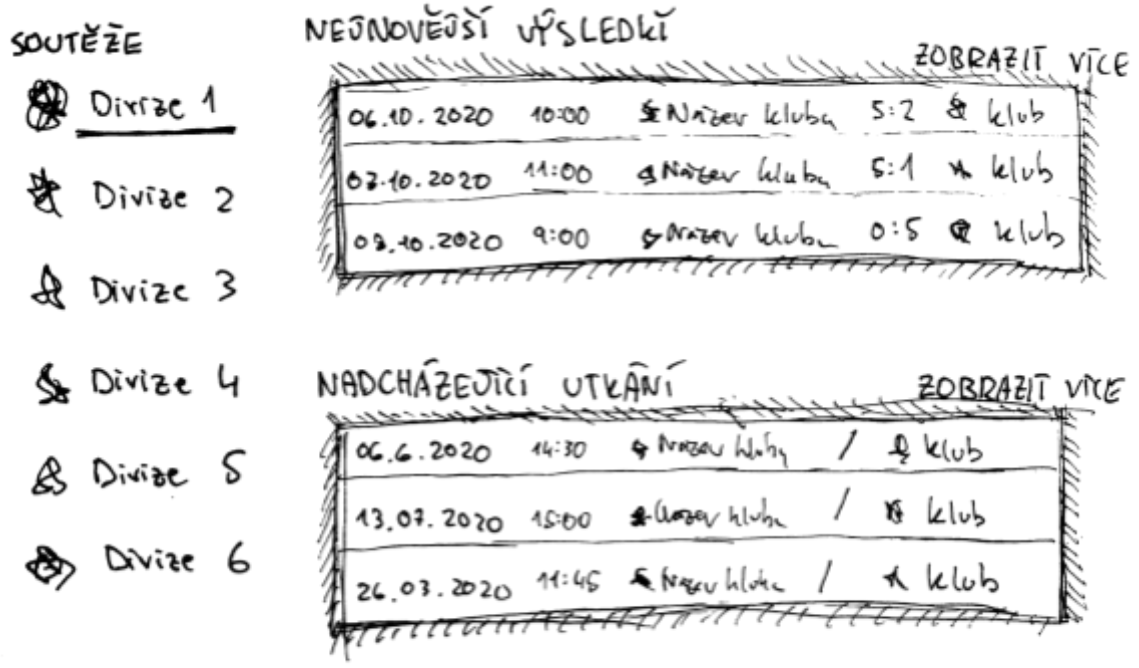
Varianty navigačního panelu

Přístup k rozšířeným funkcionalitám aplikace je umožněn přes tlačítko přihlášení na navigačním panelu.



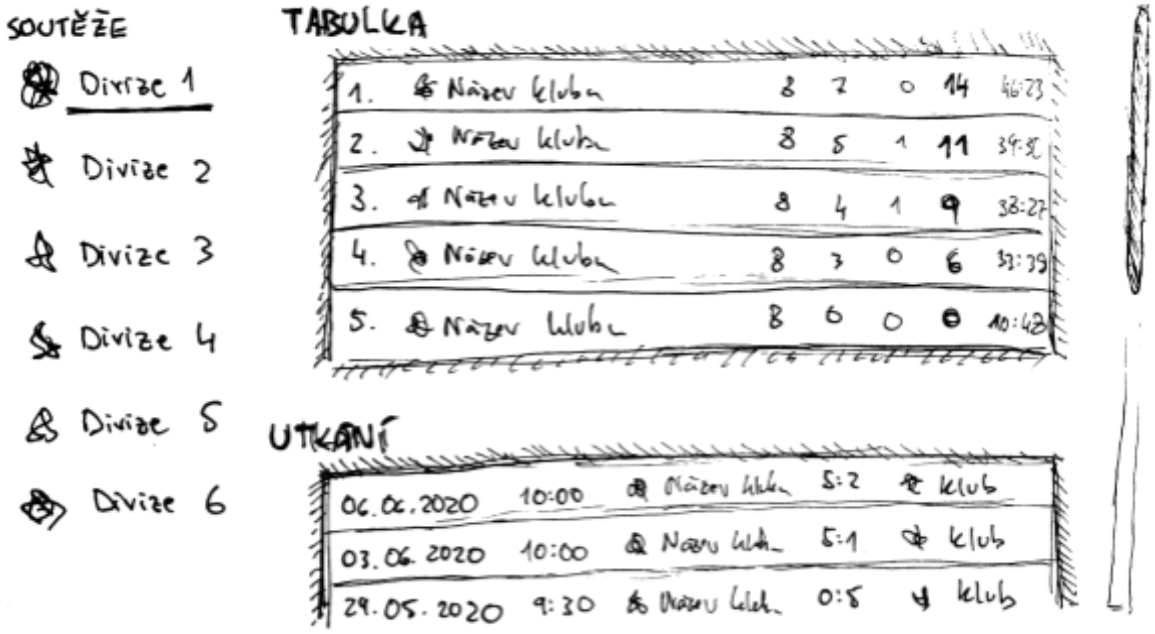
Vyskakovací okno pro přihlášení uživatele

První stránka aplikace na které se uživatel ocitne poskytuje stručný přehled o posledních odehraných a nadcházejících zápasech pro každou ze soutěží.



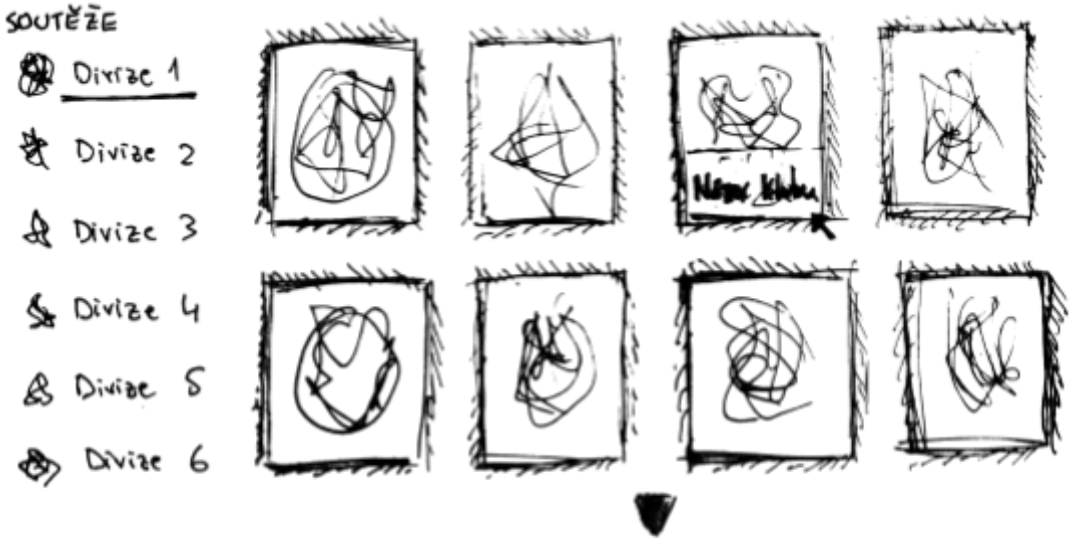
Hlavní stránka webové aplikace

Více informací o proběhlých zápasech a současných umístěních klubů v rámci soutěží poskytuje stránka výsledků.

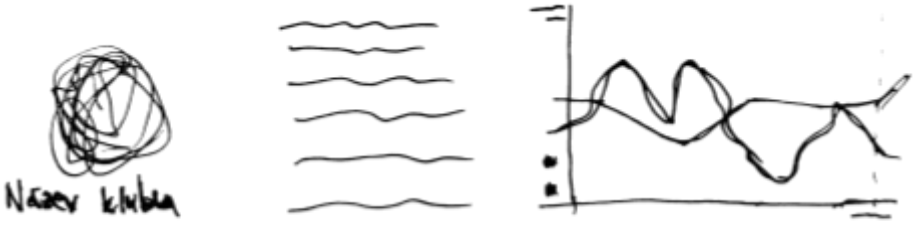


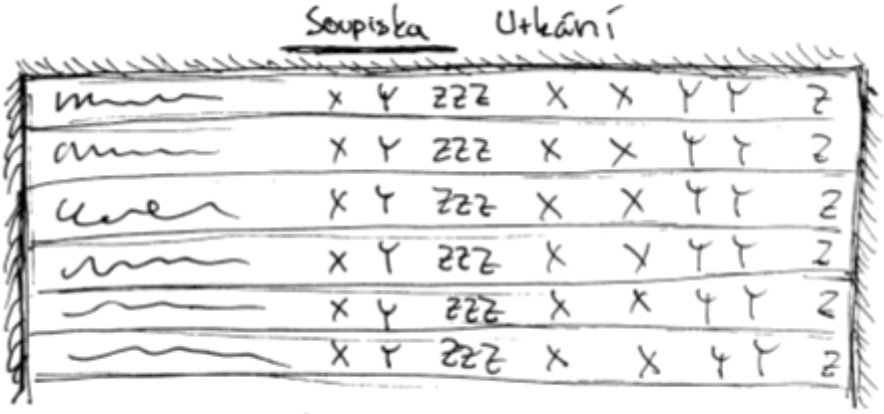
Stránka výsledků a umístění klubů v soutěžích

Stránka kluby umožňuje prohlížení klubů přihlášených do jednotlivých soutěží spolu s možností přejití na stránku detailu klubu s podrobnějšími informacemi jako je jeho soupiska, odehraná utkání a statistiky z nich.



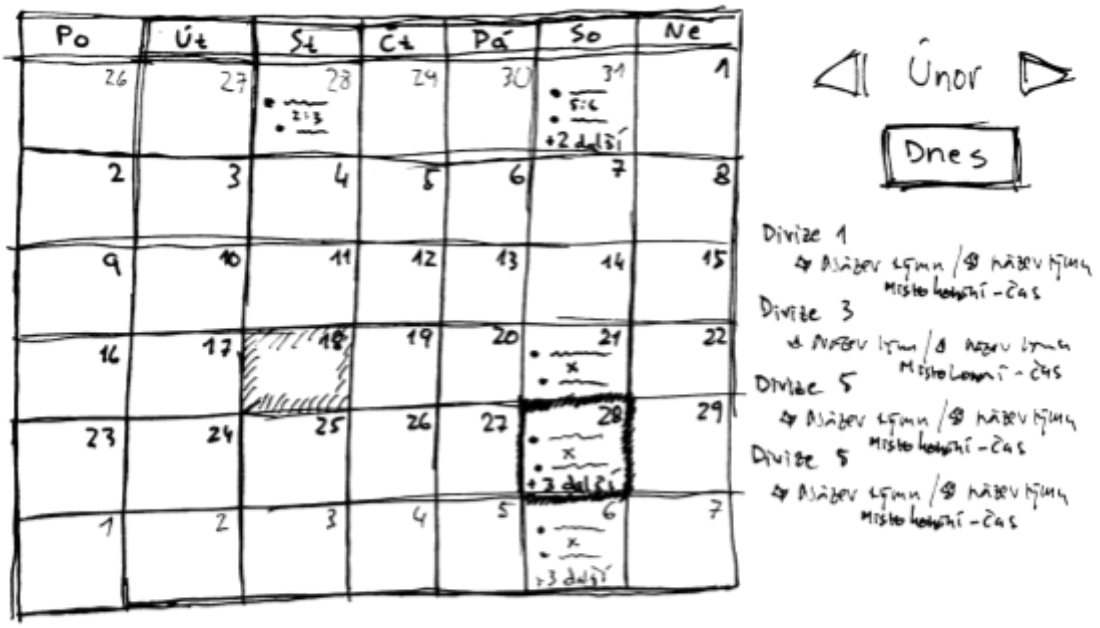
Stránka se seznamem klubů v dané soutěži





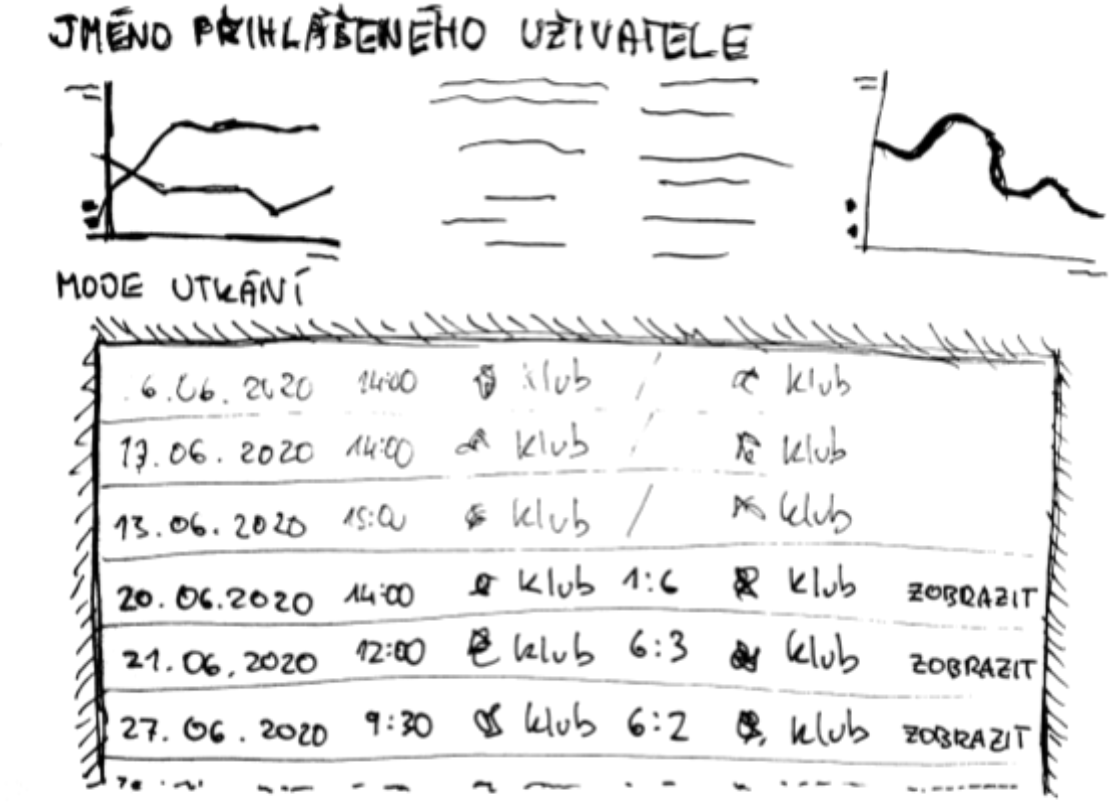
Stránka detailu klubu

Další důležitou komponentou aplikace je dvojice kalendářů - jeden přístupný všem uživatelům, zobrazující veškeré proběhlé a nadcházející utkání napříč všemi soutěžemi a druhý přístupný pouze přihlášeným uživatelům, omezený pouze na utkání kterých se daný uživatel zúčastnil jako rozhodčí nebo klub.



Kalendář k 18.2. s vybraným detailem dne 28.2.

Po přihlášení do aplikace je uživatel přesměrován na stránku se seznamem všech zápasů kterých se účastnil jako rozhodčí nebo hráč, spolu se třemi nejbližšími nadcházejícími zápasy. Stránka by navíc mohla být rozšířena o uživatelovy osobní statistiky.



Stránka Můj účet

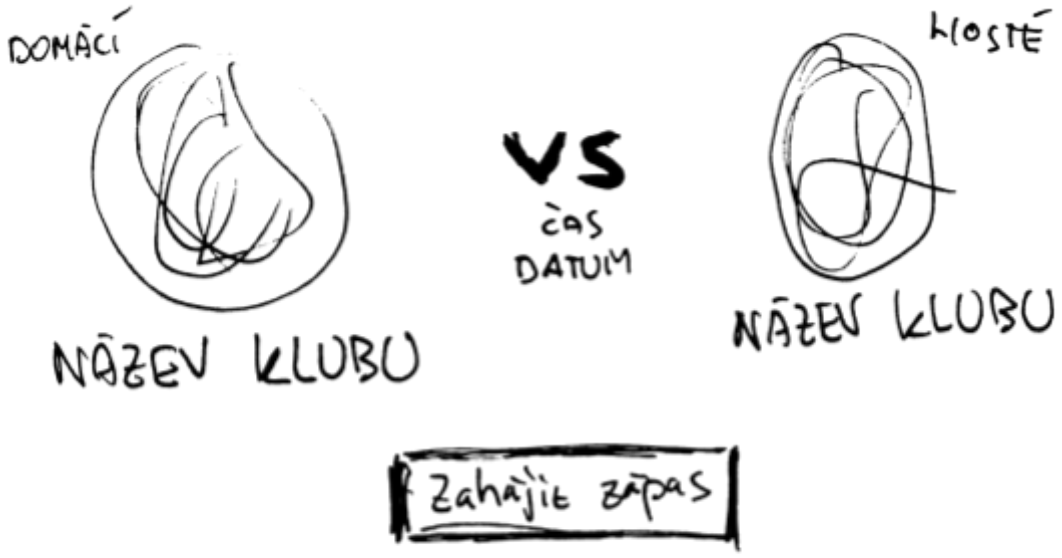
Detailní prohlížení výsledků jednotlivých zápasu spolu s jejich zadáváním a editací umožňuje stránka přístupná proklikem na tlačítko “ZOBRAZIT” ve výpisu utkání. Stánka výsledku již odehraného zápasu se dělí na 3 podstránky, jednu s obecnými informacemi,

druhou s informacemi o hráčích s designem kopírujícím sránku soupisky na detailu klubu a stránku s detailními informacemi o jednotlivých odehraných dílčích zápasech.

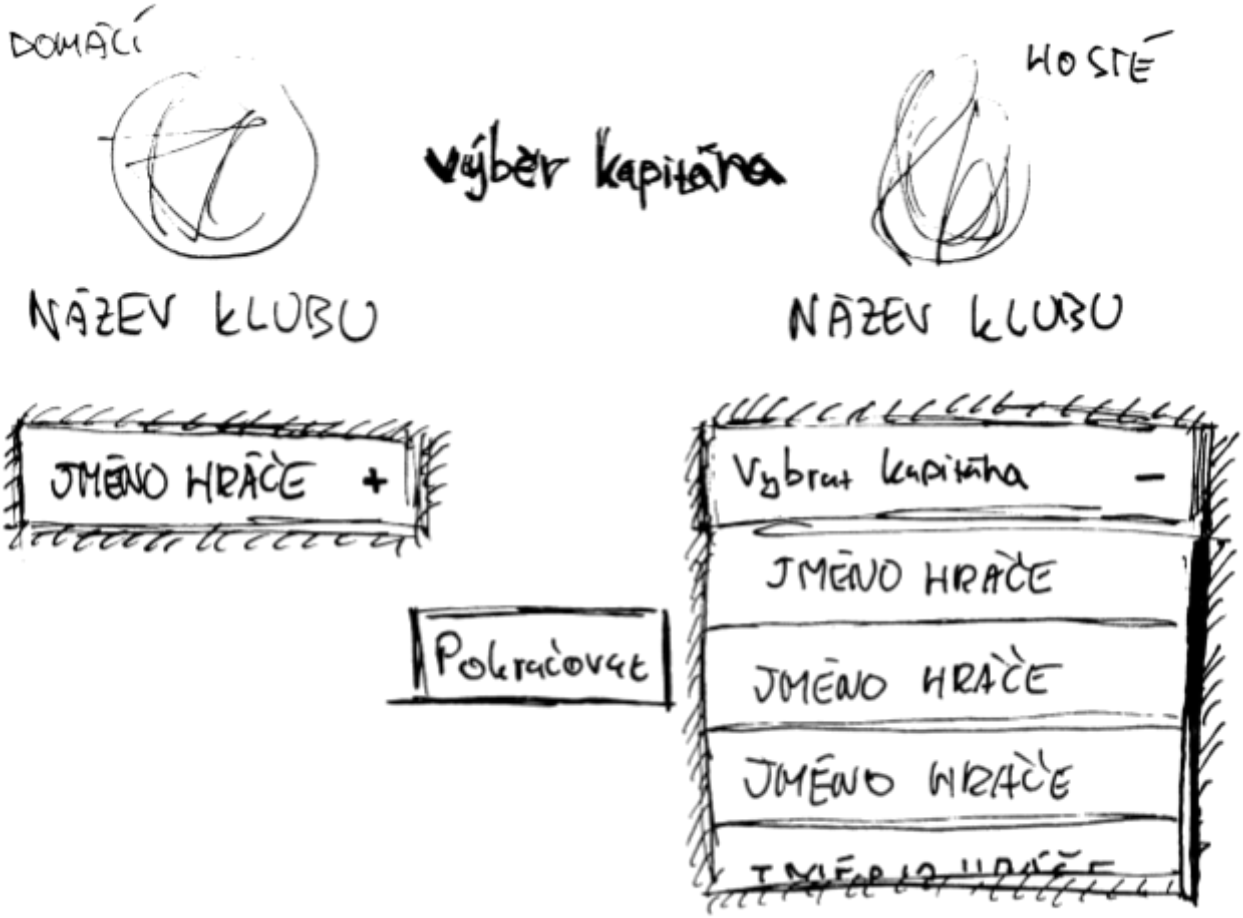
Stránka obecných výsledků

Stránka s detailem na dílčí zápasy

Celým procesem zadávání výsledků od nastavení kapitánů po konečný podpis rozhodčího provede uživatele následující sekvence po sobě jdoucích stránek.



1. Zahájení zápasu



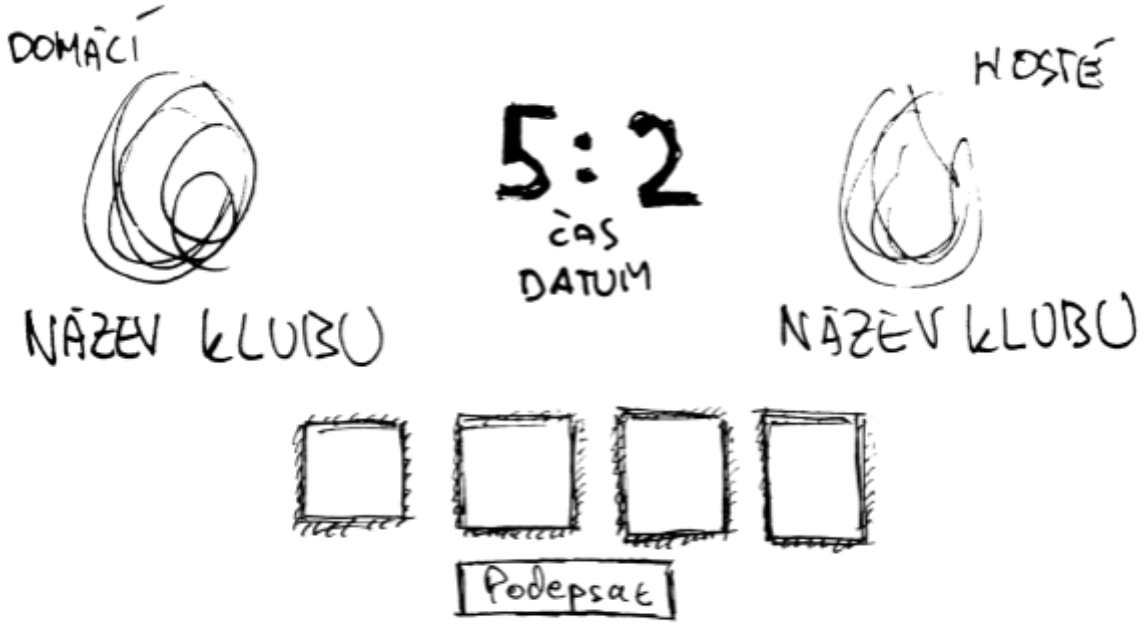
2. Nastavení kapitánů

3. Nastavení sestavy pro aktuální dílčí zápas

4. Zadávání výsledků hraného setu

5. Přidání poznámky k odehranému ukání

6. Souhrn výsledků odehraného utkání



7. Stránka podpisu rozhodčího pomocí 4-ciferného kódu

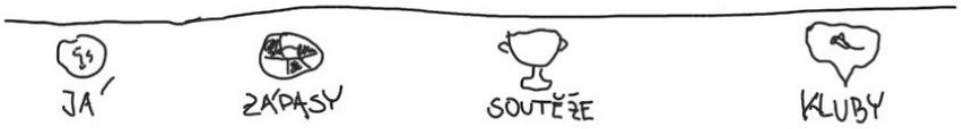
Po úspěšném dokončení zadávání výsledků utkání bude následně uživatel přesměrován na stránku prohlížení již odehraných zápasu pro nově zadaný zápas.

5.2 Mobilní aplikace

Následující rozhraní platí pro obě mobilní aplikace - Android i iOS. Hlavním ovládacím prvkem aplikace bude menu ve spodní liště. Menu bude rozděleno do tří částí *Zápasy*, *Soutěže* a *Kluby.*

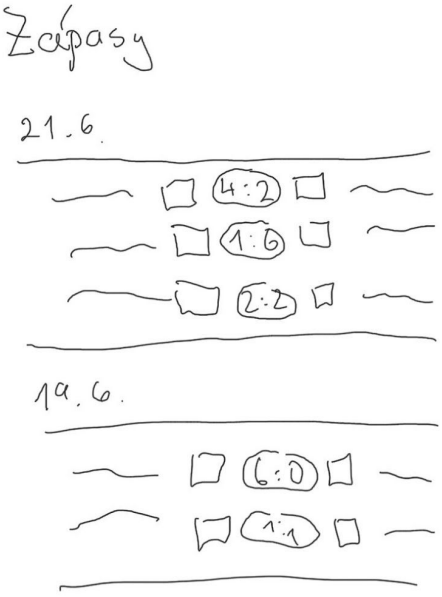
menu pro nepřihlášeného uživatele

Pokud bude uživatel přihlášen, přibude mu ještě jedna záložka k jeho profilu.



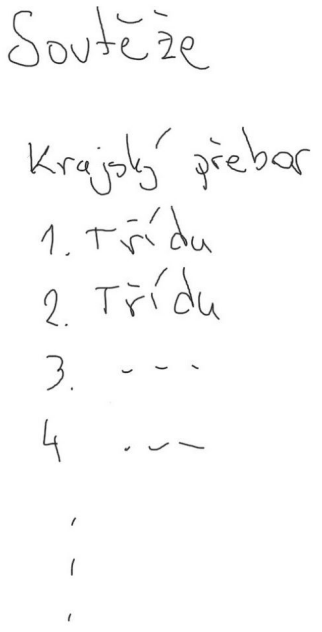
menu pro přihlášeného uživatele

V záložce zápasy budou zobrazeny všechny odehrané zápasy napříč všemi soutěžemi, seřazeny sestupně podle data.



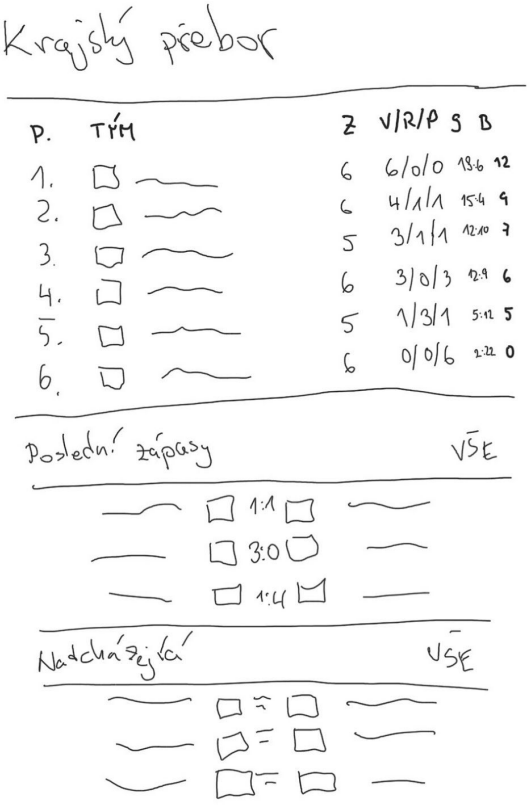
záložka *Zápasy*

V záložce soutěže bude seznam všech soutěžích spadající pod správu Pražského nohejbalového svazu. Po kliknutí na danou soutěž se zobrazí detail soutěže.



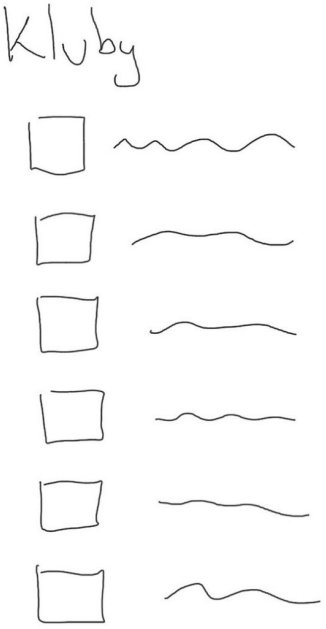
záložka *Soutěže*

V detailu soutěže bude zobrazena průběžná tabulka, odehrané zápasy z posledního kola a zápasy pro kolo nadcházející



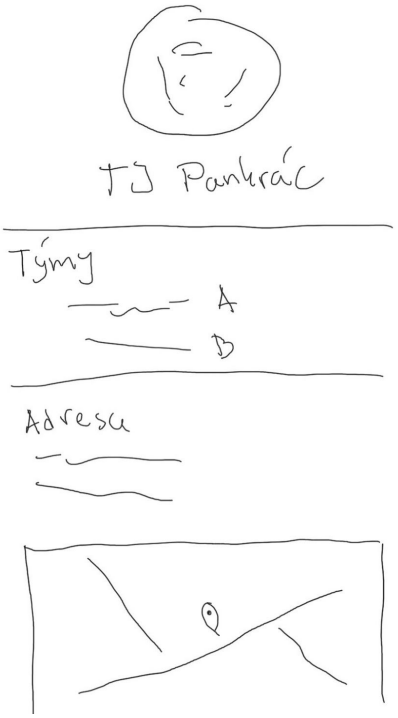
detail soutěže

V záložce Kluby bude zobrazen seznam všech Pražských klubů a po kliknutí se otevře detail klubu.



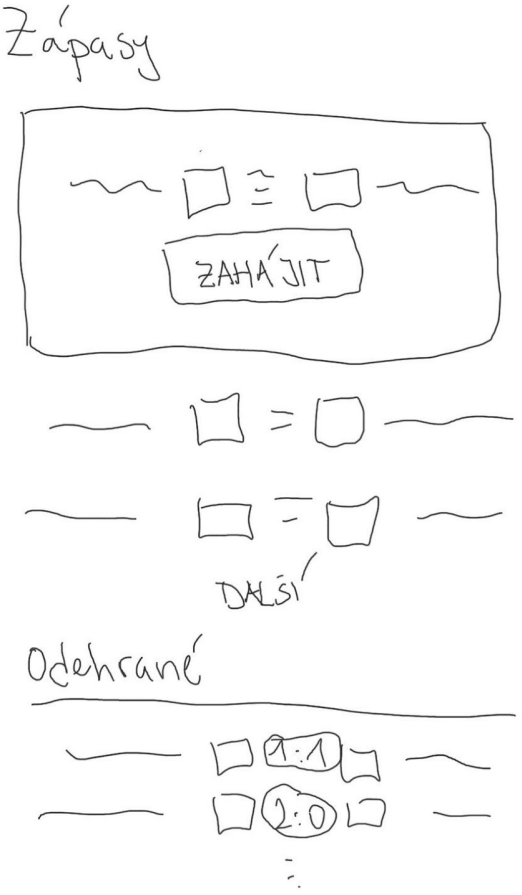
Záložka *Kluby*

V detailu klubu budou zobrazeny jednotlivé oddíly, informace o klubu a adresa hřiště s mapou.



detail klubu

V záložce pro přihlášené uživatele budou zobrazeny zápasy pro daného uživatele. Zápas, který se bude konat v daný den bude zvýrazněn a bude u něj možnost zahájit zápis.

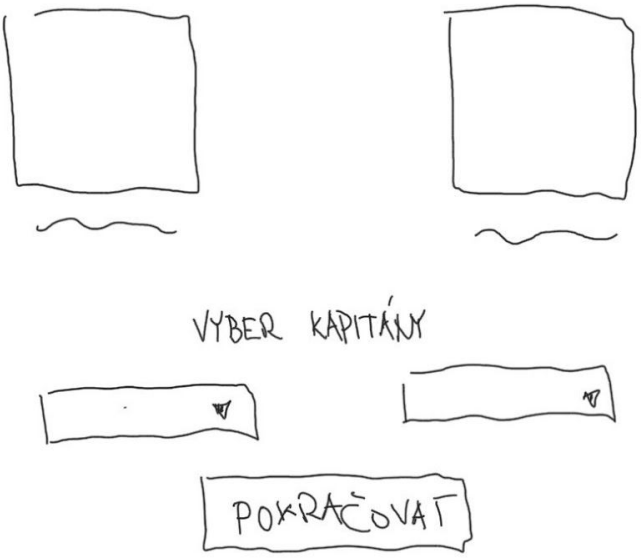


přehled pro přihlášeného uživatele

Zápis zápasu začne shrnující obrazovkou, kde uživatel potvrdí, že chce utkání zahájit.

zahájení zápasu

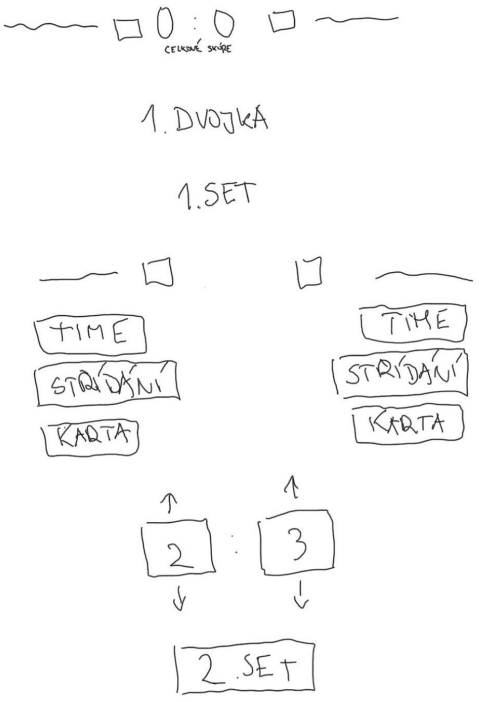
Na další obrazovce vybere uživatel kapitány pro oba týmy.



výběr kapitánů

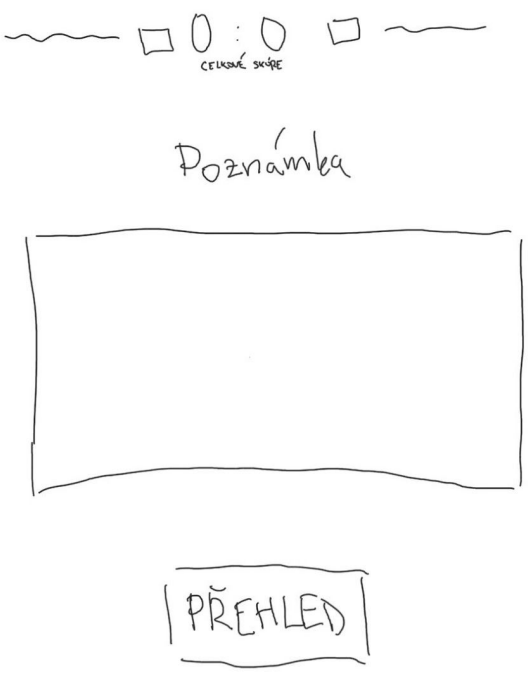
Poté začnou samotné dílčí zápasy. Uživatel nejprve vybere sestavu pro daný zápas a poté bude vyplňovat po setech skóre. Během setu si můžou oba týmy vzít oddechový čas, mohou vystřídat a některý z hráčů může dostat kartu.

zadávání sestavy pro dílčí zápas

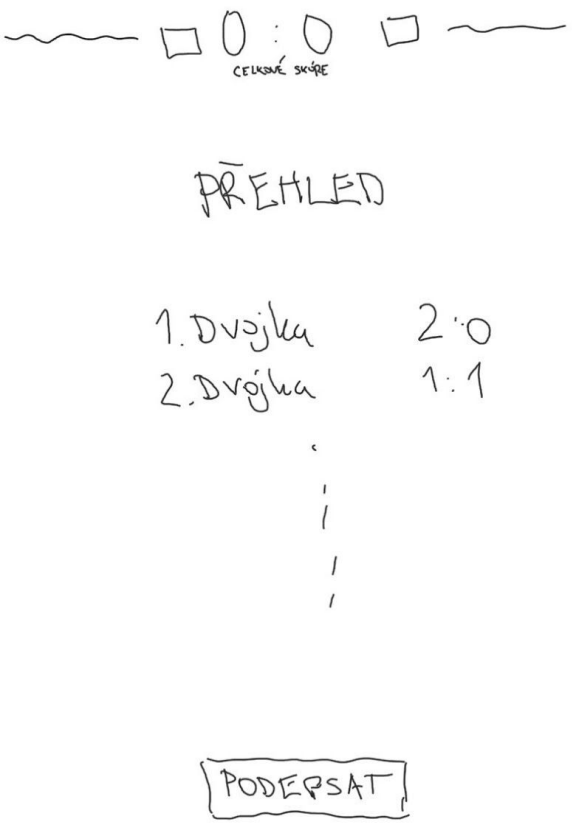


průběh setu dílčího zápasu

Po proběhnutí všech dílčích zápasů může zadávající doplnit poznámku a přesunout se na přehled celého utkání.



poznámka k utkání



přehled celého utkání

Po zkontrolování správnosti údajů z celého zápasu zástupci obou týmů podepíší zápis utkání. Po podepsání se zápas ukončí.

podpis utkání

6 Roadmap

MVP (Minimum Viable Product)

Minimální použitelný produkt by měl umožnit přihlášeným uživatelům zadat zápis z jejich zápasu ve dne jeho konání. U vyplněných dat musí být ověřena jejich správnost a konzistence a musí být uložena do perzistentního úložiště.

Nepřihlášení uživatelé by měli mít možnost zobrazit výsledky zápasů ve všech soutěžích a jejich průběžnou výsledkovou tabulku.

Rozdělení lidí do podčástí aplikace

Backend
- Adam Harmanec
- Tomáš Kukaň

Webový frontend
- Filip Horký
- Tomáš Kukaň

Mobilní aplikace
- Nik Harmanec
- Richard Savčinský

1. Fáze - až 31. březen

Repozitář obsahuje všechny projekty.

Backend
- základní Django aplikace, propojení s DB
- definované DB modely, serializéry
- vývojářský server poskytuje všechny statické soubory a nějaké základní dotazy do DB

Web
- konfigurace pro webpack, babel, eslint a JS prostředí
- kostra aplikace s entrypoint komponentou, routing komponentou, root sagou a root reducerem
- komponenty pro landing page, přihlašovací dialog a dashboard přihlášeného uživatele

Mobilní aplikace
- nadefinování modelů, objektů a struktur
- obrazovky soutěže, zápasů a klubů zobrazující statická data

2. Fáze - až 15. květen

Backend

- všechny potřebné API endpointy definované
- koupený server, ssh přístup, nainstalované potřebné produkční programy: NGINX, Postgres, uWSGI...

Web

- implementace zbývajících komponent

Mobilní aplikace

- přihlášení uživatele
- formulář na zadávání výsledků zápasů a veškerá logika s tím spojená

3. Fáze - až 15. červen

Backend

- definované skripty pro automatické nasazení aplikace
- uživatelské testování
- HTTPS protokol

Web

- uživatelské testování
- vyhlazování UI

Mobilní aplikace

- propojení s backendem a zobrazování dynamických dat
- vyhlazování UI
- testování

Dokumentace

Zdroje

Rozpisy a další Pražské dokumenty zde: https://www.nohejbalpraha.cz/dokumenty/
Pravidla a další celostátní dokumenty zde: http://www.nohejbal.org/text/63-zakladni