Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Implementace webové aplikace pro docházkový systém

Bakalářská práce

Autor práce: David Kalianko
Studijní obor: Aplikovaná Informatika

Vedoucí práce: Ing. Michal Macinka

Hradec Králové 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury

.....

David Kalianko

30. dubna 2021

Anotace

Předmětem práce je popsat vývoj aplikace pro management interních a externích záležitostí v organizaci. První část je zaměřena na teorii a analytiku řešeného problému, ve které jsou popsány aktuální dostupné technologie pro vývoj a popis procesů v organizaci. V druhé části je popsán návrh struktury aplikace a pomocí jakých technologií se implementovala. Další část je zaměřena na ukázku implementace aplikace, kde je popsáno z jakých částí se aplikace skládá a částí, které vyžadovali více pozornosti. V poslední části je zpracováno testování, které bylo použito pro aplikaci. Celá práce je provedena postupy, jak se vytváří nová webová aplikace.

Klíčová slova

webová aplikace, docházka, management organizace, Vue.js, PHP

Anotation

Title: Implementation of a Web Application for Attendance System

The subject of this work is to describe development process of an application for the management of internal and external affairs in an organization. First part is focused on the theory and analytics of problem, which describes current technologies available for the development and description of processes in an organization. Second part describes design of the application structure and technologies used to implement it. Next part is focused on an example of the implementation of the application, where it is described which parts the application consists of and parts that required more attention. Last part deals with testing, which was used for application. All work is carried out procedures for how to create a new web application.

Keywords

web application, attendance, organization management, Vue.js, PHP

Obsah

Seznam obrázků

Seznam tabulek

Seznam kódů

1 Úvod

V dnešní době by chtěl každý vědět o tom, co se děje v jejich organizaci, kdo chodí na kroužky, kdo se dostavuje včas do práce a mít všeobecný přehled o členech, či o zaměstnancích.

Pro organizace je dnes standardem uchovávat si seznam členů nebo docházejících. V menších organizacích se tyto záznamy uchovávají na papír, nebo v excelové tabulce, ke které má přístup jen omezený počet lidí. Ačkoli pro malé organizace jsou tato řešení jednoduchá, levná a záznamů není mnoho, není potřeba hledat jiná řešení. Pokud však má organizace 100 členů a každý rok se mění, mohou být tato řešení nedostačující a nepřehledná. V případě, kdy bude potřeba dohledat pět let starou docházku, je nutné, aby tyto dokumenty byly pohromadě a dobře roztřízené.

Dnes už však systémy pro správu docházky existují a fungují ve velké škále organizací za účelem správy lidských zdrojů, ekonomiky a logistiky. Většina lidí ví, jak jsou majitelé malých podniků zaneprázdněni. Díky takovým systémům jako je docházkový systém, který může pomoci ušetřit čas na administrativních úkolech. Řešení docházky může zlepšit jak efektivitu, tak přesnost při vykazování a správě času. Bohužel tato řešení jsou drahá a podnikatelé musejí být opatrní při jejich výběru a integraci.

Pro zaměstnance není taková výhoda používat tento systém ve firmě, kde majitel vidí, jakou mají pracovní docházku. Proto s tímto systémem ve většině případů nesouhlasí a nechtějí ho aplikovat ve firmě. Také ve většině případů nesplňují požadavky dané organizace při vyšších nárocích na aplikace. Pokud uvedu příklad na organizaci, pro kterou se vyvíjí aplikace v této bakalářské práci, tak zde lze použít několik příkladů, proč standardní řešení docházkového systému není dostačující. Například organizace chce zaznamenávat i akce, které v organizaci probíhají a plánovat tyto akce společně s docházkou. V tomto případě by organizace musela najít vhodné řešení, které nebývá levné, nebo bude zapotřebí využít dva systémy pro správu dalších akcí v organizaci.

Cíl práce

Cílem práce je vytvořit aplikaci pro management a kontrolu docházky pro organizaci Hybatelna z.s., která se zaměřuje na volnočasové aktivity. Pořádá tréninky parkouru, workoutu, akce pro veřejnost, umožňuje pronájem haly a další aktivity. Pro tyto aktivity je třeba vytvořit systém, který bude obsahovat přehled všech docházejících, uživatelů, akcí, o aktivitách v organizacích atd. Aplikace bude vytvořena na míru pro Hybatelnu a bude obsahovat pouze funkcionality, které využije. Měla by nahrazovat aktuální registrační formulář a správu docházky v excelových tabulkách a umožnit tyto funkcionality i externím organizacím.

2 Rešerše docházkových systémů

Docházkové systémy ulehčují práci hlavně při zaznamenávání prezence, kdy není potřeba zadávat vše ručně, ale stačí pouze načíst čip a dle identifikátoru automaticky doplní příchod a odchod. Pro firmy je to výhodné, jelikož mají přehled o tom, jak se lidi pohybují ve firmě, a zjednodušuje počítání zdrojů. Docházkové systémy pomáhají firmám zajistit dodržování předpisů od průmyslově stanovených požadavků na plánování až po zásady pracovní doby.

Systém umožňuje zaměstnavateli sledovat zaměstnance, zda přijeli včas, nebo naopak pozdě, či sledovat čas strávený pauzami. Dále pomáhá kontrolovat finanční náklady snížením nadměrných plateb, které jsou často způsobeny placením zaměstnanců za čas, po který nepracují. Tyto výhody pro zaměstnavatele jsou hlavní důvody, proč zaměstnanci nesouhlasí s integrací docházkového systému.

Jedním z typů je i Time-tracking. To je software pro sledování času, který zaměstnancům dovoluje zaznamenávat čas na úkolech nebo projektech. Tento software se používá ve většině průmyslových odvětví a někde nahrazuje docházkový systém jako takový. Toto řešení často využívají živnostníci, jako jsou právníci, účetní a ve velké míře vývojáři softwaru, kteří si účtují práci za hodinu. Tento systém však zde nebude rozebrán, protože neodpovídá požadavkům implementované aplikace.

Docházkové systémy jsou řazeny do několika typů podle toho, jak je docházka zpracována, nebo zaznamenávána [7]:

- Manuální systémy - tyto systémy jsou prvními systémy pro docházku, dříve se používaly papírové karty s časovými razítky. Dnes se manuální systémy stále používají, ale ve formě zapisování do excelových tabulek, knih atd.
- Automatizované systémy - systémy vyžadují, aby se zaměstnanci identifikovali pomocí nějaké metody a zaznamenali svůj příchod nebo odchod. Zaznamenání většinou spočívá v používání elektronického štítku RFID nebo čárového kódu, ty byly nahrazeny biometrickými údaji a zařízeními s dotykovou obrazovkou. Tyto systémy jsou přesnější, vyžadují méně práce a umožňují další funkcionality oproti manuálním systémům.
- Biometrické systémy - zaznamenávají příchod nebo odchod podle biometrických údajů, jako je otisk prstu, obličej, hlas nebo podpis.

Automatizovaná řešení pro čas a docházku dávají manažerům možnost jednodušeji spravovat pracovní vytížení na projektech. Toto může zvýšit efektivitu, splnit cíle a lépe řídit náklady. Aplikace, která zobrazuje přehled časů, umožňuje majitelům malých podniků kontrolovat data v průběhu času, sledovat příležitosti, jak být efektivnější, a řešit problémy, které mohou zhoršovat produktivitu.

Plánování a vyřizování žádostí o volno je důležitou součástí udržování spokojených zaměstnanců. S automatizovanými systémy je možné rychle kontrolovat, schvalovat nebo řešit žádosti o nepřítomnost a spravovat plánované nepřítomnosti.

2.1 Rešerše funkcionalit

Pro organizaci je potřeba najít funkcionality, dle kterých se budou vyhledávat existující systémy, které je možno využít pro organizaci. Jako docházkový systém lze využít téměř vše, ale některá řešení nemusejí být vhodná a bude složité nastavit postupy pro dané funkcionality. Prvně by bylo vhodné zanalyzovat, jaké funkcionality organizace potřebuje.

První funkcionalitou je zaznamenávání docházky, které by mělo být manuální a automatické pomocí identifikační karty. Klient již vlastní systém pro správu karet, který by rád využil i v procesech docházky. Tento systém umožňuje inicializovat karty a po přiložení je možné spustit skript, který může poslat požadavek na jiný systém. Tato docházka by měla být živě sledovatelná a umožňovala zobrazit přehled všech docházejících na jedné události. Jako další funkcionalita je potřeba umožnit omezený přístup do aplikace externím uživatelům neboli uživatelům, kteří jsou pouze jako docházející, nebo mají pod sebou docházející. Těmto uživatelům bude umožněn přístup do sekcí, které jsou veřejné, a do sekcí, které patří danému uživateli. Jako další funkcionalitou je generovat události na základě předem definovaných dat. Tyto události bude možno vytvářet a editovat a následně bude umožněno přihlášení docházejících na události. Další funkcionalitou pro systém je možnost dodatečných vylepšení nebo úprav. Pro růst organizace je vhodné zvolit takový systém, který umožňuje rozšíření, například v organizaci vznikne nový typ docházky nebo události, který je zpracováván jinak, nebo bude potřeba jiné zpracování pro již existující události.

2.2 Rešerše existujících systémů

Existující systémy byly hledány dle kritérií zmíněných výše a byly zde vybrány takové, co nejvíce odpovídají potřebám organizace. Je zde vhodné zmínit, že tato kritéria jsou potřebná a je nutné je v systému mít. Některé systémy, pak mohou mít funkcionality, které by se také hodily do systému, proto zde mohou být zmíněny i ty, které úplně neodpovídají daným kritériím.

TSheets

Tento systém od firmy QuickBooks umožňuje zaznamenávat docházku pomocí geofencingu. Pokud se pracovník nenachází v okruhu sítě, není mu počítán čas, který nestrávil prací. Pokud však majitel nechce zaznamenávat docházku pomocí geofencingu, lze využít TSheets Clock Kiosk, kterým je možno měřit pouze čas, který zaměstnanec stráví prací. TSheets nabízí správu a plánování směn zaměstnancům, což může být výhoda pro vícesměnné podniky.

QuickBooks jako název firmy je účetní systém, který lze napojit na TSheets. Řešení pak usnadňuje fakturování zaměstnanců nebo vyplácení jejich mzdy a zaevidování nákladů, které vznikly za dané období. TSheets nabízí dvě varianty. Jedna je Premium, která stojí 20 $ plus poplatek za každého evidovaného uživatele. Zde nabízí základní funkce bez

geofencingu a bez časového rozvrhu. Další funkce nabízí varianta Elite, která stojí 40 $ plus poplatek za každého evidovaného uživatele [8].

Alveno

Alveno vznikl českou firmou Iresoft a je založen na cloudových technologiích. Umožňuje přístup ze všech zařízení a nabízí mimo základní funkce docházkového systému i evidence žádanek na dovolenou, nároky na stravenky, rozpis směn, sledování přítomnosti zaměstnanců nebo čtečku s potvrzením pomocí GPS lokace a další. Je dobrým pomocníkem při vykazování práce a přestávek zaměstnanců podle české legislativy a je vhodný jak pro malé, tak i velké firmy s jednoduchým řešením pro správu zaměstnanců. Cenová škála se pohybuje od 2000 Kč (do 50 zaměstnanců) až 5500 Kč (do 200 zaměstnanců) a ceny jsou počítány za měsíc [9].

Aktion

Aktion nabízí různé produkty a jedním z nich je docházkový systém. Ten umožňuje spravovat docházku pomocí webové nebo mobilní aplikace a díky cloudovým technologiím lze sledovat docházku odkudkoli. Zaznamenávat docházku lze pomocí otisku prstu, NFC karty, čipu nebo manuálně. Fungují zde omluvenky, export dat, rozdělení uživatelů dle práv a přístupu, plánování schůzek atd. Cenová škála od 1 do 5 zaměstnanců je Aktion zdarma, od 5 zaměstnanců cena začíná na 290 Kč a za každého dalšího zaměstnance zhruba 10 Kč za měsíc [10].

Bakaláři

Bakaláři se využívají ve školním prostředí a nabízejí spoustu služeb, které usnadňují administrativu. Takový systém je přímo zaměřený pro školy, tudíž je složité ho integrovat v jiných odvětvích, ale je vhodné ho zmínit pro jeho funkcionality, kterými je možno se inspirovat.

Bakaláři umožňují evidovat docházku žáků i zaměstnanců, který splňuje požadavky MŠMT. Dále nabízí možnost udělat si žákovskou knížku, kde se zapisují známky a celkový prospěch žáka ve škole, možnost plánování rozvrhu, plánování suplování, rozpisy důležitých termínů, jako jsou maturitní zkoušky. Díky aplikaci, kterou Bakaláři nabízejí, můžou rodiče studentů kontrolovat docházku a prospěch svého dítěte ve škole. Jako další výhodou jsou moduly, které se dají do systému nahrát a umožnit další funkcionality pro systém. Moduly lze i vytvářet a spolupracovat na vývoji systému.[11]

2.3 Srovnání

Dle tabulky 1 je vidět, že nejlépe využitelný systém pro organizaci jsou Bakáláři. Ty obsahují možnosti, které jsou použitelné i pro tréninkovou činnost. U Bakalářů však chybí identifikace pomocí karty a obsahuje pouze manuální zadávání docházky. Sice pro Bakaláře

Tabulka 1: Přehledová tabulka funkcionalit docházkových systémů

Funkcionalita/Systém	Tsheets	Alveno	Aktion	Bakaláři
Zaznamenání docházky pomocí karty	Ano	Ano	Ano	Ne
Přehled docházky	Ano	Ano	Ano	Ano
Rozdělení přístupů	Ano	Ano	Ano	Ano
Dostupnost odkudkoli	Ano	Ano	Ano	Ano
Možnost úprav	Ne	Ne	Ne	Ano
Kalendář	Ano	Ano	Ano	Ano
Události	Ne	Ne	Ano	Ano

je možné vytvářet pluginy, ale tento proces je zdlouhavý a musí být plugin schválen. Ostatní systémy jsou spíše použitelné pro firemní prostředí, kdy je potřeba vypočítávat mzdu nebo zaznamenávat pracovní nasazení zaměstnanců.

Tyto systémy by však bylo možné použít pro účely tréninkové činnosti, ale neodpovídaly by dané infrastruktuře a neobsahovaly by veškeré funkce, které vyžaduje organizace. Také by některé funkcionality nebyly vhodné pro nasazení v tomto prostředí a zbytečně by zvyšovaly náklady.

3 Analýza procesů

V této kapitole budou probrány procesy, které probíhají okolo tréninkové činnosti. Na základě těchto procesů je potřeba najít místa, kde je možné činnost nebo proces zautomatizovat a ulehčit práci organizaci. Většina procesů lze zautomatizovat a předat z člověka na stroj, ale v některých případech vznikají nové procesy, které po integraci systému mohou být náročnější jak časově, tak provedením. Je proto třeba ještě zhodnotit, zda aktuální chování pro daný proces zanechat.

3.1 Analýza základních procesů

Procesy organizace nejsou složité a rozvětvené, proto analýza proběhne v rámci uživatelských příběhů, kdy uživatel řekne, co by chtěl jako funkcionalitu. Tyto příběhy vycházejí z aktuální zkušenosti uživatelů, které tyto procesy provádí a chtěli by je trochu zautomatizovat. Pro jeden příklad lze uvést zapisování docházky. Pokud akce nebo trénink má hodinu, je potřeba docházku udělat rychle, aby zbylo co nejvíce času na zbytek tréninku. Pokud se odmyslí část s přípravou a zaznamenáním docházky, lze ušetřit až pět minut z tréninku, což by mohlo vytvořit pozitivní vliv na členy a prodloužit čas, který je věnován tréninkové činnosti. Dále zde budou zahrnuty části, které ulehčí práci jak pro veřejnost, tak pro interní záležitosti.

3.2 Uživatelské příběhy

Uživatelské příběhy neboli user stories popisují příběh uživatele. Ten má za cíl upřesnit a udělat si představu o funkcionalitě, která má být implementována. Většinou uživatel nebo klient, který chce novou funkcionalitu, musí ji někomu zadat. Popsáním user story si dokáže více lidí udělat lepší obrázek o funkcionalitě a je lépe pochopitelný pro následující implementaci. Pro lepší představu, jak se mají vytvářet user stories, viz 2. Na základě výpovědí od klienta bylo vytvořeno několik user stories. Před implementací aplikace je třeba promyslet, jaké tyto user stories budou. Pro zachycení myšlenky těchto user stories zde budou uvedeny dva příklady:

- Vytvořit registrační formulář pro uživatele.
- Jako uživatel bych se rád registroval, abych se mohl přihlásit na tréninky.

Obě zadání říkají to samé, má být vytvořen registrační formulář. V některých systémech je dostačující první zadání, jelikož systém bude mít pouze jeden typ uživatele. V tomto systému však bude více typů uživatelů a tedy druhé zadání říká, že registrační formulář, který má být vytvořen, je pro externího uživatele. V prvním příkladu je možno zaměnit s interními uživateli. Při nedostatečné informovanosti programátora by mohl vytvořit formulář, který není vyžadován, a tím by mohl ztratit čas. Proto v této části bude uvedeno několik základních user stories pro lepší pochopení systému. Pro uživatele je třeba zvážit několik

možností řešení pro daný úkon. Tyto možnosti, které zde budou rozebrány jako návrh, budou klientovi představeny a bude uvedeno řešení v dané sekci.

Tabulka 2: Vzor pro vytváření user stories [3]

Pro koho je funkcionalita určena?	Jako typ uživatele
Co se bude implementovat?	Já chci něco
Proč se to bude implementovat?	Kvůli potřebě nebo hodnotě

3.2.1 Přihlášení docházejícího

User Story Jako rodič bych rád přihlásil své dítě na trénink, aby mohlo docházet na tyto tréninky.

Aktuální proces Uživatel na webových stránkách vyplní formulář pro registraci docházejícího. Následně se v databázi vytvoří docházející a je třeba docházejícího zařadit dle skupiny a vložit do excelové tabulky.

Návrh První možností je vytvořit pro každého docházejícího účet v systému. Tento scénář je nejméně náročný, ale neobsahuje všechny případy užití. Například rodič má dvě děti a ty chce přihlásit na tréninky. Pro každé dítě bude v systému vytvořen účet se stejným rodičem a kontaktními informacemi, tudíž by se rodič musel přihlašovat pomocí unikátního jména a pro každý účet si toto jméno a heslo pamatovat. Tento scénář odpovídá tomu, pokud daný rodič má jedno dítě nebo sám registrující uživatel bude docházejícím. Scénář je inspirovaný systémem Bakaláři, kdy každý student dostane účet pro přihlášení. Rodič, pokud mu je umožněn přístup, bude mít dva uživatele, na které se může přihlásit.

Druhou možností je registrace jednoho uživatele jako rodiče. Tento rodič bude zaregistrován do systému a bude mu umožněno přihlásit si pod sebe docházející. Tedy pro dvě děti bude pouze jeden účet. S touto možností se lze zaregistrovat jako rodič a být docházejícím. Uživateli bude vytvořen jeden účet, pod kterým může spravovat vše a není nutnost, v případě dvou dětí, spravovat více účtů.

Třetí možnost je blíže k první, ale bez nutnosti vytvářet účet. Rodič přihlásí dítě na trénink a žádný účet nebude vytvořen, pouze bude dítě zapsáno na tréninky. Přístup k docházce bude umožněn přes odkaz, který bude zaslán na e-mail po vyžádání.

Řešení Pro přihlášení se rozhodlo využití jednoho účtu pro přístup do systému a následná správa všech docházejících pod jedním účtem. Tímto rozhodnutím přibyly nová user stories, které je potřeba zohlednit při vývoji aplikace. Například docházející už je plnoletý a chce se přihlašovat pod vlastním účtem. Tady je potřeba zvážit možnost, zda se docházející převede na tento účet nebo vytvoří znovu.

3.2.2 Vytvoření docházky

User Story Jako manažer bych chtěl vytvářet tréninky s docházkou, aby se mohli docházející přihlásit. Chtěl bych vygenerovat docházku na základě daných kritérií.

Aktuální proces Do tabulky se vypíšou termíny, kdy se daný trénink koná. Pokud nastanou školní prázdniny nebo zrušení tréninku v daném termínu, nejsou tyto termíny označeny dopředu a při zjištění je potřeba projít všechny excelové tabulky, doplnit informace o této situaci a informovat všechny uživatele.

Návrh Ke všem řešením je potřeba vytvořit formulář pro vytvoření tréninku. Tento formulář bude obsahovat kritéria jako den v týdnu, období, kdy budou tréninky probíhat, čas tréninku. Tyto informace jsou potřebné k vygenerování docházky. Další informace o tréninku mohou být doplněny o další atributy nedůležité pro generování jako popis, jméno zodpovědné osoby atd. Manipulace s tréninky a docházkou bude umožněna pouze manažerovi. První možností bude na základě výše zmíněných kritérií vygenerovat docházku a uložit. Pro potřeby editace bude možno doplnit, odstranit nebo popřípadě editovat termíny v detailu tréninku. Další možností je vygenerovat docházku a před uložením ji editovat. Manažer bude moci přidat další termíny akce nebo některé odebrat. Pokud je potřeba upravit termíny v průběhu období, bude k tomu sloužit kalendář, který bude zobrazovat aktivity v daném termínu. Pokud se stane, že v daném termínu odpadnou tréninky, bude možné změnu označit pouze na jednom místě, a tím se změna provede u všech tréninků v daném dni.

Řešení Pro vytvoření docházky se zvolila možnost editace termínů při vytváření docházky. Klient se rozhodl, že chce mít možnost upravovat jednotlivé akce nebo tréninky v kalendáři. Proto odpadá nutnost vytvářet editaci termínů dané docházky.

3.2.3 Proces s kartami

User Story Jako trenér bych chtěl mít možnost přiřadit danému docházejícímu identifikační kartu, podle které bude zaznamenávána docházka.

Aktuální proces Tento proces vznikne po integraci aplikace.

Návrh Trenér vezme kartu a přiloží ji k terminálu. V systému karet vyplní údaje jako jméno, příjmení a identifikační číslo. Této kartě je pak vytvořen identifikátor, podle kterého se bude prokazovat. Tento identifikátor bude potřeba uložit v systému docházky k danému docházejícímu. Zde jsou tři možnosti. Manuální, kdy manažer bude muset najít uživatele, kterému tato karta patří a přiřadit ji k němu. Tato možnost bude výchozí a v systému implementována. Další možností je exportování všech karet ze systému karet a namapování na docházející v databázi. Tato možnost vyžaduje mít jednoznačný identifikátor docházejícího v databázi a v systému karet. Další možností je vytvořit v systému sekci, která bude kontrolovat přiložení karet. V případě, kdy se karta přiloží, najde se v databázi daný docházející, kterému tato karta patří a zobrazí ho. Pokud karta nikomu nepatří, bude možnost tuto kartu přiřadit docházejícímu. Poté bude potřeba vyexportovat uživatele a nahrát do systému karet.

Řešení Pro proces s kartami bylo vybráno pouze manuální zadávání karet. Klient má zaměstnance, který bude mít toto na starosti. Není tedy potřeba implementovat rozsáhlé automatické řešení, ale pouze implementovat možnost a přiřadit kartu na docházejícího.

3.2.4 Zaznamenání docházky

User Story Jako docházející chci mít možnost zaznamenat svoji docházku. Pokud zapomenu kartu doma, rád bych byl označen v docházce, že jsem na tréninku byl.

Aktuální proces Na začátku každého tréninku se všichni docházející dostaví na předem dané místo. Trenér si připraví docházku a postupně ji vyplňuje, zda daný docházející je přítomen.

Návrh Docházející přijde na trénink a přiloží si svoji kartu k terminálu. Systém najde uživatele a zjistí, jaké má tréninky v daný den. Pokud má jen jeden trénink, jeho přítomnost bude označena v docházce. Pokud má více tréninků za celý den, je třeba promyslet, zda jedno přiložení karty znamená, že se účastnil nejbližšího možného tréninku nebo označí v tento den všechny tréninky, na které je přihlášen. Pokud by měl dva tréninky za sebou, tak v případě označení pouze prvního nejbližšího bude si muset na začátku druhého tréninku dojít pro svoji kartu a přiložit ji znovu. V případě označení všech nejbližších může znamenat, že mu bude označen trénink, na kterém nebyl nebo nebude. Tímto vznikne chyba v zaznamenání a rodič nemusí vědět o tom, že se dítě neúčastnilo tréninku. Proto je potřeba zvážit, které z těchto řešení bude vhodné pro danou situaci. Sám trenér pak bude moci upravovat docházku v případě, že si docházející zapomněl kartu.

Řešení Pro zaznamenání docházky se rozhodlo, že docházející bude muset přiložit kartu před každou událostí. Jelikož se nevyskytuje mnoho takových případů, že by v jeden den měli více naplánovaných událostí.

3.3 Požadavky klienta

Při schůzce s klientem bylo sepsáno několik požadavků, které by chtěl mít v systému. Prvním požadavkem je registrace a přihlášení přes služby třetích stran, jako je Google nebo Facebook. Pro uživatele, kteří nemají tyto služby, bude možnost registrace a přihlášení přes e-mail. Dalším z požadavků je vytvořit frontend za pomoci Material Designu a předem vybrané šablony pro zachování stylu s podobnou aplikací, kterou klient vlastní. Klient bude chtít v budoucnu aplikaci rozšiřovat o nové funkcionality, jako například rozdělení entit do organizací, evidovat přijaté platby, generovat tabulky, integrovat notifikace a vytvořit e-mailovou komunikaci skrze aplikaci. Posledním zde zmíněným požadavkem je osobní kalendář. Každý přihlášený uživatel bude mít možnost do kalendáře přidat své osobní akce nebo události, které uvidí pouze on sám. V roli manažera, může vytvořit událost, která bude viditelná pro danou skupinu uživatelů.

4 Analýza technologií pro vývoj webových aplikací

Každým rokem přibývá spousta technologií pro vývoj aplikací. Dříve nebyl takový výběr, muselo se pracovat s tím, co bylo a možnost výběru technologie nebyla takovým trendem jako jím je dnes. Tyto technologie jsou ve většině případů určeny k danému použití. Integrace je sice možná, ale vlastnosti technologie nejsou vhodné pro daný typ. Pro představu Node.js je vhodný pro použití s aplikacemi v reálném čase, ale není vhodný pro náročné výpočty.[12]

4.1 Databázová část

Pro každý systém, který uchovává data je potřeba mít nějakou formu úložiště. Těchto úložišť existuje několik a pro tuto aplikaci bude využit databázový systém. Tyto systémy slouží k ukládání dat a používají se pro správu dat. Pro tyto systémy je čtení jednou z důležitých vlastností databáze, kdy je potřeba číst data podle zadaných parametrů nebo optimalizace v případě tisíce řádků.

Databázové systémy lze rozdělit podle základních typů ukládání dat a jejich vazeb. Těmito typy jsou hierarchická, síťová, relační, objektová a objektově relační databáze. Tato analýza se bude zabývat pouze objektovou, relační a NoSQL databází, ostatní typy jsou zastaralé a dnes se nepoužívají, proto nebudou zmíněny. Je vhodné zmínit i rozdělení podle CAP teorému, ten umožňuje definovat databázi podle třech hlavních vlastností.

Nedílnou součástí databází jsou transakce. Transakce jsou jakékoli operace nebo skupiny operací prováděné v databázi, například vytvoření nového záznamu nebo editace existujícího záznamu. Tyto operace převádí databázi z jednoho konzistentního stavu do druhého. Tyto transakce jsou řízeny podle pravidel ACID.

4.1.1 ACID

ACID je zkratkou čtyř základních pravidel databázových transakcí, které zajišťují spolehlivé zpracování, aby nedošlo k poškození dat uvnitř. Tato pravidla jsou atomicita, konzistence, izolace a trvanlivost. Použitím vlastností ACID na každou úpravu databáze je nejlepší způsob,jak zachovat přesnost a spolehlivost databáze.

Atomicita (Atomicity)

Atomicita transakce zajišťuje, že jakákoli transakce nebo operace, která bude provedena, úspěšně dokončí celou operaci nebo v případě ztráty spojení či chyby v průběhu se databáze vrátí zpět do konzistentního stavu, ve kterém byla před zahájením transakce.[13]

Konzistence (Consistency)

Konzistence označuje zachování integrity dat. Konzistentní transakce neporuší omezení integrity na datech, která jsou definována v databázi. Vynucení tohoto pravidla zajistí, že pokud se databáze dostane do nekonzistentního stavu, nebo dojde-li k narušení omezení

integrity dat, proces bude zrušen a provedené změny budou vráceny zpět do původního konzistentního stavu.[13]

Izolace (Isolation)

Izolované transakce jsou považovány za „serializovatelné“, což znamená, že každá transakce probíhá v odlišném pořadí, aniž by k transakcím došlo najednou. Jakékoli transakce prováděné v databázi nebudou ovlivněny jinými transakcemi, ke kterým dochází v jedné chvíli ve stejné databázi. Každá transakce je zařazena do fronty, aby se zajistilo, kompletní dokončení všech transakcí, než začne další transakce. Toto však neznamená, že dvě transakce nemohou proběhnout najednou. V jednu chvíli může proběhnout více transakcí, které se navzájem neovlivňují a nemohou narušit konzistenci dat. Toto pravidlo má dopad na rychlost prováděných transakcí, kdy některé transakce musejí čekat na dokončení jiných transakcí.[13]

Trvanlivost (Durability)

Trvanlivost zajišťuje, že změny provedené v databázi, které se úspěšně dokončí, jsou zapsány na disk a uloženy v energeticky nezávislé paměti. Úspěšně dokončené transakce pak přetrvávají v databázi i v případě selhání systému. Tímto pravidlem je zajištěno, že data v databázi nebudou ztracena nebo poškozena výpadky. [13]

4.1.2 CAP teorém

CAP teorém, který je znám i jako Brewerův teorém říká, že může poskytnout dvě ze tří vlastností databázového systému. Těmito vlastnostmi jsou konzistence (Consistency), dostupnost (Availability) a tolerance oddílů (Partition Tolerance). Z těchto vlastností je vytvořena kombinace dvou, která definuje databázi.

Konzistence

Pojmem konzistence se zde nemyslí u ACID transakcí, zde tento pojem znamená, že každý kdo v jednu chvíli zažádá o data, vidí vždy stejná data jako ostatní. Jinými slovy, pokud je prováděna operace čtení po více zápisech, pak by konzistentní systém měl vrátit stejnou hodnotu pro všechna čtení a při zápisu by měl pracovat s nejnovějšími daty.[14]

Dostupnost

Dostupný systém zůstává funkční po celou dobu běhu. To znamená, že každá žádost, musí být zpracována a musí vrátit výsledek, který nebude obsahovat chybu. Dostupný systém nemusí být konzistentní, ale musí vždy vracet data, i když nebudou aktuální a musí být neustále k dispozici pro všechny žádosti nebo pro všechny zápisy. Tato vlastnost může mít negativní vliv na data, kdy se mohou editovat záznamy, které jsou v danou chvíli editovány a mohou tak vznikat nepřesné a nekonzistentní záznamy. [14]

Obrázek 1: Vizualizace CAP teorému [1]

Tolerance oddílů

Tato vlastnost říká, že by systém měl pokračovat v chodu, i když dojde k chybě, výpadku komunikace nebo ke zpoždění mnoha zpráv v síti mezi oddíly. Pokud vypadne komunikace, která oddíly spojuje, pak nemohou vědět o aktualizacích ostatních oddílů. Pokud je potřeba, aby databázový systém běžel na více serverech, je nutné vybrat databázový systém s touto vlastností. [14]

Kombinace Konzistence a Dostupnost

Tyto systémy se vyznačují vysokou úrovní konzistence a snaží se dosáhnout nejvyšší možné úrovně dostupnosti. Příkladem pro tuto kombinaci jsou relační databáze jako MariaDB, MySQL, Postgres atd.[15]

Kombinace Konzistence a Tolerance oddílů

V této kombinaci jsou data konzistentní mezi všemi oddíly a udržují mezi sebou spojení. Pokud nastane nějaká chyba či výpadek, tak tento oddíl nepřijme žádnou žádost nebo požadavek a budou zrušeny místo vracení nekonzistentních dat. Příklady této kombinace jsou např. Google Bigtable, Hbase, MongoDB, MemcacheDB, Redis atd.[14]

Kombinace Dostupnost a Tolerance oddílů

S kombinací těchto vlastností jsou systémy vysoce dostupné a odolné vůči přerušení. To znamená, že každý požadavek na čtení nezaručuje nejnovější informace, ale bude zpracován a vrátí data bez ohledu na to, jestli jsou zastaralá. Příklady této kombinace jsou např.: Voldemort, SimpleDB, CouchDB atd.[14]

4.1.3 Relační databáze

Jeden z typů je relační databáze, která vznikla v 70. letech a definoval ji Edgar Frank Codd [16]. Poskytují efektivní a flexibilní způsob ukládání a přístup k datům, která jsou strukturovaně poskládaná. Těmito strukturami jsou tabulky, které mohou být propojeny na základě společných dat.Tabulkám se také říká relace a skládají se ze sloupců, které mají přesně definovaný typ dat, neboli datový typ, který popisuje, co tento sloupec obsahuje za typ dat. Tímto je splněna datová integrita, kdy při vložení nového záznamu nelze vložit libovolnou hodnotu a datový typ hodnoty se musí shodovat s datovým typem sloupce, jinak vyhodí chybu. Jednou z dalších struktur jsou řádky, které znázorňují záznamy v databázi. Aplikace, které čtou data z databáze, pak přistupují k těmto datům pomocí dotazů. Nejčastěji jsou dotazy psány pomocí SQL, který je dnes standardem pro psaní dotazu v relačních databázích. SQL je strukturovaný dotazovací jazyk a dovoluje v systému provádět interaktivní dotazy a pomocí tohoto jazyku spravovat databázi.[17]

Relační databáze poskytují prostředí, ze kterého lze k datům přistupovat, nebo je znovu sestavovat různými způsoby. Každá tabulka má jedinečný identifikátor nebo primární klíč, který lze vygenerovat, nebo zvolit jeden unikátní sloupec či více sloupců, které pak budou odkazovat na daný záznam. Každý řádek obsahuje jedinečnou instanci dat a díky primárnímu klíči se nemohou vyskytovat identická data. Logické spojení neboli vztahy mezi různými tabulkami lze poté navázat pomocí cizích klíčů, tento cizí klíč je primárním klíčem jednoho záznamu, který je připojen.

Jednou z hlavních výhod relační databáze je tabulková struktura. Tabulky lze snadno ukládat a lze je kategorizovat, dotazovat se na ně pomocí výše zmíněných dotazů a filtrovat data bez nutnosti reorganizace databázových tabulek. Další výhodou je jednoduchost skládání dotazů pomocí SQL, které není těžké pro naučení a je použitelné pro téměř každou situaci. V relační databázi je možnost zabezpečení dat. Tedy přístup není umožněn všem uživatelům, tudíž jsou data lépe zabezpečena.

MySQL

MySQL vznikla v roce 1995 švédskou firmou MySQL AB[18] v návaznosti na zpopulárnění databází jako Microsoft SQL server a OracleDB. MySQL je malou a lehkou databází, která běží na mnoha prostředích jako je Windows, Unix, Linux, macOS atd. MySQL má dvě licence, jednou z nich je GPL (GNU Public License) pro volné použití. Pro komerční použití vyžaduje firemní licenci.

MySQL má několik výhod. Jednou z výhod je podpora více programovacích jazyků jako PHP, které je často spojováno s MySQL databází, Java, Python, C++, Ruby a další. Jelikož je MySQL jednodušší a menší než většina relačních databází, umožňuje rychlejší provádění operací speciálně pro čtení dat. Velkou výhodou je takzvaně Cloud-ready, která umožňuje její nasazení téměř kdekoli. [19]

MySQL se stala jednou z nejpopulárnějších databází na trhu a pro její rychlost a jednoduchost je využívána velkými firmami jako jsou Twitter, Netflix, Uber atd. MySQL se stala i součástí LAMP stacku [20], který je používán pro mnoho webových aplikací.

MariaDB

MariaDB je forkem MySQL, který odstartoval v době, kdy Oracle koupil MySQL[21]. Vývojáři MySQL měli strach, že bude zpoplatněna, nebo že na ní přestanou pracovat. Proto se vývojáři rozhodli pracovat na nové databázi, která bude fungovat stejně jako MySQL a bude pod licencí Open Source GPL.

MariaDB nabízí stejný výkon, mnohdy je i výkonnější než MySQL. Na druhou stranu nepodporuje platformy, jako jsou Solaris nebo FreeBSD, ale s technologiemi jako Docker lze spustit i na těchto platformách. MariaDB má úplnou kompatibilitu s MySQL, lze přejít mezi databázemi bez větších problémů. Data jsou ukládána stejně a dotazy v MariaDB jsou stejné jako v MySQL. Pro zachování všech funkcionalit, které má MySQL, se každý měsíc slučuje MySQL do MariaDB [19]. Díky GPL licenci se MariaDB používá v mnoha projektech a je vhodnější i pro malé projekty, které nemají mnoho financí.

4.1.4 Objektové databáze

Objektově orientovaná databáze je založená na objektově orientovaném programování neboli OOP. Objektové databáze se běžně používají v aplikacích, které vyžadují vysoký výkon, výpočty a rychlejší výsledky. Některé běžné aplikace, které používají tyto databáze, jsou systémy v reálném čase. Aplikace pro architekty, aplikace pro 3D modelování a vědecké aplikace, které pracují s velkým množstvím dat a díky rychlosti výpočtů jsou vhodné pro tyto aplikace.

V objektově orientované databázi je velkou výhodou práce s objekty. Při uložení objektu do této databáze, je takto vytvořený záznam strukturově stejný, jako byl vytvořen v aplikaci. Stejná struktura je vrácena i při načtení objektu z databáze, kdy vrátí stejný objekt bez potřeby mapování dat a umožňuje rovnou s tímto objektem pracovat. Jednou z výhod databáze objektů je trvalá perzistence, tedy při práci s objekty se data neztrácejí a díky tomuto jsou tyto systémy odolné.

Nevýhodou těchto systému jsou problémy s výkonem při rostoucí komplexitě, náročnost čtení, kdy při vrácení objektů vrací i zanořené objekty. Tento problém se však může řešit hloubkou zanoření dotazů a vracet data, která neobsahují přebytečné objekty. Podobně tomu tak funguje i při editaci, kdy je editován celý objekt místo jen jedné části jako u relační databáze.[22]

Objecitivity/DB

Objectivity/DB je vysoce výkonná a škálovatelná objektová databáze. Je vhodná pro zpracování složitých dat, kde objekty mají mezi sebou mnoho typů spojení. Běží na 32

i 64 bitových procesorech se systémy Linux, macOS, UNIX nebo Windows. Pro vývoj jsou k dispozici C++, C#, Java a Python API. Objectivity/DB má výhodu přistupování k serverům, kde uživatelé mohou k těmto serverům přistupovat transparentně. Objectivity/DB používá pluginy a umožňují přizpůsobit aplikace daným potřebám. Pluginů existuje mnoho jak pro bezpečnost, dotazování, nebo možnost přizpůsobení serverům a jejich souborovým systémům. Všechny platformy a jazykové kombinace jsou interoperabilní. Například objekty uložené programem používajícím C++ v systému Linux lze číst programem C# v systému Windows a programem Java v systému macOS. [23]

4.1.5 NoSQL databáze

Jedním z dnešních trendů je použití NoSQL, zde by bylo vhodné zmínit, že tento typ databází vznikal už v 60. letech, ale označení NoSQL dostalo až v 21. století. Dnes je velké množství těchto databází, protože každá NoSQL databáze se liší vhodností použití pro dané účely. Data jsou zde modelována jiným způsobem než tabulkami, které jsou používány v relačních databázích, viz 3 a díky tomuto jsou některé operace v NoSQL rychlejší. Používané datové struktury jsou považovány za flexibilnější, jelikož se může měnit struktura dat téměř kdykoli.

Z velké části jsou NoSQL databáze používány ve webových aplikacích a aplikacích, které zpracovávají velká data v reálném čase. Jednou z výhod je jejich jednoduchost a horizontální škálování. Mnoho z těchto systémů upozaďuje konzistenci ve prospěch dostupnosti a tolerance oddílů. Výhodou je automatická replikace dat, která v případě selhání vrací data zpět do konzistentního stavu. Při použití NoSQL nastává spoustu nevýhod, které jsou však v některých aplikacích zanedbatelné a jsou soustředěny spíše na jiné výhody. Jednou z těchto nevýhod jsou transakce, které postrádají reálné ACID. Kvůli této nevýhodě nejsou data okamžitě vrácena aktualizovaná, nebo jsou vrácena, ale nejsou přesná a některé z NoSQL systémů mohou data i ztrácet. Za tuto nevýhodu však nabízejí koncepci případné konzistence, ve které jsou změny databáze šířeny do všech oddílů. NoSQL databáze jsou Open Source systémy a zde tkví nevýhoda ve standardizaci, která zatím neexistuje a struktura databáze může v systémech vypadat jakkoli. [24]

NoSQL databáze lze rozdělit na několik typů. Dokumentové, které místo řádku v relační databázi používají dokument konkrétně, JSON, BSON nebo XML a využívá relační a objektové typování. Dokumenty mohou být vnořovány a tvořit vztahy jako v relační databázi. Tento typ je vhodný pro dynamicky se měnící data a díky této vlastnosti se čas implementace zkracuje. Proto jsou NoSQL vhodná pro rychlou změnu struktury dokumentu. Dalším typem je klíč-hodnota, která je nejjednodušším typem NoSQL databáze. Tento typ pokládá důraz na rychlost a práci s daty, která jsou uložena jako klíč (key) a hodnota (value).

Sloupcově orientovaná databáze je organizována jako sada sloupců, to znamená, že v případě, kdy je potřeba číst data na malém počtu sloupců, můžou se tyto sloupce číst přímo, aniž by byla spotřebována paměť nechtěnými daty. Sloupce jsou často stejného typu a těží z

efektivnější komprese, díky níž jsou čtení ještě rychlejší. Sloupcové databáze mohou rychle spojovat hodnoty daného sloupce.

Databáze grafů je zaměřena na vztah mezi datovými prvky. Každý prvek je uložen jako uzel. Mezi jednotlivými prvky se nacházejí odkazy neboli vztahy. Databáze je optimalizována tak, aby zachytila a prohledala spojení mezi datovými prvky. Databáze grafů se obvykle spouštějí spolu s jinými tradičnějšími databázemi.

MongoDB

MongoDB je Open Source NoSQL databáze, která patří mezi dokumentové databáze. MongoDB je vysoce výkonná, škálovatelná s velkými možnostmi zpracování dat. Architektura MongoDB obsahuje dokumenty, které jsou seskupeny do sbírek podle jejich struktury. Tato databáze využívá BSON, to je binární reprezentace JSON a podporuje ukládání dokumentů a výměnu dat.

MongoDB poskytuje velkou kolekci sad replik, kde každá sada může obsahovat více než jednu kopii dat. V sadách replik se všechny primární funkce provádějí na primární sadě, zatímco sekundární sady se používají v případě selhání první sady. MongoDB obsahuje horizontální dělení, které využívá proces změny horizontálního měřítka. Vlastnost vyrovnávání zatížení je provedena tím, že běží na více serverech. Tímto poskytuje duplikaci dat a vyrovnávání zatížení. Na oplátku poskytuje zálohu během selhání hardwaru. Využívá systém souborů mřížky, který rozděluje konkrétní soubor na různé části a ukládá je samostatně.

Návrh modelů je tu jednodušší než u relačních databází. Snižuje potřebu spojení neboli vztahů mezi objekty a poskytuje snadný vývoj schématu. Díky absenci vztahů a transakcí se výkon při dotazu na data zvyšuje. MongoDB používá místo vztahů zanoření dokumentů. Tato vlastnost může být nevýhodou v případě více zanořených dokumentů a vytažení zbytečných dat, jako tomu je u objektových databází. Vytažení dokumentů může být omezeno na atributy, které budou vytahovány společně s dokumentem, nebo úplné omezení vytažení vnořeného dokumentu.

Vlastnost horizontálního dělení MongoDB umožňuje, aby fungovala rychle a efektivně. To je možné díky podpoře horizontálního škálování dat. MongoDB má svůj vlastní dotazovací jazyk s názvem Mongo Query Language (Mongo dotazovací jazyk), který může nahradit SQL.[25]

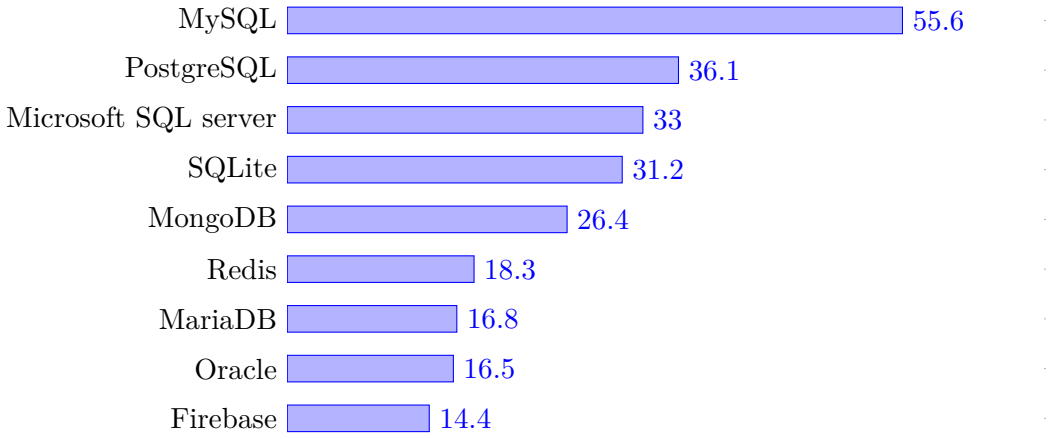
Tabulka 3: Rozdíl ve struktuře MongoDB a Relační databázi

RDBMS	MongoDB
Databáze	Databáze
Tabulka	Kolekce
Řádek	Dokument
Index	Index
JOIN	Vnořený Dokument/Reference

CouchDB

CouchDB je také Open Source NoSQL databáze založená na dokumentech, která se zaměřuje hlavně na snadném použití a je velice spolehlivá z hlediska dat. Díky možnosti spuštění na jakémkoli zařízení Android nebo iOS vyniká CouchDB mimo jiné databáze. CouchDB je kompatibilní s jakoukoli aplikací nebo softwarem, který podporuje formát JSON. Umožňuje uživateli spouštět jednu databázi na mnoha serverech.

CouchDB poskytuje REST API, které se používá k zápisu a dotazování na data. Nabízí čtení, přidávání, úpravy a mazání dokumentů. Poskytuje GUI založené na prohlížeči pro zpracování dat, oprávnění a konfigurace. Usnadňuje autentizaci a podporu relací pro udržení autentizace pomocí cookie relace jako u webových aplikací. Z hlediska zabezpečení poskytuje zabezpečení na úrovni databáze, kde jsou oprávnění na databázi rozdělena na čtenáře a správce.[25]



Graf 1: Nejpopulárnější databázové technologie za Leden 2020 [26]

4.2 Backendová část

Backend nebo BE je vrstva, která pracuje mezi databází a frontendem. Data čte pomocí I/O operací z databáze a zpracovává pro frontend, aby tato data mohl zobrazit. Backend umožňuje provádět operace nad daty a provádět výpočty, které by zatěžovaly frontend. Z backendu jsou pak poslána už zpracovaná data. Toto nejsou jediné možnosti, co nabízí, je možno napojit BE na více databázových systémů, nebo provádět operace v aplikacích třetích stran, odesílat e-maily a jiné.

Na backendové části jsou používány programovací jazyky, které mohou mít několik výhod i nevýhod, například výkon, počet funkcí, možnosti napojení na databázi atd. Různé

programovací jazyky mohou nabízet i rozdílné zpracování dat, jinou strukturu, která je více vhodná pro daný typ aplikace, nebo jednoduchost implementace či naučení daného jazyka. Často je vybírána i dle programovacího paradigmatu neboli stylu programování.

4.2.1 Programovací paradigmata

Existuje několik různých paradigmat a jejich využití pro aplikace daného typu. Programovacích paradigmat existuje více, ale pro účely aplikace není možné ostatní paradigmata využít, proto zde nejsou zmíněna a rozebrána:

- Aspektově orientované programování - Toto paradigma umožňuje rozdělovat program na ucelené funkční části, také jako modularizace [27]. AOP se používá spíše jako doplněk k jiným paradigmatům a je využíváno jazyky jako Java pomocí rozšíření AspectJ, PHP, Javascript, Python a další.
- Deklarativní programování - Skládá se z definic toho „Co se má udělat“. U tohoto paradigmatu se popisuje, co je potřeba udělat od interpretru. Vše ostatní jako postupy nebo použité algoritmy je necháno na interpretru [28]. Tento styl je využit jazyky SQL a Prolog.
- Procedurální programování - Procedurální neboli imperativní programování oproti deklarativnímu popisuje „Jak se to má udělat“. Kód řídí, jak dokončit úkol a je popsán kroky, které míří k dokončení cíle [29]. Toto paradigma využívají jazyky, jako je Java, PHP, C atd.
- Objektově orientované programování - Vytváří sbírku objektů, které obsahují metody a atributy. Tyto objekty pak společně spolupracují na dosažení daného cíle [29]. Pro toto paradigma existuje hodně jazyků, které umožňují použití tohoto paradigmatu jako Java, C++, Python, C#, PHP a další.

4.2.2 Objektově orientované programování

Objektově orientované programování neboli OOP umožňuje tvořit kód tak, jak ho vidí člověk a ne stroj. OOP je abstrakcí světa, jak ho vnímáme. Spolu související funkce a atributy se strukturují do objektů. Objekt je abstrakcí objektu z reálného světa skládající se z funkcí a atributů, které spolu souvisí. Objekty jsou izolovány od dalších objektů, pro které nemají význam. Tomu se říká zapouzdření, které vytváří tzv. blackbox[1] a zpřístupňuje jen rozhraní, přes které probíhá komunikace. Další užitečnou vlastností je dědičnost, kdy od jednoho objektu jsou přenášeny atributy a metody na potomka, který vytváří specializaci předka a je doplněn o další metody nebo atributy. Na tuto vlastnost navazuje polyformismus, který dovoluje v potomkovi přepsat metodu, která může mít jiný průběh nebo zpracování

[1] Černá skříňka - Z venku není vidět, jak funguje a jaké procesy uvnitř probíhají

než v předkovi. V OOP se často využívají rozhraní neboli interface. Interface je pravidlem, jaké chování by objekt měl vykazovat, a zavazuje se implementací metod, které interface obsahuje.

OOP se dnes používá často díky možnosti rozšířitelnosti, kdy objekty lze znovu použít. Díky vlastnostem OOP je programování jednodušší, a čím lépe je kód napsaný, tím více je implementace v budoucnu zjednodušena.

Obrázek 2: Vlastnosti OOP [2]

4.2.3 PHP

PHP je skriptovací programovací jazyk určený hlavně pro vývoj webových stránek a aplikací ve formátech HTML nebo XHTML. V dnešní době je velmi opovrhovaným jazykem, ale pořád se jedná o jeden z nejpopulárnějších jazyků díky jeho jednoduchosti. PHP byl původně vytvořen pro začátečníky a podle toho se odvíjela i jeho cesta. PHP vznikal od formulářového interpretru přes sadu nástrojů a v roce 1997 již vznikl programovací jazyk PHP. V pozdější době bylo PHP používáno pro jednoduché weby a byl používán spíše začátečníky a nadšenci. Existuje velké množství webů napsaných v jazyce PHP, ale díky jeho jednoduchosti a nepředvídatelnosti je složité implementovat některé funkce, které v jazycích vyšší třídy nejsou takovým problémem.[30]

Výpis 1: Ukázka PHP kódu

```
<?php
```

```php
declare(strict_types=1);

namespace App;

class User
{
    private string $firstname;
    private string $lastname;

    public function __construct(
        string $firstname,
        string $lastname
    ){
        $this->firstname = $firstname;
        $this->lastname = $lastname;
    }

    /**
     * @return string
     */
    public function getFirstname(): string
    {
        return this->firstname;
    }

    /**
     * @param string $firstname
     */
    public function setFirstname(string $firstname)
    {
        this->firstname = $firstname;
    }
}
```

Implementace například webové aplikace je jednodušší a méně časově náročnější než u jazyků vyšší třídy. To je jeden z důvodů, proč je tak populární. Menší organizace, které si nemohou dovolit investovat tolik financí do vývoje, zvolí PHP jako programovací jazyk. Z velké části za to můžou i frameworky, které ještě více zjednodušují implementaci například Laravel, Symfony, CakePHP nebo Nette, který je v České a Slovenské republice nejpopulárnější díky jeho českým kořenům. [31] Z velké části frameworky pro PHP implementují MVC vrstvu, DI kontejner, router, databázovou vrstvu, šablonovací systémy, CLI a spoustou dalších užitečných funkcí například cache, mail atd.

Nette Framework

Nette je pod Open Source licencí a bylo vyvinuto Davidem Grudlem. [32] Nette běží na PHP 5, PHP 7 a nejnovější PHP 8. Součástí tohoto frameworku je MVC vrstva, v Nette MVP (Model View Presenter), router, DI kontejner, tracy (debugovací nástroj),

formulářové komponenty, testování a další. Všechny komponenty, ze kterých se Nette skládá, jsou samostatně použitelné a lze je využít ve více frameworcích. Okolo Nette se v České republice vytvořila velká komunita a spolu s českou dokumentací se stalo Nette stupněm pro nováčky ve vývoji webových aplikací. Nette používají velké společnosti, jako jsou například Slevomat, DHL, Zásilkovna a další.

Jednou z velkých výhod Nette je jeho zabezpečení. Nette klade velký důraz na bezpečnost, a proto dbá na dobré zabezpečení formulářů. Kromě toho, že formuláře ochrání před útokem Cross-Site Scripting (XSS) a Cross-Site Request Forgery (CSRF), dělá spoustu drobných zabezpečení a řeší bezpečnostní rizika, o kterých mnoho vývojářů ani netuší, že existují.

Laravel Framework

Laravel je Open Source framework pro PHP, který navazuje na model MVC. Jeho předností je usnadňovat vývoj webových aplikací, kdy je možno se zaměřit na vývoj a detaily nechat na Laravelu. Laravel poskytuje spousty funkcí jako MVC vrstvu, DI kontejner, databázovou vrstvu, fronty pro úlohy, testování jednotek a obsahuje CLI s názvem Artisan [33].

Jelikož je Laravel nejpopulárnějším z PHP frameworků, existuje k němu spousty materiálů, dokumentací a díky popularitě i velká komunita. Komunita stále přispívá k růstu toho frameworku a díky tomu se stává všestranným nástrojem při tvorbě webových aplikací.

4.2.4 Java

Java je programovací jazyk vyvinutý v roce 1995 pro Sun Microsystems, dnes spadá po Oracle. [34] Java je objektově orientovaný jazyk, který se podobá jazyku C++ a běží na všech platformách. Od C++ je odlišován tím, že odstraňuje pointry[2] a není nutno uvolňovat místo v paměti ručně. V Javě uvolňování paměti řeší garbage collector, který ji uvolňuje automaticky. Pokud na nějaký objekt už neexistuje reference, tak tuto paměť uvolní pro další použití. Původní použití Javy bylo v set-top boxech a televizích. Dnes se pomocí tohoto jazyka dají implementovat webové aplikace pomocí Spring frameworku, Thymeleaf, big data, mobilní aplikace pro Android, počítačové hry, pomocí grafických knihoven jako je třeba OpenGL. Jednou z nejslavnějších her vyvinutých v jazyce Java je Minecraft, který je i dnes jednou z nejhranějších her na světě. Dnes Java není tak populární, ale stále pohání spoustu zařízení a programů, tudíž zájem o programátory stále je a velké firmy, které vytváří velké projekty, stále používají Javu pro vlastnosti, které nabízí.[35]

Java se nabízí v několika edicích. Java Standard Edition (Java SE) je používána pro desktopové aplikace s knihovnou pro grafické komponenty Java FX a Swing. Další edicí je Java 2 Enterprise Edition (J2EE). Od roku 2018 se nazývá Jakarta EE a je nadstavbou Java SE. Tato edice je určena pro provoz a správu webových systému, podnikových systémů a informačních systémů [36]:

[2]Přímé ukazatele do paměti využívány jazyky C++, C a Pascal

- Jednoduchý - Java je jednoduchým jazykem pro její syntaxe. Jak je zmíněné výše, tak odpadá i ruční správa paměti.
- Přenositelný - Je přenositelný, kdy jazyk vyvinutý na jedné platformě se lehce přenese na jinou platformu.
- Objektově Orientovaný - (Kapitola 4.2.2)
- Interpretovaný - Java je kompilována do bytekódu, který je pak interpretován run-time prostředím.
- Bezpečný - Díky tomu, že je Java kompilována do bytekódu, spouštění kódu v sandboxu a explicitním pointerům jsou aplikace více zabezpečené.
- Dynamický - Podporuje dynamické změny prostředí, dynamickou alokaci paměti a výkon aplikace je zvýšen.
- Vícejádrový - Více vláken umožňuje pracovat paralelně a díky tomuto je programování o dost jednodušší.
- Robustní - Díky paměťovému managementu pomáhá s hledáním chyb při kompilaci.
- Vysoce výkonný - Java dosahuje vysokého výkonu díky kompilaci do bytekódu, který je čitelný pro stroj a lehce kompilovatelný do nativního strojového kódu.

Výpis 2: Ukázka Java kódu

```
package app;

public class User {

    private String firstname;
    private String lastname;

    public User(String firstname, String lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }

    public String getFirstname() {
        return this.firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
```

```
        return this.lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
}
```

Spring

Spring je nejpopulárnější Open Source framework pro Jakarta EE a vznikl v roce 2002 pod licencí Apache 2.0. [37] Výhodami tohoto frameworku je vysoký výkon, lehce testovatelný a znovupoužitelný kód. Spring má vlastnosti jako většina frameworku, jako je testování, DI kontejner, MVC vrstva, vlastní aspektově orientované programování, autorizační a autentifikační vrstvu a další. Spring spolu s jeho základními vlastnostmi používá několik knihoven pro rozšíření, jako jsou ORM frameworky [3], které slouží pro mapování dat z databáze na předem definované objekty a naopak. Další knihovnou, která zastupuje funkce cronu pro plánování a automatické spouštění úloh v aplikaci je Quartz. Často je Spring spojován s frontend frameworkem Thymeleaf, který je i doporučený pro implementaci se Spring aplikací.[38]

4.2.5 Python

Python je Open source skriptovací programovací jazyk, který vznikl v roce 1991. [39] Od roku 2018 vzrostla jeho popularita a je používán velkými firmami, jako jsou Google, Netflix, Spotify, Instagram, Facebook, Dropbox atd. Díky jeho jednoduché a přehledné syntaxi je vhodný pro začátečníky. Python umožňuje programovat podle několik programovacích paradigmat, jako je objektově orientované, procedurální a funkcionální programování. Python běží na většině platforem, jako jsou Unixové systémy, Windows, macOS a Android. Jeho jádro je napsáno v jazyku C a díky tomu umí velice rychle zvládat výpočty. Python je dynamicky interpretovaný jazyk a díky tomu se chyby projeví až při spuštění programu. Toto chování je někdy příhodné, když je potřeba, aby program chvíli běžel a vidělo se, kde nastala chyba. Odpadají chyby, které vyskočí při spouštění kódu. Python také velice dobře spolupracuje s ostatními jazyky, jako jsou Java, C++, C# atd.[40]

Výpis 3: Ukázka Python kódu

```
class User:
    def __init__(self, firstname, lastname):
        self._firstname = firstname
        self._lastname = lastname

    def get_firstname(self):
```

[3] Object Relational Mapping

```
        return self._firstname

    def get_lastname(self):
        return self._lastname
```

Django

Django je Open Source webový framework pro Python. Django podporuje rychlý vývoj a čistý design jako Python. Django řeší bezpečnostní chyby, nabízí autentifikační vrstvu, ORM, který vytváří API pro komunikaci s databází pomocí modelu jako u Spring frameworku, formuláře renderované do HTML a přímo ve frameworku jsou zabudované lokalizace pro více jazyků. [41]

Django používají velké firmy jako Youtube, Instagram, Spotify, Bitbucket a další.

4.3 Frontendová část

Frontendová část nebo FE je prezentační vrstva, která tvoří rozhraní pro uživatele za účelem práce s daty, která jsou uložena v databázi, nebo jen prohlížet obsah stránky. Zde se jedná o jazyky, které běží v prohlížeči u klienta. Dnes je kladen velká důraz na frontend kvůli vzhledu celé aplikace. Pokud se jedná například o e-shop, je důležité prezentovat zákazníkovi co nejpřívětivější prostředí a jednoduše ovladatelné. V této vrstvě je důležité vidět vše potřebné a mít funkční rozhraní pro komunikaci s backendem.

Frontendová vrstva se nachází téměř kdekoli, například v mobilní aplikaci, na domácím spotřebiči, v desktopové aplikaci, webové stránce a aplikaci. Zde bude zmíněná pouze FE vrstva webové aplikace či stránky. Pro tvorbu webových stránek je používáno HTML, které je standardem při vytváření webů. HTML je značkovací jazyk, používá hypertextové odkazy, s kterými jsou webové stránky propojeny, a obsahuje elementy jako paragraf, tabulka, list, odkaz a mnoho dalšího. Jedna webová stránka se skládá z těchto elementů, které jsou pak zobrazeny. K HTML neodmyslitelně patří i CSS, což je jazyk, který popisuje, jak se má element zobrazit. Myšlenkou vzniku bylo oddělit stylování stránky od struktury. CSS používá selektory ukazující na element, který se styluje. Pomocí HTML a CSS lze vytvořit statickou stránku, která pouze zobrazuje obsah a není potřeba dynamicky měnit data. Javascript se stal součástí tvorby webových stránek, který umožňuje ovládat GUI prvky, dynamicky vykreslovat data nebo měnit vzhled stránky.

4.3.1 Javascript

Javascript neboli ECMAScript je řazen mezi multi-paradigmatové jazyky. To znamená, že při psaní jazyka je možné psát pomocí více stylů programování jako například procedurální, aspektově orientované nebo objektově orientované. Javascript lze popsat slovem multi-platformní, objektově orientovaným, událostmi řízeným skriptovacím jazykem. Vše, co je potřeba pro spuštění Javascriptu, je webový prohlížeč jako Chrome, Firefox, Opera atd.

Javascript není jen jazykem, který se používá při vývoji webových stránek jako frontendová část aplikace. Javascript může běžet i na serveru díky Node.js. Ten je pouze prostředím, které umožňuje spustit kód jinde než v prohlížeči, a slouží jako backendová vrstva. Toto prostředí využívá engine V8, který používá Google Chrome, Chromium, Opera a dnes již Microsoft Edge pro interpretaci Javascriptu v prohlížečích. Javascript je velmi populárním jazykem a při vývoji webových stránek nebo webových aplikací je na vrcholu. Zde je výhodou velká komunita, kdy vznikají různé knihovny pro ulehčení práce s daty, cykly, UI komponentami atd. Na internetu lze najít i spoustu materiálů jak, tvořit webové stránky.

V Javascriptu existuje spoustu funkcí a vlastností, které dělá Javascript lepším programovacím jazykem. Proto je vhodné zde nějaké z těchto funkcí zmínit. Jednou z těchto vlastností jsou Arrow funkce. Arrow funkce se staly velmi populární součástí Javascriptu nejen díky krátkosti a eleganci v psaní, ale i díky vlastnostem, které mají. Normální funkce totiž vnímají, jak jsou zavolány, ale u Arrow funkcí záleží, kde jsou zavolány, to činí tyto funkce znovupoužitelné na více místech v programu. Další oblíbenou vlastností je destructuring, tato vlastnost umožňuje vzít z objektu jen chtěné atributy nebo metody a není potřeba pracovat s celým objektem. Pro upřesnění zde je výhoda v čitelnosti kódu, kdy se přesně ví, co je z objektu používáno. Další velmi ceněnou vlastností je práce s asynchronním voláním pomocí async/await. To umožňuje čekat na asynchronní operace bez nutnosti volání callback funkcí, které se při větším počtu zanoření stávají nečitelné.[42]

Javascript má jako většina programovacích jazyků i nevýhody. Jednou z nevýhod je jeho volnost, kdy je to zároveň výhoda i nevýhoda. Lze si dělat téměř vše a v případě, kdy programátor nezná dobře Javascript, může tvořit špatný přesto funkční kód, který je těžko opravitelný. Typy proměnných nelze deklarovat, tudíž není spolehnutí na datové typy atributů v kódu. Tyto chyby se objevují za chodu programu, ten může fungovat správně, ale jakmile přijdou špatná data, může spadnout aplikace. Jako další nevýhodou je interpretace ve více prohlížečích, kdy záleží, jak je implementován a může měnit chování některých funkcí. Dále jsou problémy s hledáním chyb, nebo v případě jedné chyby nevykreslí zbytek stránky. [43]

Výpis 4: Ukázka Javascript kódu

```
<script>
    const person = [
        {
            firstname: "John",
            lastname: "Doe",
        },
        {
            firstname: "Nhoj",
            lastname: "Eod",
        },
    ]
    let firstnames = "";
```

```
    for (let key in person) {
        firstnames += `${person[key].firstname} `;
    }
    console.log(firstnames.trim());
</script>
```

4.3.2 React

React je Open Source Javascript knihovna a byl vytvořen pro zjednodušení tvorby webových stránek a byl vytvořen společností Facebook. [4] Jeho význačnou vlastností je jeho rychlost, kde při vykreslování je rychlejší než většina frontend frameworků a knihoven. React se skládá ze znovupoužitelných komponent, které mohou mít danou funkcionalitu. React místo DOM používá virtuální DOM, který se aktualizuje, pokud nastala nějaká změna. Tomuto se říká, že je reaktivní a dynamicky vykresluje změny, které se provedou na stránce bez nutnosti znovu načíst stránku. React nabízí Devtools, pro jednodušší vývoj a správu aplikace při vývoji.

Výpis 5: Ukázka React kódu [4]

```
class Button extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            value: null,
        };
    }

    render() {
        return (
            <button className="my-button" onClick={() => alert('click')}>
                {this.props.value}
            </button>
        );
    }
}
```

React využívá pro šablonování JSX, který umožňuje vykreslovat dílčí HTML komponenty. React nabízí i vývoj mobilních aplikací pomocí React Native. Je to framework, který využívá Javascript a pomocí stejného designu a stejné funkcionalitě lze s knihovnou React tvořit jak webové, tak mobilní aplikace, tudíž se vývojář nemusí učit novou technologii. React používá tzv. one-way data binding, což znamená, že data jsou předávána komponentě, a pokud jsou tato data změněna v komponentě, tak model o těchto změnách neví.[44]

4.3.3 Vue.js

Vue.js je progresivní framework pro Javascript, který se používá k tvorbě webů, single-page aplikací, za použití Electron frameworku k desktopovým aplikacím a díky frameworkům

umožňuje vývoj mobilních aplikací. Vue.js se řadí mezi nejpopulárnější frameworky díky jeho jednoduchosti na naučení. Lze implementovat do skoro jakéhokoli projektu a nabízí flexibilní řešení pro využití více frameworků v jednom projektu. Jeho ceněnou vlastností je velikost frameworku, která činí něco okolo 20 KB. Využívá virtuální DOM, jako tomu je u Reactu a oproti Reactu umí Vue.js two-way data binding, které si vzalo od Angularu. Což znamená, že model vidí změny, které komponenta udělala a obráceně. Díky této vlastnosti je jednodušší implementace komponent, kdy není potřeba řešit, jak data upravená v komponentě namapovat zpět na model. Pro někoho výhodou a pro někoho nevýhodou je jednosouborová komponenta, kde v jedné komponentě se nachází šablona, funkční kód a stylování.[45]

Vue.js dále nabízí VueRouter, který se stará o pohyb v aplikaci a odkazy na sekce v aplikaci. VueLoader, díky kterému lze psát komponenty do jednoho souboru, Vue-CLI pro jednoduchou konfiguraci projektu a ovládání z příkazové řádky. Vuex, které nahrazuje state jako v Reactu a umožňuje ukládat proměnné, manipulovat s nimi a funguje jako takové uložiště. Nakonec by bylo vhodné zmínit Vue Devtools, které lze přidat do prohlížeče a je velkým pomocníkem při vývoji Vue aplikací.

Výpis 6: Ukázka Vue.js kódu [5]

```
// Template
<div id="app">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>

// Script
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

4.3.4 Angular

Angular je Open Source Javascript framework pro tvorbu webů vytvořen firmou Google v roce 2001. Angular nabízí vývoj webových aplikací a single page aplikací. Angular nabízí několik vlastností jako zjednodušenou MVC architekturu. Zde se kód nepíše do jednoho souboru, ale je rozdělen do souborů podle byznys logiky, datové vrstvy a prezentační vrstvy. Dále nabízí two-way data binding, které bylo zmíněno ve Vue sekci, routování, DI,

šablonování, CLI, testování animace, integraci A11y[46], jednotkové a integrační testovaní pomocí nástroje Karma a frameworku Jasmine.[6]

Výpis 7: Ukázka Angular kódu [6]

```
// HTML part
<h2>Products</h2>
<div *ngFor="let product of products">
  <h3>
    <a
          [title]="product.name + ' details'"
          [routerLink]="['/products', product.id]"
        >
      {{ product.name }}
    </a>
  </h3>
  <p *ngIf="product.description">
    Description: {{ product.description }}
  </p>
  <button (click)="share()">
    Share
  </button>

  <app-product-alerts
    [product]="product"
    (notify)="onNotify()">
  </app-product-alerts>
</div>

// TypeScript part
import { Component } from '@angular/core';
import { products } from '../products';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent {
  products = products;
  share() {
    window.alert('The product has been shared!');
  }
  onNotify() {
    window.alert('You will be notified when the product goes on sale');
  }
}
```

Pro vývoj v Angularu je potřeba znát jazyk TypeScript. Proto je Angular složitější oproti Reactu nebo Vue. Avšak díky TypeScriptu vytváří Angular spolehlivý framework pro tvorbu webů. TypeScript nabízí statické typování, rozhraní, třídy a některé funkce z OOP a pro vývoj velkých projektů je Angular dobrou volbou, ale vyžaduje více času na implementaci.

5 Návrh aplikace

V každém systémů nebo aplikaci je stěžejní vybrat řešení či technologie, které se budou v implementaci používat. Jedno řešení, které je vhodné pro správu pracovníků na pracovišti nemusí splňovat požadavky, v tomto případě s docházkou. Správná volba technologií je důležitou částí a může dát záruku, že nebude potřeba přepisovat aplikaci do modernějších či jiných technologií v blízké době.

V průběhu tohoto návrhu je potřeba zvážit klady a zápory technologií a vybrat ty, které nejméně omezí ve spolehlivosti a rychlosti aplikace a zároveň budou co nejlevnější pro klienta. Například

Java je výkonný a bezpečný jazyk, ale je finančně náročnější než vývoj, nasazení a správa webové aplikace v PHP. V návrhu aplikace budou zohledněna následující kritéria pro výběr technologie. Tato kritéria byla sepsána podle článku [47].

Prvním kritériem jsou firemní potřeby. Zde je důležité zhodnotit, která technologie je vhodná pro budoucí chování aplikace, kde se aplikace bude využívat, kdo ji bude využívat a jaké nároky budou kladeny na aplikaci na základě potřeb a přání klienta.

Druhým kritériem je zhodnotit čas implementace, kolik času zabere. Některé technologie jsou časově náročnější na implementaci a například z důvodů klientových prostředků je potřeba toto kritérium zhodnotit a rozhodnout, zda je finančně jednodušší použít jinou, méně spolehlivou technologii.

U třetího kritéria se řídí podle financí klienta. Tento krok souvisí s dalšími kroky. Pokud klient nemá dostatek finančních prostředků nelze očekávat, že aplikace bude splňovat všechna klientova kritéria a čas na její implementaci je zkrácen.

Dalším faktorem je i budoucí správa aplikace. Zde je potřeba odhadovat, kolik financí a finančně závislých prostředků může klient vložit do budoucí správy aplikace. V tomto případě se volí na základě klientových prostředků, například klient má server, kde aplikaci může nasadit, nebo má lidi na správu aplikace.

Jedním z důležitých aspektů, na který je potřeba brát ohledy, je rozšířitelnost aplikace. Zda má klient plán v budoucnu aplikaci rozšiřovat o další funkcionality. Při výběru technologií se často volí například technologie s jednoduchou škálovatelností v případě databáze a znovupoužitelnými částmi kódu v případě frontendové části, těmito částmi mohou být komponenty.

Dalším kritériem je zohlednit bezpečnost technologie kvůli ochraně osobních údajů. Dnes většina aplikací pracuje s osobními daty. Tato data jsou dnes pod ochranou GDPR a musí se zabezpečit, aby tato data nebyla veřejně dostupná a omezit čtení těchto dat jen na osoby, které jsou k tomuto pověřeny.

Na základě těchto kroků se bude zohledňovat, zda je takové řešení adekvátní k aplikaci a klientovým potřebám.

Dostupné prostředky klienta

Na základě konzultací s klientem bylo zjištěno, že klient vlastní webovou aplikaci na svém serveru spolu s relační databází. Tyto prostředky by bylo vhodné použít z hlediska financí, kdy se databáze vytvoří a přidá k existujícímu serveru a backendová část se nasadí spolu s aktuální aplikací. Prostředí jsou už vytvořená a na jednu stránku omezuje použití jiné technologie pro aplikaci i databázi, ale na druhou je výhodou, kdy není potřeba vytvářet nové prostředí.

5.1 Databázová část

Pro databázovou část bylo vybráno nejlevnější řešení, které je v dané situaci možno nabídnout. Jelikož autor má nejvíce zkušeností s relačními databázemi, bude toto řešení nejrychleji implementovatelné. V tomto případě, kdy klient má možnost nasadit relační databázi na svůj server, není potřeba řešit vytváření nového serveru nebo přizpůsobení podmínek pro jinou technologii. Relační databáze jsou spolehlivé, co se týče dat, a pokud se opustí od nutnosti měnit strukturu, ve které tento typ databáze moc nevyniká, tak splňuje kritéria, která jsou uvedena výše.

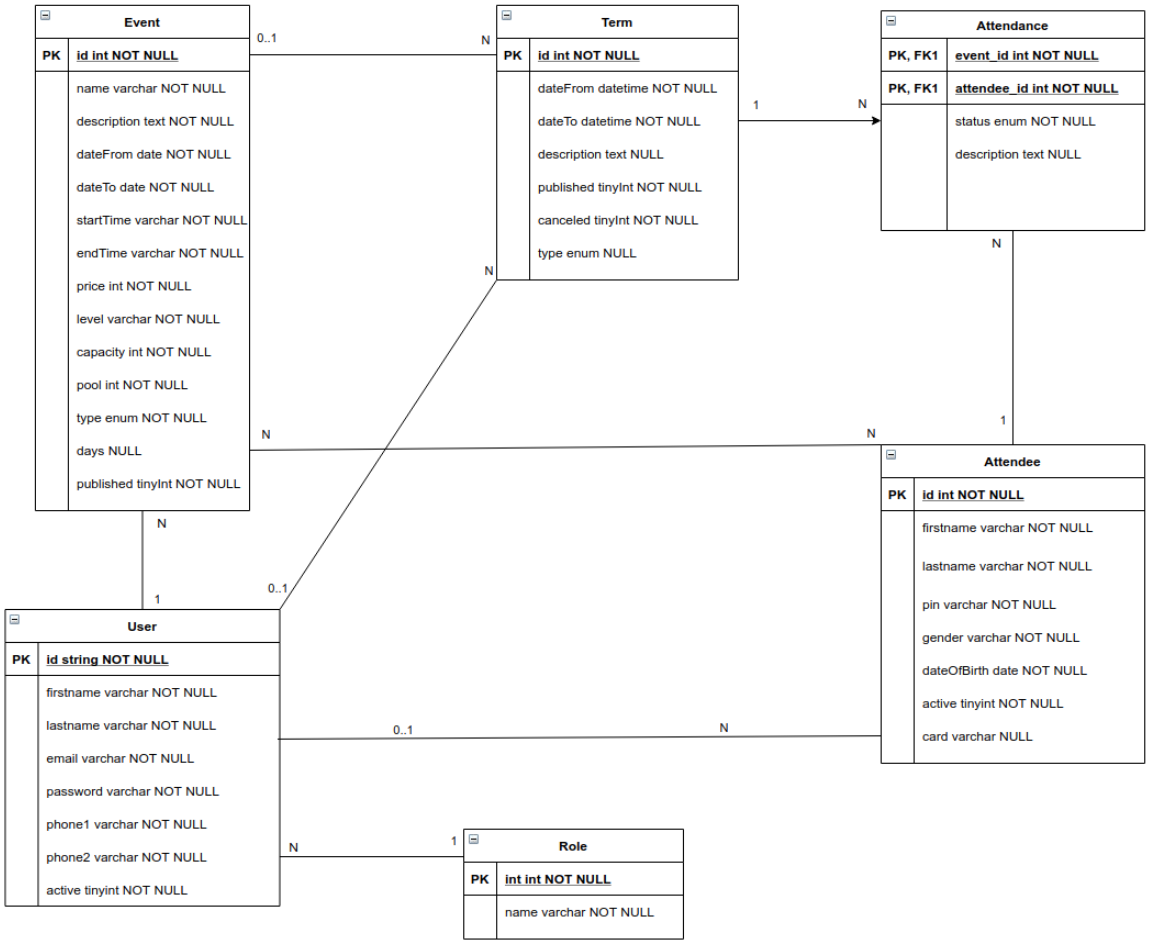
5.1.1 Zvolené technologie

Kvůli výhodám zmíněných výše není potřeba vyhledávat řešení v NoSQL databázích, které mají jako vlastnost toleranci oddílů a je vhodné zvolit technologii s vlastnostmi dostupnost a konzistence. Konkrétně byla vybrána MariaDB, která je Open Source technologií oproti MySQL a je výrazněji lehčí technologií oproti PostgreSQL. Tyto tři technologie mají téměř stejné funkcionality. Jelikož je toto systém menších rozměrů, není potřeba integrovat databázi, která má velký výpočetní výkon, ale je spíše spolehlivá, jednoduše integrovatelná a ke správě není potřeba mnoho prostředků.

5.1.2 Návrh databázové struktury

Návrh struktury byl potřeba udělat tak, aby schéma bylo co nejjednodušší a snadno rozšiřitelné. Ve schématu je uživatel navržen do jedné entity a má relaci s rolí. Tabulka s rolí byla přidána kvůli jednodušší validaci role u uživatele, tudíž se nemůže stát, že by uživatel měl roli, která není v systému. Zajímavější částí je relace mezi docházejícími a událostí. Je zde potřeba zmínit, že v budoucnu při implementaci funkcionality s kontrolou o zaplacení zde bude využita vazební tabulka, která bude obsahovat všechny údaje, které je potřeba pro namapování skutečnosti, že docházející zaplatil za událost. Pro vytvoření identifikátoru bude použito číslo události, identifikátor docházejícího a termín konání. Bylo by vhodné zmínit relaci mezi docházejícím a termínem. Kdy daný docházející může mít docházku, která je pro jakýkoli termín. Je možnost vytvořit docházku pro akci, trénink nebo osobní událost v kalendáři. V budoucnu by mohlo být možné vytvářet interní události jako je například

úklid nebo schůze. Schéma je připraveno pro implementace funkcionality organizací, kdy bude potřeba vytvořit vztah uživatele a události k organizaci.



Obrázek 3: Návrh databázové struktury

5.2 Backendová část

Pro backendovou část byly vybrány technologie, které jsou jednoduše implementovatelné a z hlediska finančních i časových prostředků nejlevnější. Konkrétně bylo vybráno PHP díky jeho jednoduchosti v implementaci. PHP je rozšířeným jazykem, proto je vhodný pro následnou správu, kdy rozšíření PHP v České republice je velké a není proto problém najít někoho, kdo se bude o aplikaci nadále starat. Je zde zahrnuto jednodušší a levnější nasazení PHP aplikace, kdy není těžké najít server, který bude finančně výhodný i spolehlivý.

Jako framework pro zjednodušení práce bude použito Nette. Nette je jednoduchým řešením a rychle implementovatelným frameworkem pro webovou aplikaci a autor má s ním zkušenosti. Oproti více populárním frameworkům je Nette více minimalistický, kdy

jeho struktura není tak velká jako u Symfony. Nette má kolem sebe rozrostlou komunitu a vzniká pod tímto projektem několik knihoven, které se využijí v této aplikaci.

Jednou z knihoven pod Nette se zvolilo použití Apitte. Pomocí Apitte se bude vytvářet API rozhraní, pomocí kterého bude komunikovat s dalšími částmi aplikace. Pod Apitte je i knihovna, která umožňuje vytvářet takzvané middleware. Middleware umožňuje přidat funkcionality mezi přijmutím a zpracováním požadavku. Pomocí middleware se bude řešit zabezpečení, kdy se bude kontrolovat, zda uživatel bude mít přístup k danému koncovému bodu, nebo jestli má platné přihlášení.

Další knihovnou bude použit Nettrine. Tato knihovna je integrací Doctrine do Nette projektů. Doctrine implementuje objektově relační mapování, které umožňuje přistupovat k záznamům v databázi jako k objektům. Ulehčuje práci s tabulkami, protože k nim je možno přistupovat objektově a držet se OOP. Pod Nettrine knihovnou existují další knihovny jako migrace, které ulehčí práci s vytvářením migrací a následným spuštěním v prostředí databáze.

Dále bude použita knihovna s integrací konzole. Ta je doporučená s využitím Doctrine, kdy je možné provádět generování migrací nebo vytváření migrací podle vytvořených entit. Pomocí konzole je možné přistoupit k dalším rozšířením přímo na produkčním prostředí a lze ušetřit čas při řešení problémů.

5.2.1 Architektura

V této části bude navržena architektura ve smyslu společného fungování komponent v aplikaci za dosažením daného cíle. Základní proces je jednoduchý. Přijde požadavek, ten je vyhodnocen a dále předán pro zpracování a vrácení odpovědi. Díky jednoduchosti aplikace je možno navrhnout základní architekturu tak, aby byla přehledná a jednoduše rozšířitelná. Aplikace je malá a není tedy potřeba aplikovat složitá architektonická řešení. Backendová část bude pouze jako API, který umožní komunikaci s databází a v budoucnu s aplikacemi třetích stran. V budoucnu bude obsahovat skripty, které budou kontrolovat stav zaplacení za akci a další rozšíření.

Pro přístup k aplikaci budou vytvořeny koncové body, které před vstupem budou prověřovat, zda daný uživatel má právo přistoupit. Tuto funkci bude mít na starosti Middleware, které bude ověřovat validitu tokenu a uživatele. V případě, kdy není potřeba ověřovat uživatele, se toto prověření přeskočí. Proto je vhodné rozdělit koncové body podle viditelnosti na veřejné a privátní.

Dalším balíkem jsou kontrolery, které budou rozděleny podle dat, ke kterým přistupují. Kontrolery budou komunikovat se službami, které implementují očekávané chování. Pokud se bude vytvářet entita, tyto služby vykonají vše potřebné pro vytvoření entity. Služby budou komunikovat s repozitáři, jenž v základu budou mít klasické CRUD operace. Kvůli čistotě budou modelové třídy obsahovat pouze metody pro získání nebo nastavení atributu. Vše ostatní budou dělat služby.

Pokud bude koncový bod přijímat data, tak tato data musí být ověřována, zda chodí hodnoty v požadavku ve správném tvaru a zda jsou povinná. Dále bude potřeba tato data mapovat na modely, které zastávají roli entit z databáze. Popřípadě zde budou další funkcionality, které umožní vytvářet relace mezi entitami. Pro odpovědi na požadavky budou využívány třídy, které mapují data z entit v tabulkách na data, která se budou odesílat na frontend.

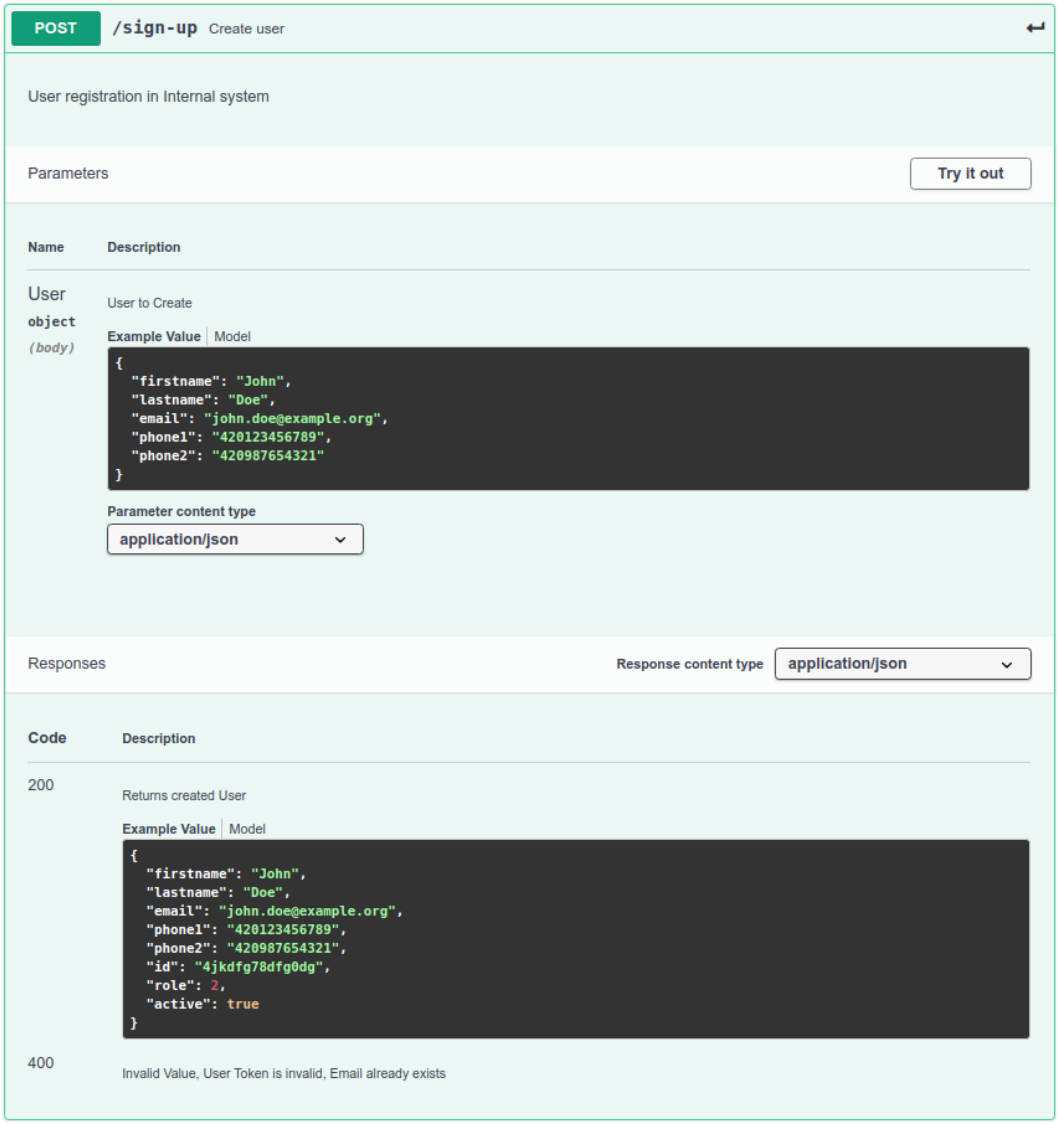
Struktura

```
app
├── config
├── Controller
│   ├── Auth
│   ├── Pub
│   └── BaseV1Controller.php
├── Decorator
├── Dispatcher
├── Fixtures
├── HandlerExtension
├── Helpers
├── Mapping
├── Middleware
├── Model
│   ├── Entity
│   ├── Factory
│   ├── Hydrator
│   ├── Repository
│   ├── ResponseMapper
│   └── EntityManager.php
├── Security
├── Service
├── ValueObject
└── Bootstrap.php
bin
data
db
docker
log
temp
vendor
www
```

5.2.2 Komunikace

Komunikace s backendem je navržena jako REST. Komunikace bude probíhat přes HTTPS protokol za pomoci JSON dokumentů. Takové řešení je jednoduché a široce rozšířené a

flexibilní při vytváření koncových bodů pro daný zdroj. Struktura bude rozdělena podle entit, což znamená, že všechny koncové body, které pracují s entitou uživatele, vytvářením, editací, mazáním nebo vylistováním, budou pod jedním zdrojem.



Obrázek 4: Ukázka formy komunikace

Na obrázku je možné vidět příklad struktury komunikace, kdy na koncový bod registrace budou poslány informace o uživateli a server poté vrací objekt v JSON formátu, v případě chyby posílá zprávu, kde nastala chyba.

5.2.3 Zabezpečení

Podle klientových požadavků je potřeba implementovat registraci a přihlašování pomocí Facebooku nebo Googlu. Pro tuto funkcionalitu bude využito existující řešení Firebase od Googlu. Firebase byl vybrán i díky tomu, že umožňuje integraci notifikačního serveru, který je využitelný v části živého sledování docházky nebo pro funkcionalitu notifikací, která

bude v budoucnu implementována. Funkcionalita s Firebasem bude rozdělena na dvě části. Frontend bude mít na starosti komunikaci s Firebase pro přihlášení a registraci a držet token s informacemi o uživateli a tento token bude posílat v hlavičce na backend, kde tento token bude validován.

Při registraci Firebase u sebe vytvoří instanci uživatele, která bude sloužit pro přihlášení. Tudíž přihlášení nebude probíhat v této aplikaci a nemusí být přihlašovací údaje ukládány do databáze. Toto řešení ušetří čas a není třeba implementovat způsob zabezpečené komunikace, pokud frontend bude posílat přihlašovací údaje, kdy je potřeba zahashovat heslo před odesláním na server. Nevýhodou je integrace aplikace třetí strany, kdy je vytvářena závislost na tom, zda aplikace běží a funguje správně.

5.3 Frontendová část

Pro frontendovou část bylo zvoleno Vue.js. Jako ve výše zmíněných technologiích má autor s Vue.js největší zkušenosti, tudíž zase odpadá problém s naučením nové technologie. Vue.js se zvolilo také díky jeho jednoduchosti, kdy čas implementace je výrazně rychlejší než u Reactu nebo Angularu a pro tuto aplikaci bude vhodným řešením.

Prvně bylo zanalyzováno, jaké komponenty bude třeba využít a vybrat takový Material Design, aby nebylo potřeba dodělávat komponenty a styl komponent byl stejný. Material Design je sada předpřipravených komponent pro jednodušší implementaci. Použití umožňuje rychle vytvořit aplikaci, bez vytváření vlastních komponent. Odpadá tu stylování každé komponenty a implementace funkcionalit, kdy ve většině případů mají již optimalizované funkce, jako například vyhledávání v tabulce. Po analytické části bylo zjištěno, že je potřeba se zaměřit na Material Design, který bude obsahovat komponenty jako kalendář, tabulku pro zobrazení dat, upozornění, dialogy, formulářové prvky a vhodné výběry datumu a času.

Na základě analýzy bylo vybráno několik Material Designu. Jedním z nich je Vue Material, ten obsahuje téměř všechny komponenty, které jsou potřeba, až na kalendář. Dále byl vybrán Vuetify, který se podobá Vue Material, ale obsahuje více komponent i se zmiňovaným kalendářem. Po domluvě s klientem byl nakonec vybrán Vuetify jako framework pro Material Design.

5.3.1 Návrh wireframe

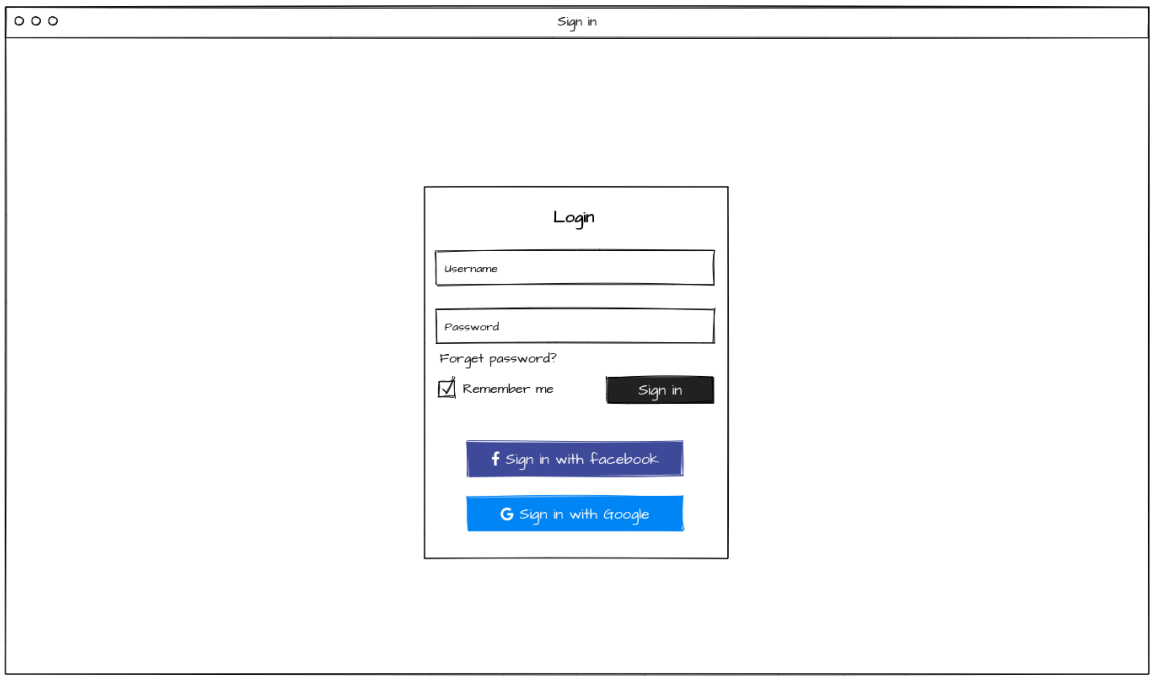
Před implementací aplikace je vhodné nakreslit návrh, jak bude uživatelské rozhraní vypadat nebo alespoň navrhnout, jak budou komponenty rozloženy. Takový návrh může posloužit i pro objevení funkcionalit, které při analýze nebyly nalezeny, nebo díky návrhu změnit některé procesy, protože z hlediska UI[4] nebo UX[5] nedávají smysl.

Často se stává, že firma nevěnuje dostatek času na UI nebo UX a vytvoří aplikaci, která nevypadá dobře, nebo má špatné UX. Tímto návrhem se alespoň snižuje šance, že

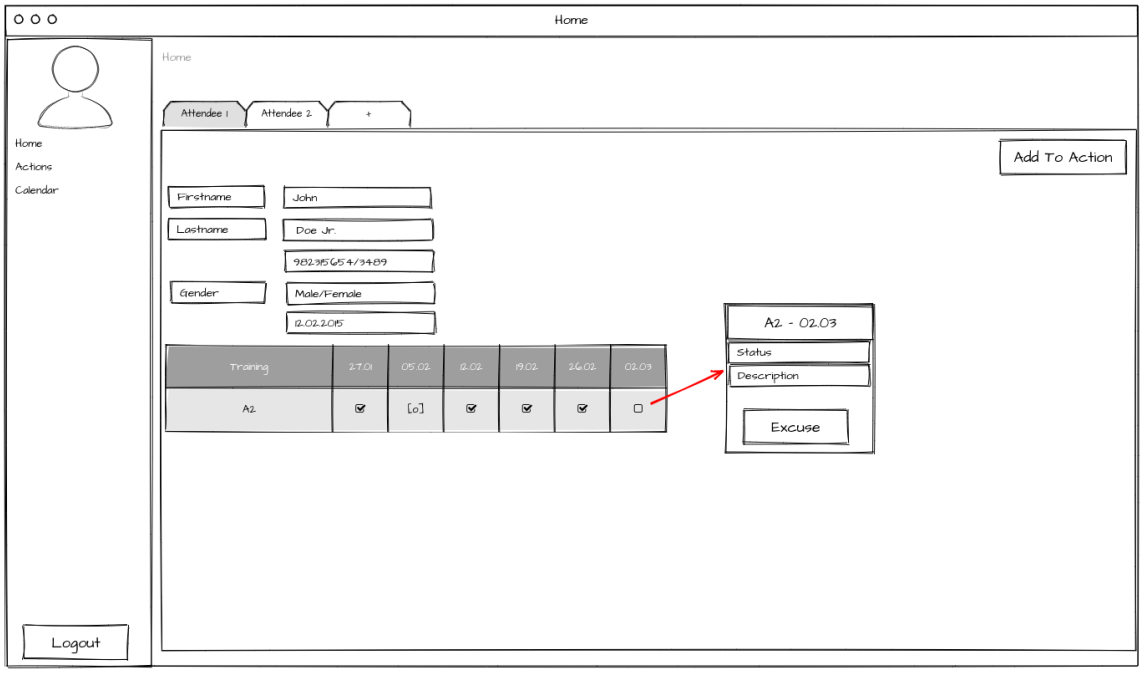
[4]User Interface - návrh vzhledu aplikace

[5]User Experience - návrh aplikace, tak aby byly splněny uživatelovy požadavky

aplikace bude fungovat nebo vypadat dle očekávání klienta. V rámci návrhu může klient zkontrolovat a schválit návrh a předejít tak problémům, které by mohly vzniknout. Tím, že vznikne návrh, je možné ušetřit práci programátorům, kdy nemusejí přemýšlet, jak udělat jednotlivé stránky nebo komponenty. Pro tuto aplikaci byl vytvořen základní návrh pro představu klienta. Ten slouží jako nastínění, jak aplikace bude vypadat z hlediska UX a v průběhu budou prováděny změny podle klientových požadavků.



Obrázek 5: Ukázka návrhu přihlášení



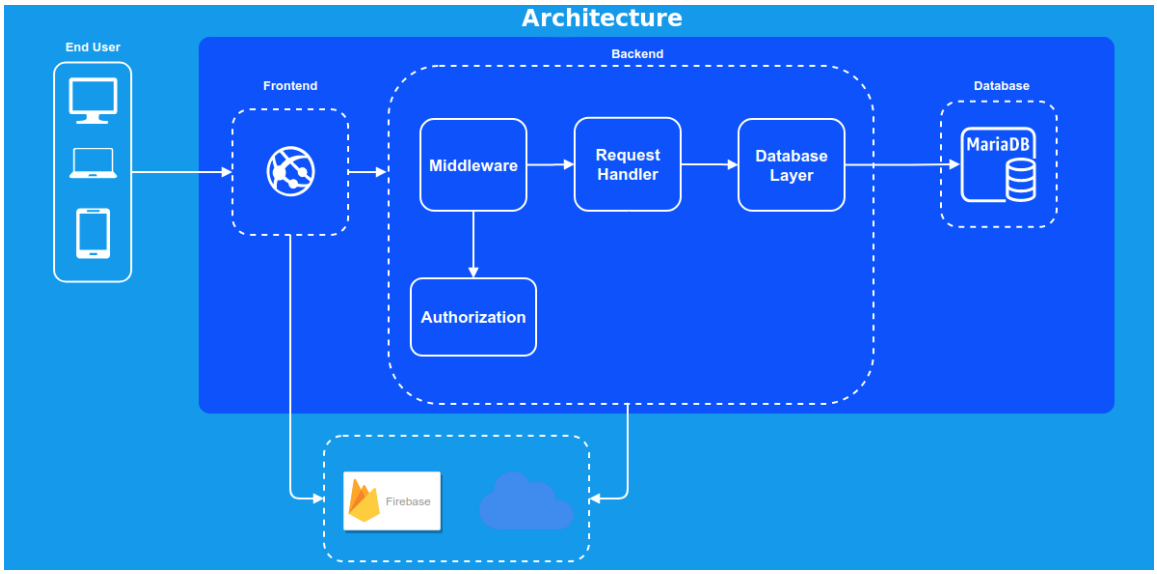
Obrázek 6: Ukázka návrhu domovské stránky

Struktura

```
public
src
    assets
    components
        app
    firebase
    layouts
        default
        login
    mixins
    plugins
    router
    services
    store
    styles
    util
    views
    App.vue
    consts.js
    main.js
```

Architektura aplikace

Dále byla navrhnuta architektura, jak budou části aplikace mezi sebou komunikovat a z čeho se bude skládat. Je zde zachyceno, kde probíhá komunikace s aplikacemi třetích stran. V budoucnu zde přibydou části jako e-mailové služby, platební brány, integrace Slacku a komunikace se čtečkou karet. V základu je architektura jednoduchá. Na obrázku 7 je možno vidět, že koncoví uživatelé budou přistupovat přes webovou aplikaci. Prvním krokem je přístup přes middleware implementované v knihovně Apitte, které ověří, zda uživatelův token od Firebase je platný a pokračuje dále na zpracování požadavku. V diagramu je znázorněna komunikace s databází pomocí databázové vrstvy, která je implementována v knihovně Nettrine.



Obrázek 7: Architektura aplikace

6 Implementace

Implementace byla rozdělena na několik částí, které se postupně protínaly. Prvním krokem bylo vytvořit lokální prostředí, kde bude aplikace vyvíjena. Autor zvolil vývoj pomocí Dockeru, kde běží backendová část spolu s databází. Frontendová část byla jednodušší, Vue.js totiž nabízí Vue-CLI a pomocí NPM lze spustit frontendová část bez dalších nástrojů a konfigurací. Jelikož v produkci nebude Docker, tak není potřeba vytvářet Docker prostředí i pro frontendovou část.

Po vytvoření vývojového prostředí začínala implementace backendové části, kdy bylo potřeba integrovat Nettrine a vytvořit základní schéma databáze a entity. Pro vyzkoušení, zda aplikace funguje, nebo jestli komunikuje s databází, bylo potřeba vytvořit nějaké rozhraní. Implementovalo se základní API pomocí Apitte. Při zprovoznění bylo vyzkoušeno, zda funguje komunikace, a začala integrace Nettrine a konzole. Po integraci Apitte byla implementována registrace a přihlášení. Spolu s tímto krokem byl vytvořen frontend a integrován Material Design s šablonou. Na frontendu vznikla stránka pro přihlášení a registraci a byla připojena na API a ověřena komunikace. Dále se integroval do projektu Firebase. Pomocí Firebase se vytvořila registrace a přihlášení a na backendové častí se vytvořilo autentifikační middleware, kde se ověřovalo, zda token poslaný z frontendové části je platný. Jakmile byla vytvořena registrace a přihlášení, postupně se implementovaly další části aplikace.

Vždy nejprve vzniklo UI na frontendu a podle potřeby byly implementovány koncové body v backendové části, aby nevznikly části aplikace, které nebudou využity. Ukázka koncového bodu, viz 8. Při implementaci koncových bodů byly vytvořeny služby, které zastávaly jejich chování, viz ukázka 9. Součástí toho byly přidány validace objektů, které přicházejí z frontendu. Tyto validace slouží k tomu, aby nechodila špatná data a byla lépe zpracovatelná v backendové části, viz ukázka 10. Pro vytvoření nebo editaci entit byly implementovány mapovací třídy s názvem hydrátor. Název hydrátor byl převzat od Symfony Doctrine, aby se nepletl s mapovacími třídami pro vytváření odpovědí na požadavky. Hydrátory mapují objekty z požadavku na databázové entity, viz 11. Jako poslední vznikly mapovací třídy, které vytvářely z objektu pole hodnot a předávaly do těla odpovědi. Při odeslání odpovědí na frontend se tělo odpovědi převedlo na JSON a bylo odesláno s příslušnými hlavičkami.

Po implementaci základní části následovalo představení klientovi. Bylo potřeba vytvořit spustitelné prostředí pro testování aplikace a představení. Pro toto prostředí se zvolilo Heroku, které umožňuje nasazení aplikací. V základním balíčku je Heroku zdarma, ale aplikace po půl hodině uspává a po každém znovuotevření aplikace je potřeba celou aplikaci nastartovat. Pro představení klientovi a testování bylo toto uspání zanedbatelné.

Po představení aplikace klientovi byla práce schválena s drobnými připomínkami a pokračovalo se dál v implementaci. Proces vytváření stránek aplikace se opakoval a ukončil

s výsledkem, který byl klientovi znovu představen. Po představení vznikly nové požadavky, opravy chyb a úpravy designu, které byly vyřešeny.

Výpis 8: Ukázka implementace koncového bodu pro editaci termínu

```
/**
 * @Path("/{termId}")
 * @Method("PUT")
 * @Responses({
 *      @Response(code="200", description="Success")
 * })
 * @RequestParameters({
 *       @RequestParameter(name="termId", type="int")
 *})
 * @RequestBody(entity="App\ValueObject\TermValueObject")
 * @param ApiRequest $request
 * @param ApiResponse $response
 * @return ApiResponse
 */
public function update(ApiRequest $request, ApiResponse $response)
{
    try {
        $termId = $request->getParameter("termId");
        /** @var TermValueObject $termValueObject */
        $termValueObject = $this->getRequestEntity($request);
        $terms = $this->termService->update(
            $termId,
            $termValueObject
        );
        return $response->writeJsonBody($terms)
            ->withStatus(ResponseHelper::OK);
    } catch (InvalidValueException $e) {
        return $response->writeJsonBody([
            'message' => $e->getMessage(),
            'validation' => $e->getErrors(),
        ])->withStatus(ResponseHelper::BAD_REQUEST);
    } catch (EntityNotFoundException $e) {
        return $response->writeJsonBody([
            'message' => $e->getMessage(),
        ])->withStatus(ResponseHelper::NOT_FOUND);
    } catch (Exception $e) {
        return $response->writeJsonBody([
            'message' => $e->getMessage(),
        ])->withStatus(ResponseHelper::BAD_REQUEST);
    }
}
```

V první části, jsou konfigurační anotace jako cesta @Path, HTTP metoda @Method, odpovědi, které vrací @Responses, url parametry @RequestParameters a tělo požadavku

@RequestBody. Dále následuje zpracování z požadavku a předání službě. V blocích catch je zpracování výjimek tak, aby vracely správné HTTP statusy a frontend věděl, kde nastala chyba.

Výpis 9: Ukázka metody registrace v uživatelské službě

```
/**
 * @param UserValueObject $userValueObject
 * @param string $token
 * @return array
 *
 * @throws AuthException
 * @throws FirebaseException
 * @throws Exception
 */
public function signUp(UserValueObject $userValueObject, string $token):array
{
    if (!is_null($this->findByEmail($userValueObject->getEmail()))){
        throw new Exception();
    }

    /** @var User $user */
    $user = $this->factory->create($userValueObject);

    $token = $this->auth->verifyIdToken($token);

    $firebaseUid = $token->claims()->get("sub");

    $user->setId($firebaseUid);

    try {
        $this->em->persist($user);

        $this->em->flush();

        $this->auth->setCustomUserClaims(
            $firebaseUid,
            ["role" => RoleEnum::USER_ID]
        );
        return $this->responseMapper->toArray($user);
    } catch (Exception $e) {
        throw new Exception();
    }
}
```

Metody služeb zastávají chování koncových bodů. Přes tyto služby se komunikuje s dalšími službami a probíhá zde zpracování entit. Služby ve většině případů vrací pole, které se v metodě koncového bodu zapisuje do těla odpovědi ve formátu JSON.

Výpis 10: Ukázka třídy pro validaci příchozích dat

```php
<?php
declare(strict_types=1);

namespace App\ValueObject;

use Symfony\Component\Validator\Constraints as Assert;

class AssignValueObject extends ValueObject
{
    /**
     * @var int
     * @Assert\NotNull()
     * @Assert\NotBlank()
     * @Assert\Type("int")
     */
    public int $attendeeId;

    /**
     * @var int
     * @Assert\NotNull()
     * @Assert\NotBlank()
     * @Assert\Type("int")
     */
    public int $eventId;

    public function getAttendeeId(): int
    {
        return $this->attendeeId;
    }

    public function getEventId(): int
    {
        return $this->eventId;
    }
}
```

V ukázce je možno vidět jednotlivé atributy, které přicházejí v těle požadavku. Je zde uveden datový typ, a zda je tento atribut vyžadován. Pokud se v těle metody pošle atribut, který není v této třídě uveden, bude ignorován.

Výpis 11: Ukázka implmenetace hydrátoru pro vytvoření a editaci události

```php
/**
 * @param ValueObjectInterface|TermValueObject $valueObject
 * @param BaseEntity|null $entity
 * @return Term
 * @throws Exception
 */
```

```
public function hydrate(
    ValueObjectInterface $valueObject,
    ?BaseEntity $entity = null
){
    if (is_null($entity)) {
        $entity = new Term();
        $entity->setPublished(false);
        $entity->setCanceled(false);
    }

    $entity->setFrom($valueObject->getStart());
    $entity->setTo($valueObject->getEnd());
    $entity->setDescription($valueObject->getDescription());
    $entity->setName($valueObject->getName());
    $entity->setType($valueObject->getType());
    if ($entity->getType() !== "my") {
        $entity->setPublished(true);
    }
    /** @var User $owner */
    $owner = $this->em
        ->getRepository(User::class)
        ->find($valueObject->getOwner());
    $entity->setOwner($owner);

    return $entity;
}
```

V ukázce je možno vidět zpracování objektu poslaného v požadavku a namapování na entitu. V hydrátorech jsou implementovány funkce pro kontrolu příchozích entit, jestli obsahují správné identifikátory, nebo jestli je vhodné změnit hodnotu. Tedy druhá část validace, kdy první validace funguje na základě datových typů a filtrování definovaných atributů.

Výpis 12: Ukázka implementace metody v repozitáři termínů

```
/**
* @param string $userId
* @param bool $withPrivate
* @return mixed
*/
public function findPublishedAndOwners(
    string $userId,
    bool $withPrivate = false
){
    $qb = $this->createQueryBuilder("t");
    $qb->where("t.published = true")
        ->orWhere("t.owner = :userId")
        ->leftJoin("t.event", "e")
        ->leftJoin("e.attendees", "a")
```

```
            ->leftJoin("a.user", "u")
            ->orWhere("u.id = :userId")
            ->setParameter("userId", $userId);
        if (!$withPrivate) {
            $qb->andWhere($qb->expr()->neq("t.type", ":type"))
                ->orWhere($qb->expr()->isNull("t.type"))
                ->setParameter("type", TermType::PRIVATE);
        }
        return $qb->getQuery()->getResult();
}
```

V ukázce je vytváření SQL dotazů pro vylistování všech termínů, které daný uživatel může vidět.

6.1 Zajímavé části

Registrace a přihlášení

Jednou ze zajímavých částí je přihlášení a registrace přes službu Firebase. Firebase byl potřeba implementovat na frontendové části i backendové části. Na frontendové části probíhá registrace a přihlášení. Proces byl navrhnut tak, aby uživateli byl zabráněn vstup do aplikace, pokud není zaregistrován a uložen v databázi.

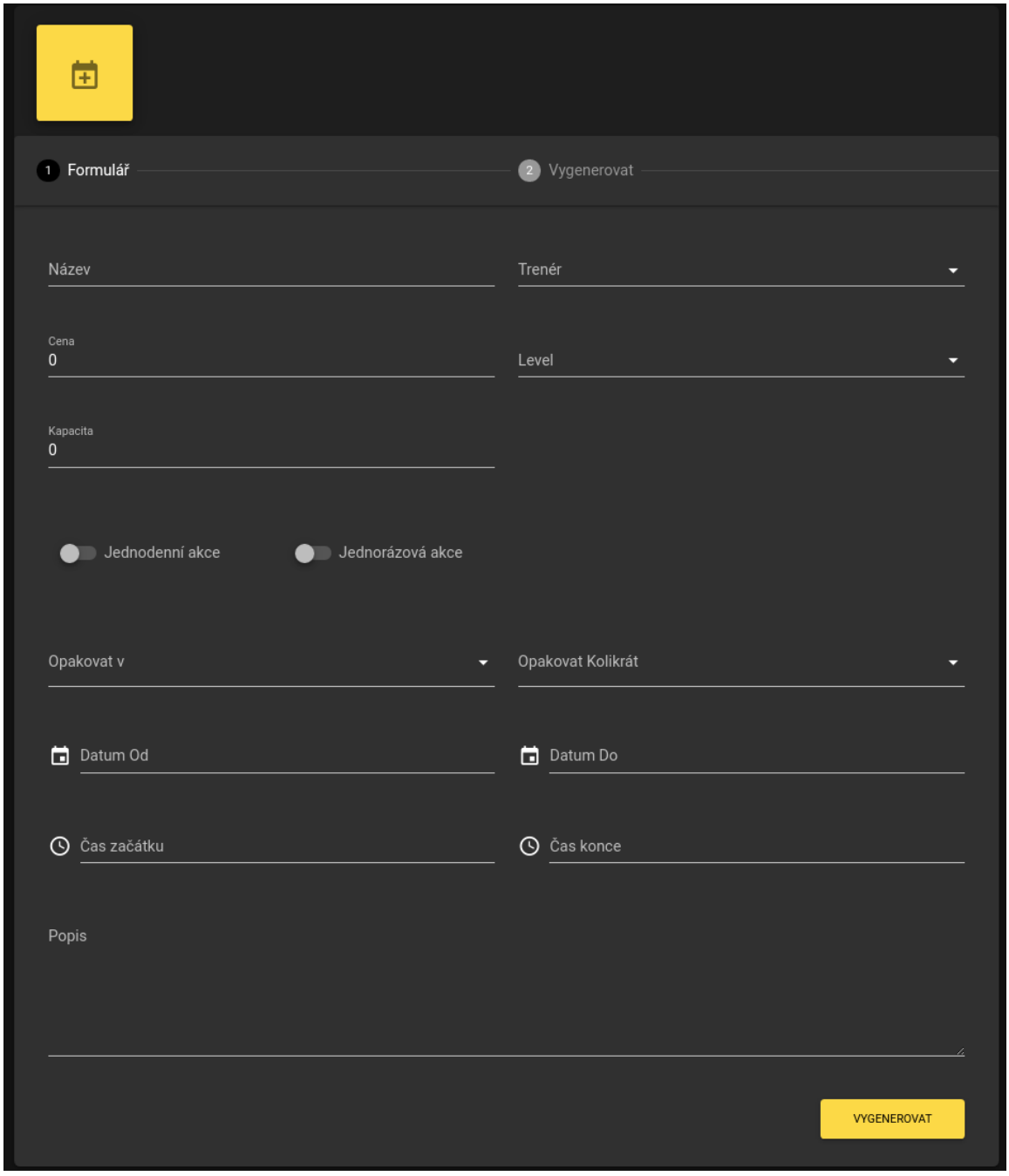
Firebase umožňuje zaregistrovat uživatele při požadavku o přihlášení v případě, kdy uživatel nemá vytvořen účet ve Firebase. Některé atributy pro uživatele jsou povinné a je potřeba si je vyžádat, aby si je vyplnil, než bude pokračovat dále. Proto bylo nutné vytvořit podmínku, pokud uživatel má platný token od Firebase a nemá vytvořenou entitu v aplikaci, nedostane se k dalšímu obsahu, ale je přesměrován na registraci. V případě, že využil například registraci přes Google, je mu předvyplněno vše z Google účtu. Dále jen potvrdí registraci a odešle požadavek na backendovou část, kde je zjišťováno, zda uživatel neexistuje s e-mailovou adresou a je pro něho vytvořena entita. Po úspěšném uložení entity do databáze se pomocí Firebase identifikátoru připíše uživateli do tokenu identifikátor role. Podle této role v tokenu frontend zjistí, zda je uživatel registrován a následně ho pustí do aplikace. Informace o roli v tokenu zůstává celou dobu existence účtu, tudíž není potřeba ji nastavovat při každém přihlášení. Dále už přihlašování probíhá pouze přes Firebase a není vyžadováno poslání požadavku na backend. S tímto řešením přišel problém se změnou role, kdy při pokusu o změnu role je nutné odhlásit uživatele a následně znovu přihlásit, protože token není při změně obnoven. Po konzultaci s klientem je toto chování v pořádku a nebude potřeba tento problém řešit jinak.

Vytvoření události

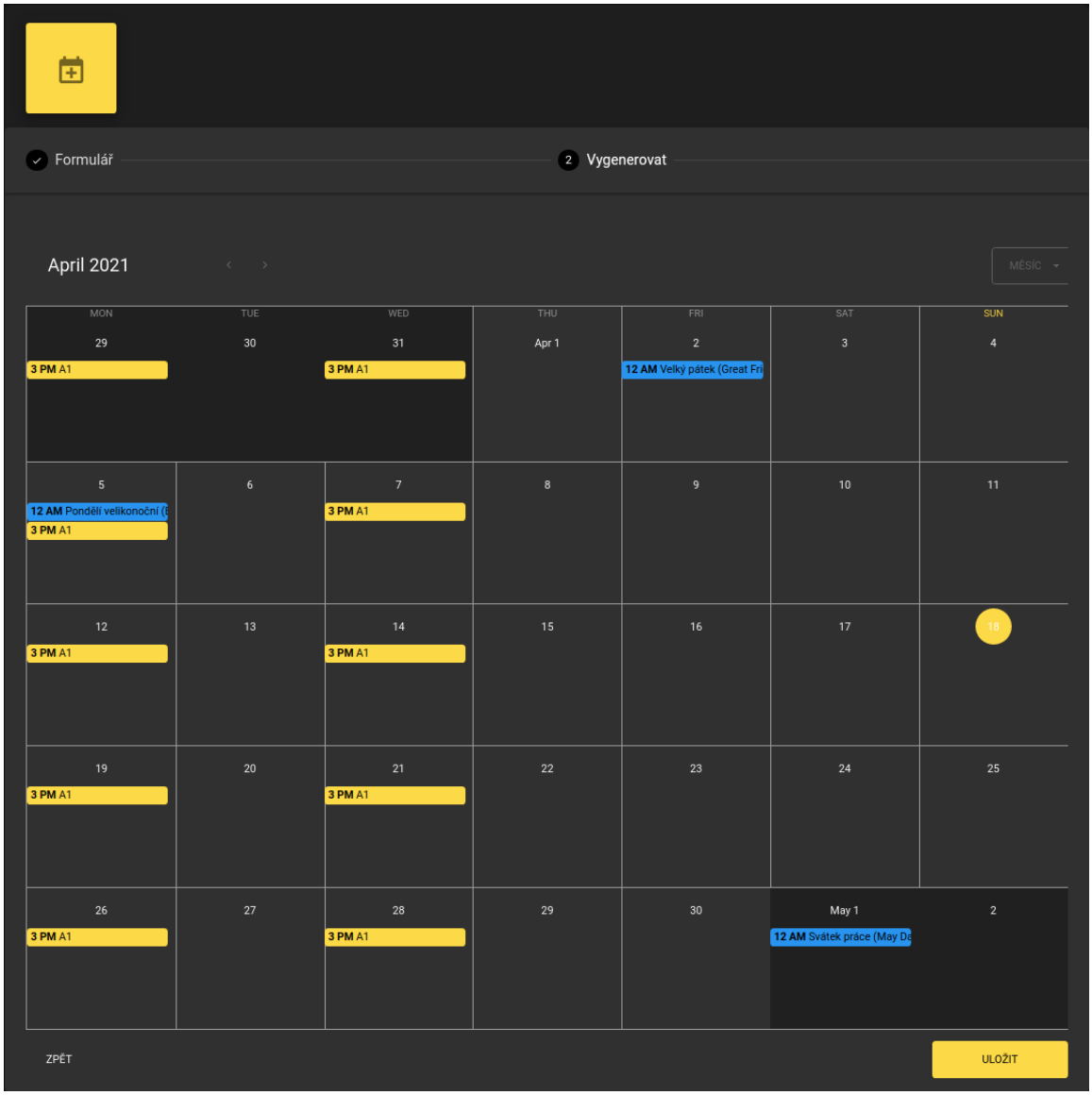
U této funkcionality proběhlo hodně konzultací s klientem. Bylo potřeba vymyslet jednoduchý proces vytvoření události, ale musel obsahovat spoustu možností na vygenerování termínů. Prvním krokem při vytvoření události je formulář, viz 8, kdy se uživateli zobrazí

možnosti podle typu události, jestli to je pravidelný trénink, nebo akce s dalšími možnostmi pro generování termínů pro jednodenní nebo jednorázovou akci. V první části formuláře jsou jen informativní prvky a v druhé jsou specifikace, jak se mají termíny pro událost vygenerovat. V této části jsou políčka, v jakém rozpětí budou generovány termíny, v jaký den a jestli jednou týdně, jednou za dva týdny nebo pouze jednou v případě jednodenní akce. Rozdílnost mezi formulářem akce a tréninku je výběr možností, zda je událost jednorázová nebo jednodenní a dle tohoto se mění výběr datumu. V případě, kdy chce uživatel vytvořit neopakující akci má možnost vybrat jednorázovou akci, kdy zvolí datum, kdy akce začíná a kdy končí, a vygeneruje podle vybraných dnů.

Dalším krokem u vytvoření události je vygenerování termínů, viz 9. Po konzultacích bylo zjištěno, že generování událostí nemíří tolik do budoucnosti, ale je tvořeno v rámci jednoho pololetí, tudíž není potřeba generovat velké množství dat, proto logika byla ponechána na frontendu a na backend jsou posílána už připravená data. V druhém kroku jsou již vygenerovány termíny podle formuláře. Ty jsou následně zobrazeny v kalendáři pro větší přehled uživatele a lze zde termíny odstraňovat nebo přidávat. Na základě klientových požadavků bylo přidáno zobrazení přerušujících akcí jako svátek nebo školní prázdniny. Po dokončení druhé části je událost společně s termíny odeslána na backend a uložena.



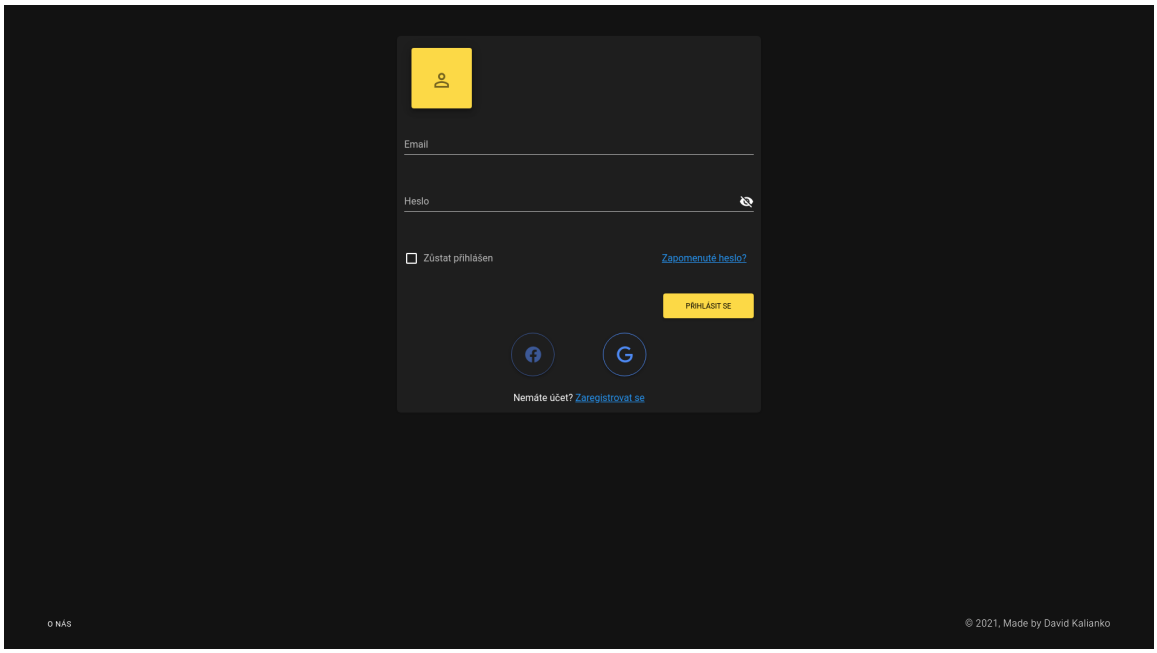
Obrázek 8: Formulář pro vytvoření události



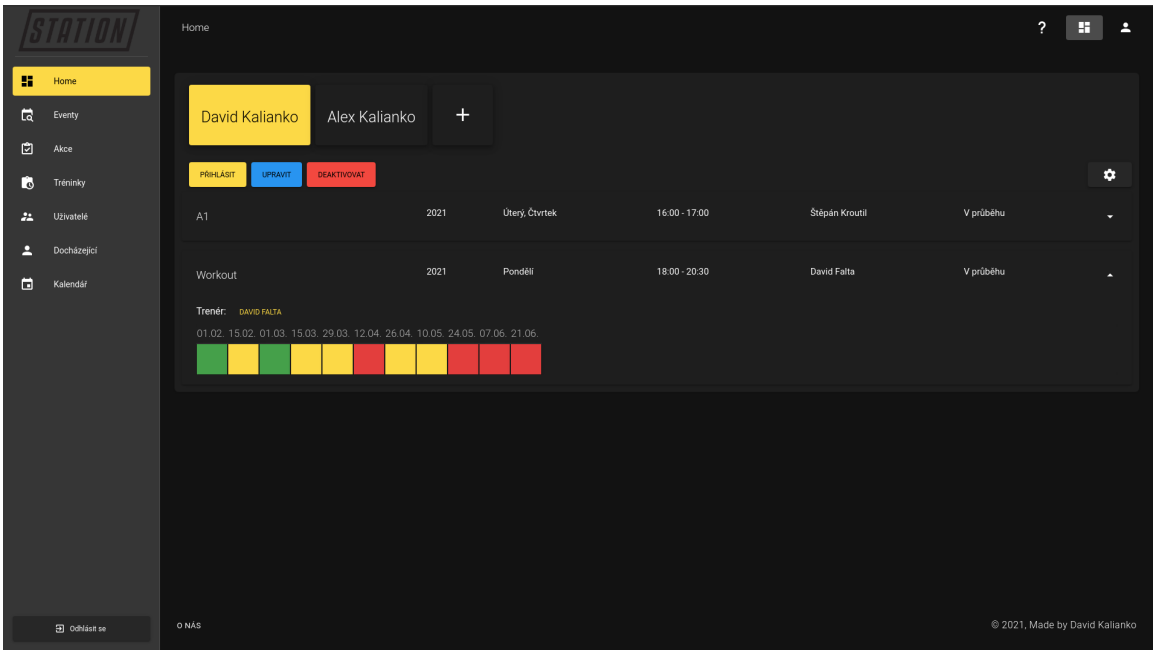
Obrázek 9: Vygenerované termíny v procesu vytvoření události

Vytvořené UI

UI se odklonilo od původního návrhu a bylo děláno podle použité šablony. Jako výchozí styl byl vybrán tmavý kvůli stylu již existujících stránek klienta a pro úplnost byl přidán i světlý režim, který slouží uživatelům, kteří tento režim preferují. S přiloženými obrázky lze porovnat návrh domovské stránky 6 a přihlašovací stránky 5 s výsledkem domovské stránky 11 a přihlašovací stránky 10



Obrázek 10: Ukázka vytvořeného přihlašovacího formuláře



Obrázek 11: Ukázka vytvořené domovské stránky

7 Testování

Testování je otázkou, kterou je důležité si položit předtím, než bude zahájena implementace aplikace. Je proto potřeba připravit si, jaké formy testování aplikace bude obsahovat. Některé formy testování lze přidat v průběhu implementace, ale některé mohou výrazně zvýšit čas implementace, pokud kód aplikace není správně napsán.

Testování programátorem

Testování programátorem je jedno z nejjednodušších a nejlevnějších testování. Jde pouze o to, že sám programátor si vyzkouší funkcionalitu, kterou vytvořil, nebo chybu, jestli opravil správně. Ač je toto testování jednoduché, je často opomíjené. Ale může ušetřit spoustu času, kdy hotový výsledek před posláním dál, vyzkouší a zjistí, zda vůbec funguje a je menší šance, že se úkol vrátí a musí jej opravit.

Testování jednotek

Toto testování je zaměřeno na test jednotek, jako jsou jednotlivé metody a třídy v případě OOP. V tomto případě se píší jednotkové testy a jsou vytvářeny programátorem. Umožní tak zjistit, že po implementaci nové funkcionality mohly vzniknout chyby ve stávajících funkcionalitách. Toto testování je potřeba implementovat už od začátku vytváření aplikace. Pokud se v průběhu implementace zjistí, že jsou potřeba jednotkové testy, už se hůř píší.

Integrační testování

Tyto testy většinou provádějí testeři, ale v případě menších týmů jsou prováděny vývojáři. Tyto testy se provádí po úspěšném testování jednotek, pokud existuje. Toto testování umožňuje testovat, jak komponenty mezi sebou komunikují. Bez těchto testů se lze vyhnout zdlouhavému nacházení chyb, ale pro výsledný produkt není tolik důležité je mít.

Akceptační testování

Akceptační testy probíhají na straně klienta, kdy klient testuje výsledek, který prošel předchozími testy. Klienti, pak dle nějakých scénářů testují aplikaci a sledují, zda jsou splněny jejich požadavky. Zde se tedy neřeší tolik programové chyby, ty by měly být vyřešeny v předchozích testech, ale zaměřují se, jestli aplikace funguje správně dle klientových specifikací.

Uživatelské testování

Uživatelské testování se nejvíce blíží akceptačnímu testování. V tomto případě však netestují aplikaci klienti, ale zákazníci nebo uživatelé, kteří budou aplikaci používat. Ve velké části je toto testování používáno při vytváření nové aplikace nebo velké funkcionality. Uživatelské

testování hlavně slouží pro frontendovou část a testuje, zda je správně navrhnuto UX a UI. Například v případě vytváření aplikace pro online obchod je důležité toto testování zahrnout, protože většina uživatelů, co vstoupí na stránku obchodu, neví, jak mají s aplikací pracovat. Proto musí být uživatelské rozhraní intuitivní a lehce pochopitelné.

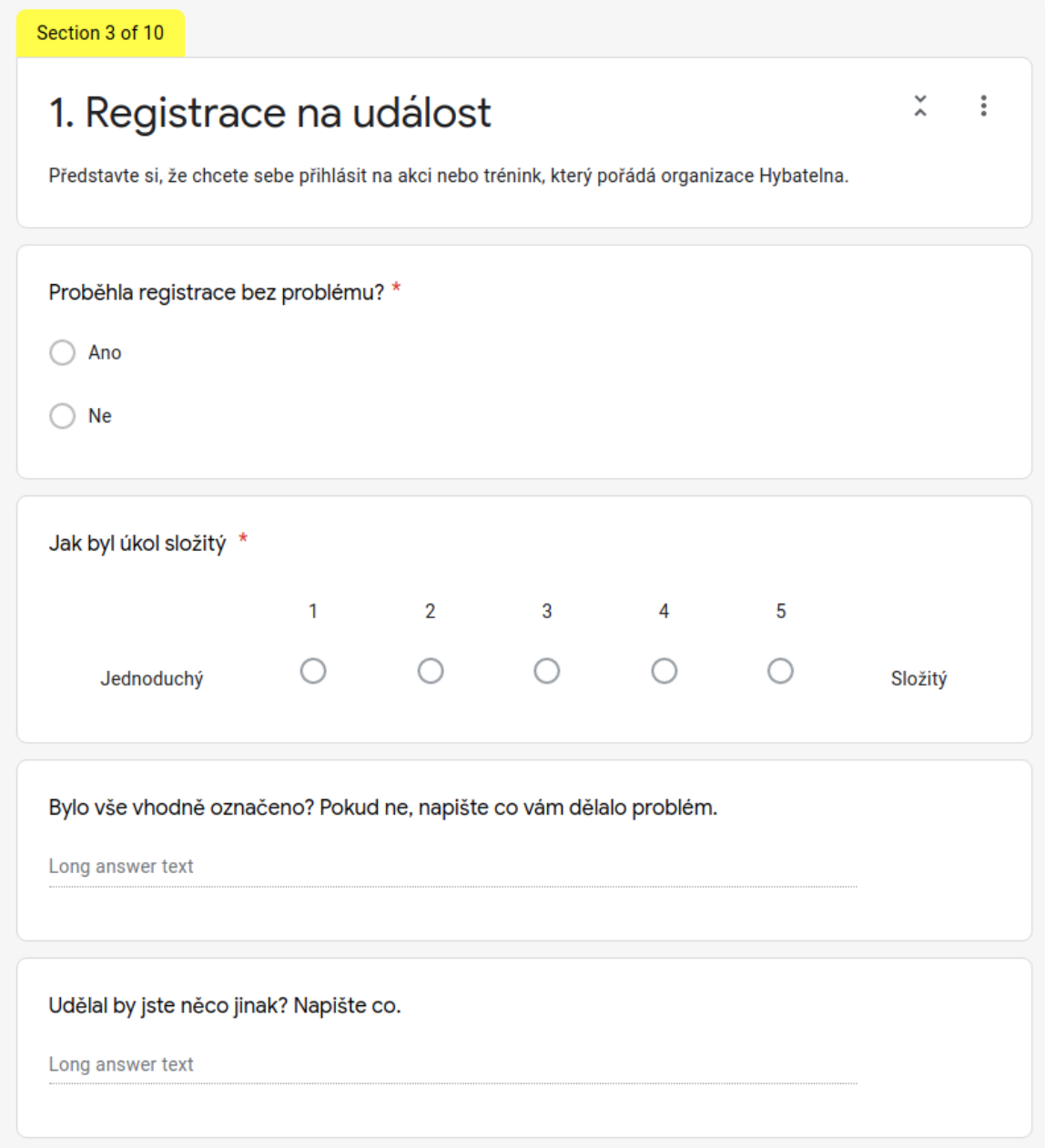
Před začátkem testování se vytváří scénáře a úkoly, které má uživatel splnit. Může se jednat o úkol vytvořit si uživatele. V uživatelském testování je důležité vhodně vytvářet úkoly. Rozdělit si uživatele na základě nějaké předdefinované osobnosti a vytvářet úkoly na míru této osobě. Dále je potřeba vytvářet úkoly tak, aby nepůsobily jako návod a neměly by obsahovat názvy sekcí v aplikaci. Pro příklad, aby se uživatel zaregistroval, tak úkol může být položen „Představte si, že se chcete dostat to aplikace“. V případě, kdy uživatel není registrován a vstup do aplikace je pouze pro zaregistrované uživatele, mělo by ho napadnout, že se musí zaregistrovat. Na základě tohoto uvědomění vyhledá sekci registrace a registruje se. Úspěšným testováním je pak rozuměno, že se zaregistroval bez problému.

7.1 Vytvoření scénářů

V rámci uživatelského testování bylo vytvořeno několik scénářů. Scénář pro rodiče, pro docházející, pro trenéry, scénář, který je rozšiřující pro docházejícího a pro manažery, který je rozšířením pro trenéra. Jelikož nejdůležitější uživatelé jsou rodiče a docházející, bylo potřeba se s testováním zaměřit více na ně, protože aplikaci budou používat často. Také rodiče nebo docházející budou používat aplikaci bez předchozích zkušeností s aplikací, tudíž bylo potřeba otestovat, zda normální uživatel dokáže všechny základní úkony. Další uživatelé jako trenéři a manažeři už budou muset projít seznámením s aplikací. Pro každou roli byly vybrány jak základní úkony v aplikaci, tak ty, které jsou hůře viditelné a zjistí, zda se uživatel dokáže na stránkách zorientovat. Scénáře byly navrhnuty tak, aby uživatelé, kteří budou chtít pracovat s aplikací v budoucnu, mohli někde zmínit své nápady.

Každá sekce formuláře zastává jeden úkol pro uživatele a následně otázky na dané úkoly, viz 12. První otázkou je vždy, zda se jim povedlo úkol vyřešit. Touto otázkou se zjišťuje počet uživatelů, kteří měli problém s úkolem a na základě toho se zjistí, které části jsou problémové. Další otázkou je výběr složitosti úkolu, kdy na základě této odpovědi lze zjistit, zda byl úkol složitý, nebo jestli si uživatel věděl rady. Pokud více uživatelům dělal problém tento úkol, bylo by vhodné lépe označit nebo zjednodušit proces. Třetí otázkou je, zda bylo vše správně označeno. V případě, kdy uživatel bude hledat dlouho nějaké tlačítko nebo formulář, lze to zmínit v této části a poté lépe označit nebo přesunout na lepší místo na stránce. Poslední otázka je na zlepšení, které uživatel může navrhnout. Uživatel může být zvyklý na podobný systém, kdy tlačítko pro odhlášení je na jiné pozici a může navrhnout změnu přesunutí.

Scénáře byly odeslány z velké části uživatelům v organizaci. Ti budou aplikaci sami používat, tudíž mají větší motivaci přicházet na funkcionality nebo hledat úpravy, které



Obrázek 12: Ukázka uživatelského scénáře

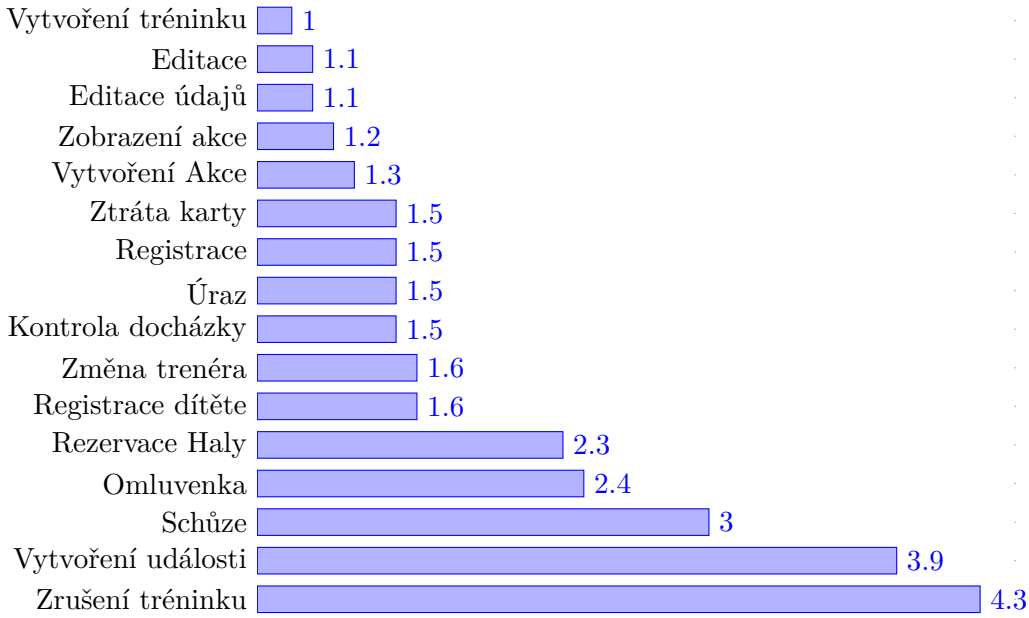
by jim usnadnily práci s aplikací. Tato skupina uživatelů pomůže zjistit, zda je aplikace přehledná a použitelná a v lepším případě navrhnout nějaké úpravy.

Dále byly scénáře odeslány uživatelům, kteří neví, k čemu stránky slouží a pomocí dotazníků zjistit, kde mají klasičtí uživatelé problém. Proto obě skupiny mají význam pro toto testování. Tato skupina uživatelů zase pomůže zjistit, zda se v aplikaci zorientují a umožní jim to splnit dané úkoly bez ohledu na jejich zkušenosti.

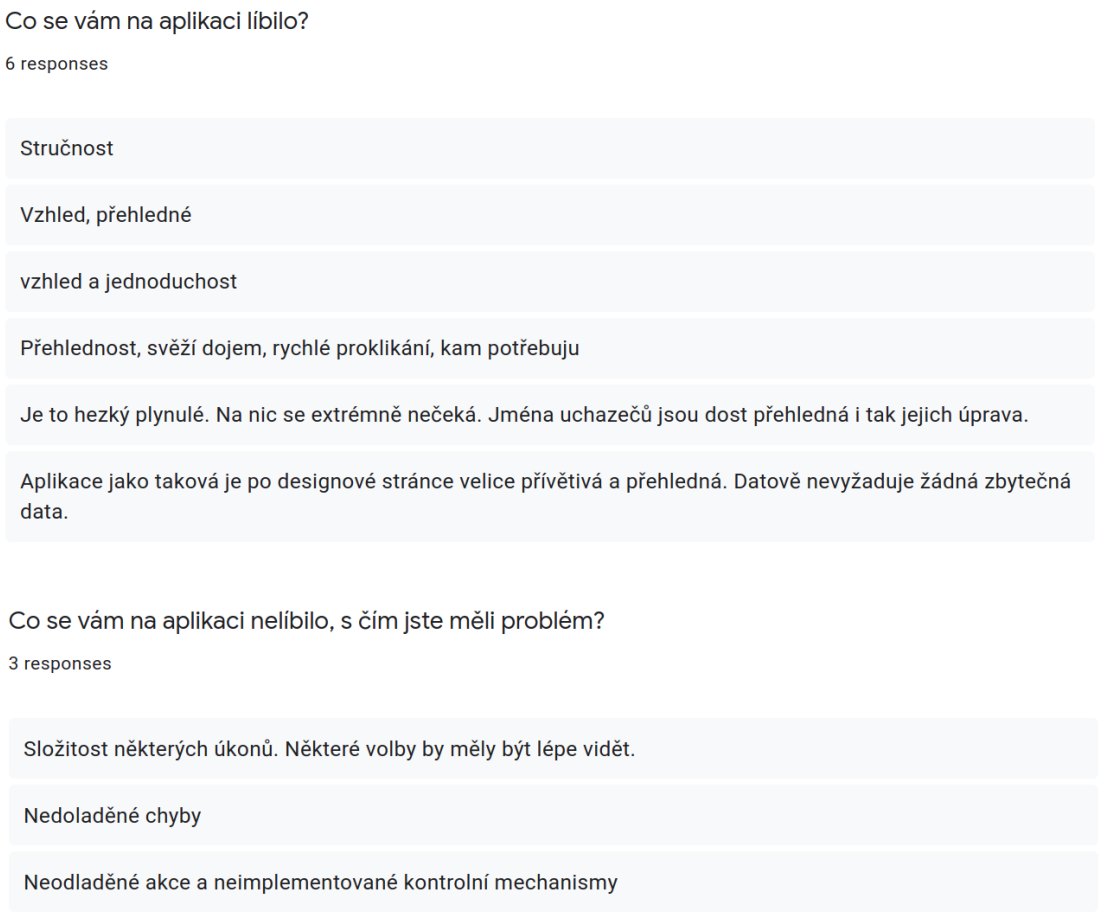
7.1.1 Výstup

Uživatelského testování se zúčastnilo 10 osob, kdy některé vyplňovaly více formulářů. Tím je myšleno, že budoucí manažer si vyzkoušel roli docházejícího i trenéra. Podle výpovědí mají respondenti zkušenosti s používáním webových stránek a věkové rozpětí je od 20 do 60 let. V grafu 2 lze vidět, jak obtížné byly úkoly, s čím měli uživatelé problém. Podle výpovědí byl největší problém s úkoly na kalendáři, kdy respondenti nevěděli, co mají dělat, nebo že tuto akci musejí provést v kalendáři.

Vesměs bylo hodnocení aplikace kladné a respondenti si chválili vzhled a jednoduché ovládání. Nebyli spokojeni s některými funkcionalitami, které buď nefungovaly podle očekávání, nebo byly jinde, než respondenti hledali. Při testování manažerů bylo těžší zorientovat se, které nastavení generování co dělá. Na základě těchto výpovědí bude provedena oprava některých částí aplikace, úprava rozložení komponent a vytvořena nápověda u jednotlivých stránek. Dále respondentům chyběly některé funkcionality jako zrušení omluvenky nebo tlačítka zpět v některých částech aplikace.



Graf 2: Přehled obtížnosti úkolů spočítán dle průměru z hodnocení všech respondentů od 1 do 5 (Čím nižší tím lepší)



Obrázek 13: Pozitivní a negativní hodnocení z uživatelského testování

8 Závěr

V aplikaci vzniklo vše podle přání klienta jako správa akcí a tréninků, kalendář, registrace a přihlášení přes služby třetích stran a další. Dle respondentů je aplikace zpracována hezky a není náročnou na naučení. Dále se do aplikace povedlo implementovat i import kalendáře v souborech .cal, který zatím slouží pouze pro administrátory pro import předpřipravených akcí nebo svátků. Povedlo se vytvořit generování událostí, které má velké možnosti, a je proto dobrým nástrojem při vytváření a správě. Přihlášení a registrace byly implementovány lépe, než bylo navrhnuto a nebyla nutnost přidávat další funkcionality pro tyto akce.

Aktuální stav aplikace není dokončen. Aplikace se bude dále vyvíjet a přibydou další funkcionality jako e-mailová komunikace, zjištění o zaplacení a finanční přehled pro manažery. Aktuálně je aplikace schopná pro nasazení a použití pro tréninky, ale neobsahuje funkcionality pro platby a vše, co respondenti navrhovali jako úpravu. Je však umožněna registrace uživatelů, kontrola a správa docházky.

Dále jako rozšíření budou vznikat notifikace, které budou chodit přihlášenému uživateli při přidání tréninku a akce nebo při změně docházky. Tato funkcionalita byla připravena a zatím je využita v sekci docházky, kdy je možno sledovat docházku živě jako, kdo přišel na událost a je mu posláno upozornění na sledovaném zařízení.

V aplikaci se nepovedlo vyřešit generování tréninků jednou za měsíc a vytvořit takový kalendář, který bude sloužit pro všechny uživatele a byl spolehlivým pomocníkem v organizaci. Chybí mu hodně funkcionalit a zatím je spíše určen pro manažery, kdy s ním mají více možností než ostatní uživatelé.

Aplikace je dobrým nástrojem pro správu docházky. Po vylepšení funkcionalit a opravy by byla schopna integrovat ve více takových organizacích. Na aplikaci bude autor dále pracovat a jeho cílem je, aby se aplikace stala plnohodnotným nástrojem pro management v organizacích.

Literatura

[1] Visualization of CAP theorem. Available from: https://www.researchgate.net/figure/Visualization-of-CAP-theorem_fig2_282679529

[2] Web Programming – The Object-Oriented Programming (OOP) Approach. Oct. 2020. Available from: https://articlesofdistinction.com/web-programming-the-object-oriented-programming-oop-approach/

[3] Writing Good User Stories. Dec. 2017. Available from: https://www.slideshare.net/EasyAgile/workshop-writing-good-user-stories

[4] React Docs. Available from: https://reactjs.org

[5] Vue.js Docs. Available from: https://vuejs.org

[6] Angular Docs. Available from: https://angular.io

[7] Different Types Of Attendance Tracking System. Available from: https://www.saralpaypack.com/blogs/attendance-tracking-system/

[8] Lofgren, L. Best Time and Attendance Systems. Mar. 2021. Available from: https://www.quicksprout.com/best-time-attendance-systems

[9] Alveno. Available from: https://www.alveno.cz/

[10] Aktion. Available from: https://www.aktion.cz/

[11] Bakaláři. Available from: https://www.bakalari.cz/

[12] Capan, T. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial. Available from: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[13] Watts, S. ACID: Atomic, Consistent, Isolated, & Durable. Apr. 2020. Available from: https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/

[14] Joshi, M. What Is the CAP Theorem and Where Is It Used? May 2019. Available from: https://medium.com/better-programming/what-is-the-cap-theorem-and-where-is-it-used-363475aa8db6

[15] CAP theorem: consistency, availability, and partition tolerance. Sept. 2020. Available from: https://www.ionos.com/digitalguide/server/know-how/what-is-cap-theorem/

[16] Codd, E. F. *Communications of the ACM*. June 1970.

[17] A Relational Database Overview. Available from: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

[18] History Of MySQL. Dec. 2019. Available from: https://www.youtube.com/watch?v=EIo1JFCWpfA&ab_channel=SpringFrameworkGuru

[19] Smallcombe, M. MariaDB vs MySQL: Everything You Need to Know. Sept. 2020. Available from: https://www.xplenty.com/blog/mariadb-vs-mysql-everything-you-need-to-know/

[20] Potter, J. What is LAMP stack. Feb. 2018. Available from: https://www.liquidweb.com/kb/what-is-a-lamp-stack/

[21] Krill, P. Oracle's ambitious plans for integrating Sun's technology. Jan. 2010. Available from: https://www.infoworld.com/article/2627785/oracle-s-ambitious-plans-for-integrating-sun-s-technology.html

[22] What Are Object-Oriented Databases And Their Advantages. Sept. 2019. Available from: https://www.c-sharpcorner.com/article/what-are-object-oriented-databases-and-their-advantages2/

[23] OBJECTIVITY/DB TECHNICAL DETAILS & SUPPORT. Available from: https://www.objectivity.com/products/objectivitydb/objectivitydbdetails/

[24] Schaefer, L. What is NoSQL? Available from: https://www.mongodb.com/nosql-explained

[25] Yangkatisal, M. The Battle of the NoSQL Databases - Comparing MongoDB and CouchDB. July 2020. Available from: https://severalnines.com/database-blog/battle-nosql-databases-comparing-mongodb-and-couchdb

[26] Most popular database technologies among developers worldwide as of January 2020. Available from: https://www.statista.com/statistics/794187/united-states-developer-survey-most-wanted-used-database-technologies/

[27] Shrivastava, A. Aspect Oriented Programming and AOP in Spring Framework. Mar. 2019. Available from: https://www.geeksforgeeks.org/aspect-oriented-programming-and-aop-in-spring-framework/

[28] Khan, M. Aspect Oriented Programming and AOP in Spring Framework. Sept. 2017. Available from: https://medium.com/@mustafakhansmt/imperative-vs-declarative-programming-106f99a7ffc5

[29] Bhatia, S. Procedural Programming [Definition]. Jan. 2021. Available from: https://hackr.io/blog/procedural-programming

[30] PHP Documentation. Available from: https://www.php.net/

[31] Skvorc, B. The Best PHP Framework for 2015: SitePoint Survey Results. Mar. 2015. Available from: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

[32] Grudl, D. Nette Documentation. Available from: https://doc.nette.org/cs/3.1/

[33] Laravel Docs. Available from: https://laravel.com/

[34] History of Java. Available from: https://www.javatpoint.com/history-of-java

[35] Johari, A. What is Java. Nov. 2020. Available from: https://www.edureka.co/blog/what-is-java/

[36] Features of Java. Available from: https://www.javatpoint.com/features-of-java

[37] INTRODUCTION AND HISTORY OF THE SPRING FRAMEWORK. June 2015. Available from: https://javajee.com/introduction-and-history-of-the-spring-framework

[38] Spring Documentation. Available from: https://spring.io/projects

[39] Python release. Available from: https://raw.githubusercontent.com/python/cpython/master/Misc/HISTORY

[40] Lekce 1 - Úvod do Pythonu. Available from: https://www.itnetwork.cz/python/zaklady/python-tutorial-uvod-do-pythonu-a-zakladni-matematicke-operace

[41] Why Django? Available from: https://www.djangoproject.com/start/overview/

[42] Buna, S. 10 popular modern JavaScript features for front-end devs. Mar. 2019. Available from: https://manifold.co/blog/10-popular-modern-javascript-features-for-front-end-devs

[43] Pros and Cons of JavaScript – Weigh them and Choose wisely! Available from: https://data-flair.training/blogs/advantages-disadvantages-javascript/

[44] Pandit, N. What and Why React.js. Jan. 2021. Available from: https://www.c-sharpcorner.com/article/what-and-why-reactjs/

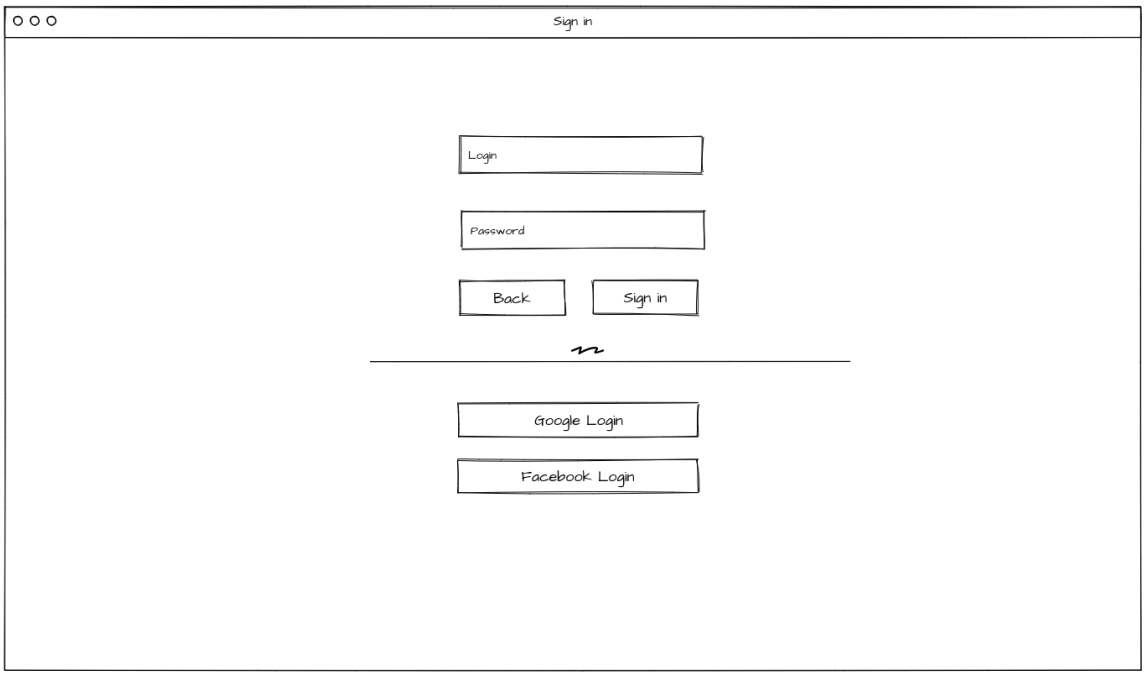
[45] The Good and the Bad of Vue.js Framework Programming. Sept. 2019. Available from: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

[46] A11Y project. Available from: https://www.a11yproject.com/about/

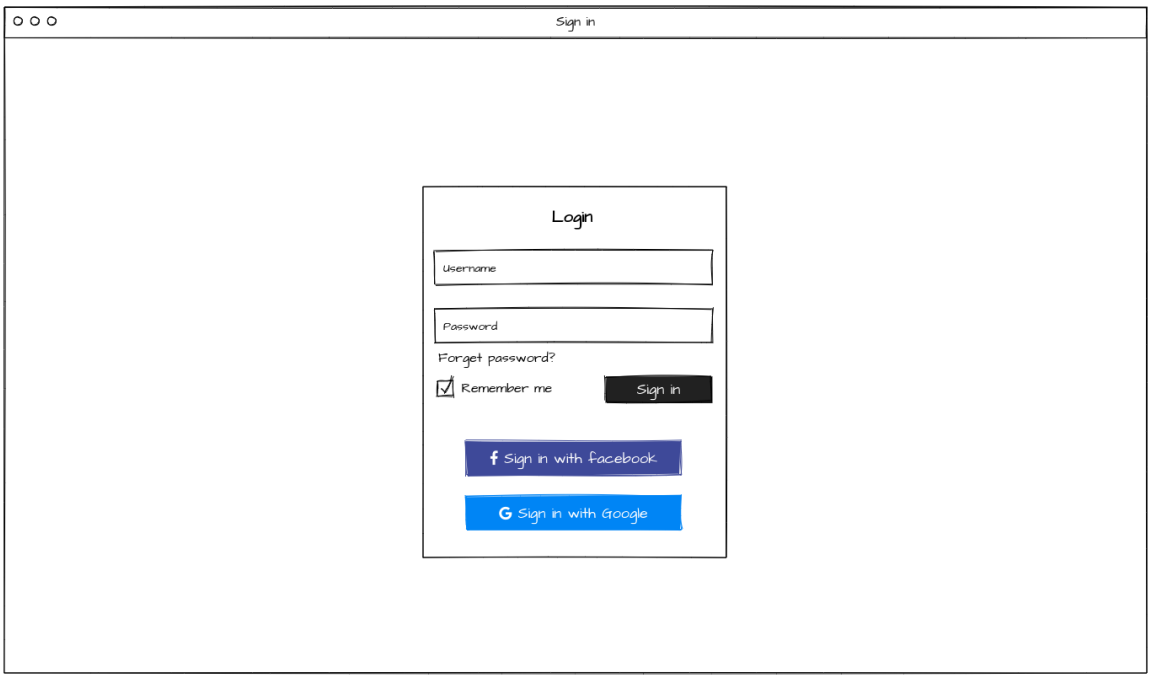
[47] Arateg. How to Choose the Technology Stack for Your Web Application. Mar. 2020. Available from: https://hackernoon.com/how-to-choose-the-technology-stack-for-your-web-application-hm4t3yte

Přílohy

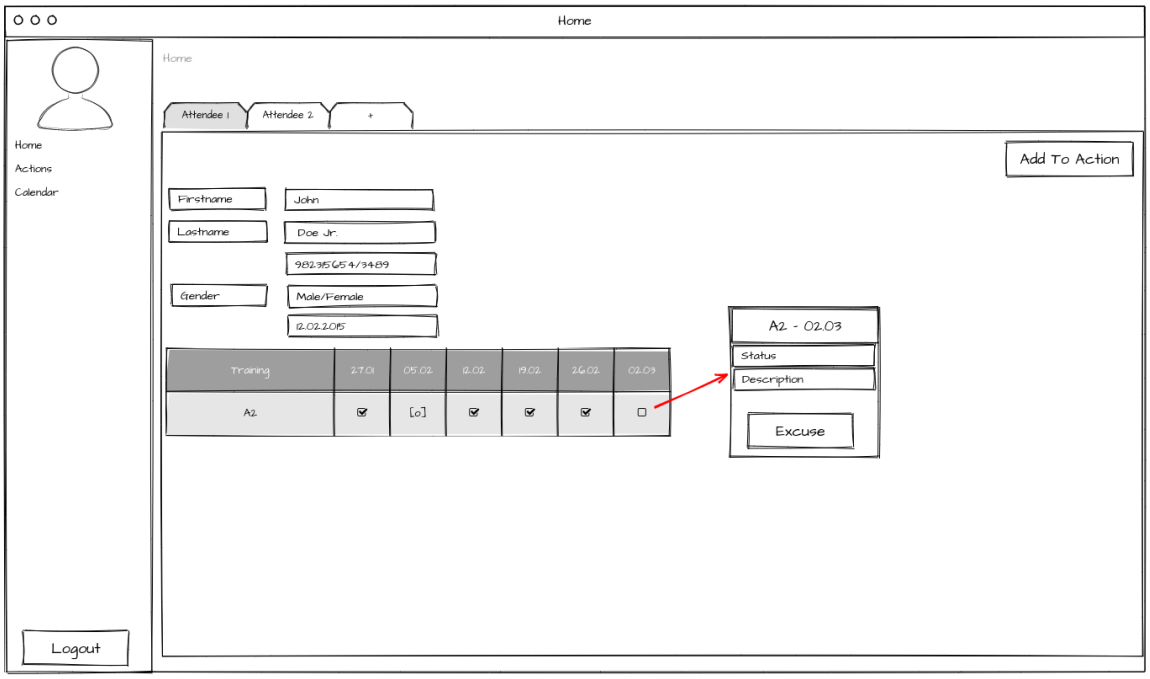
Wireframe



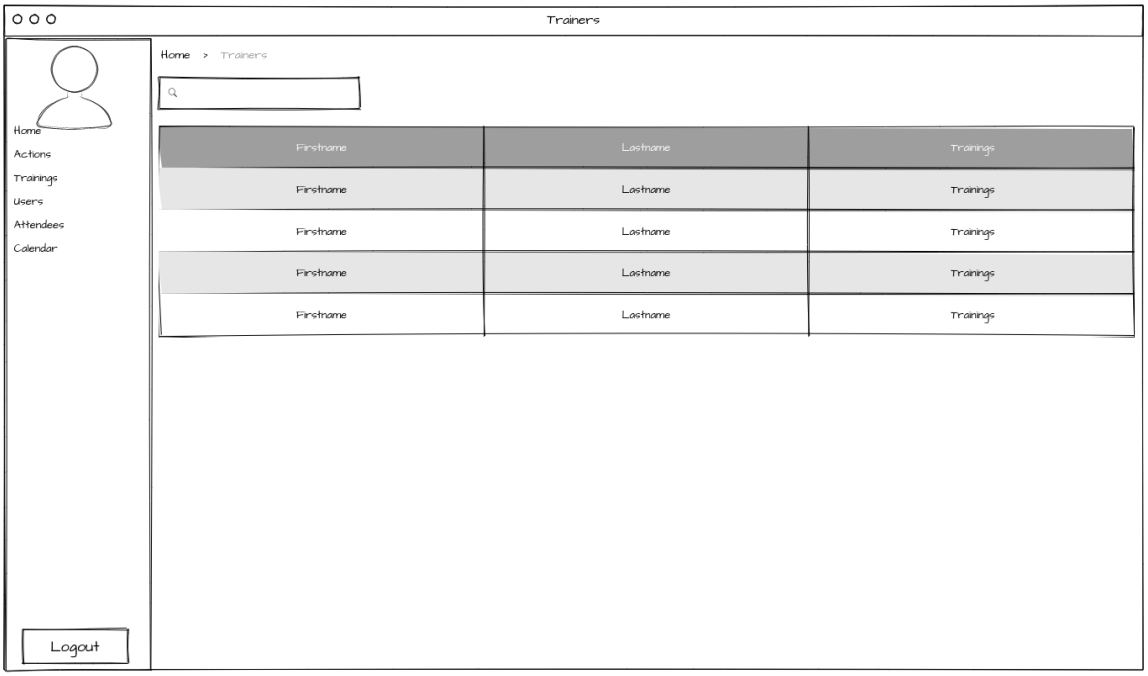
První návrh přihlášení



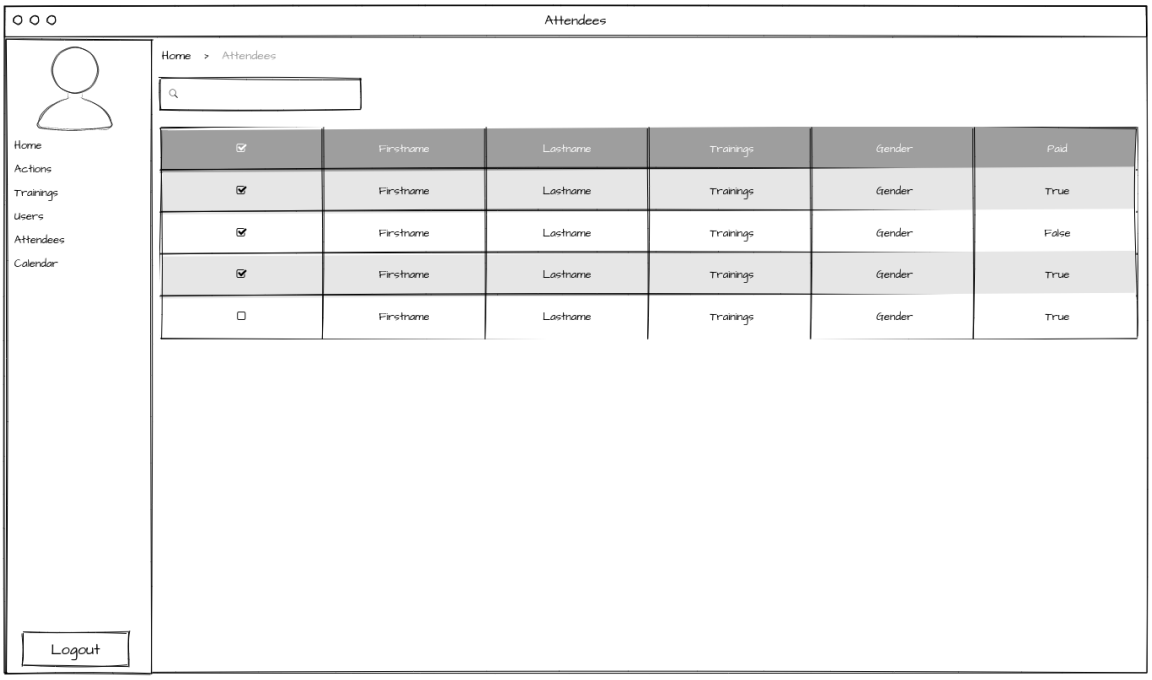
Druhý návrh přihlášení



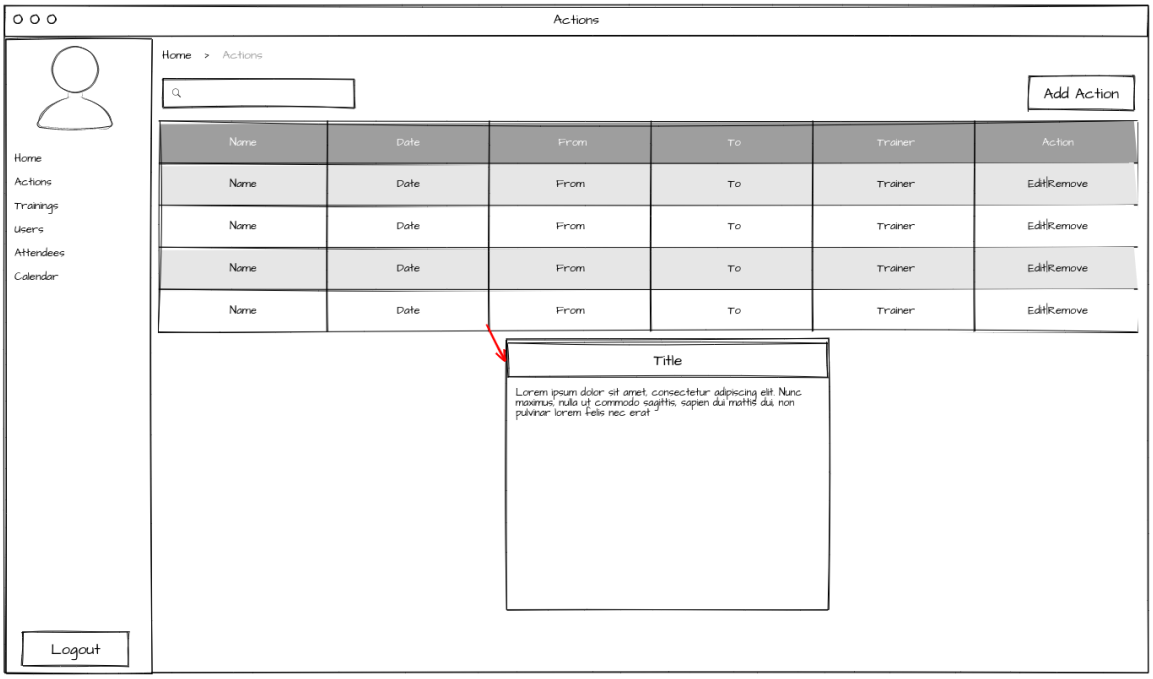
Domovská stránka



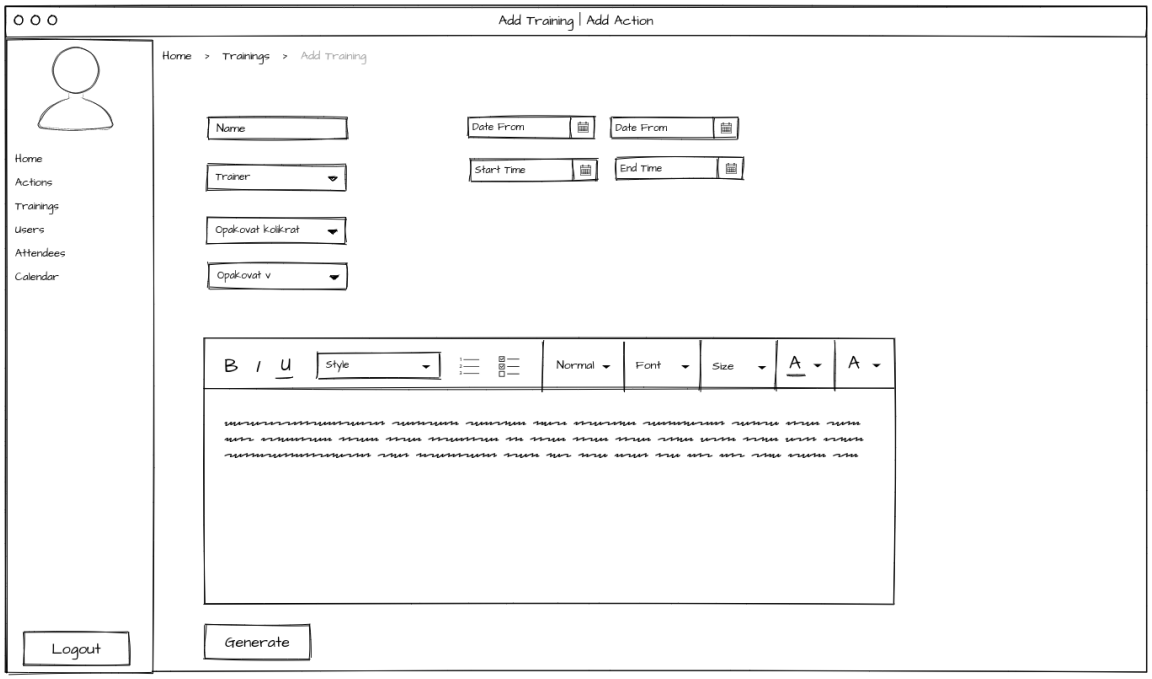
List trenérů



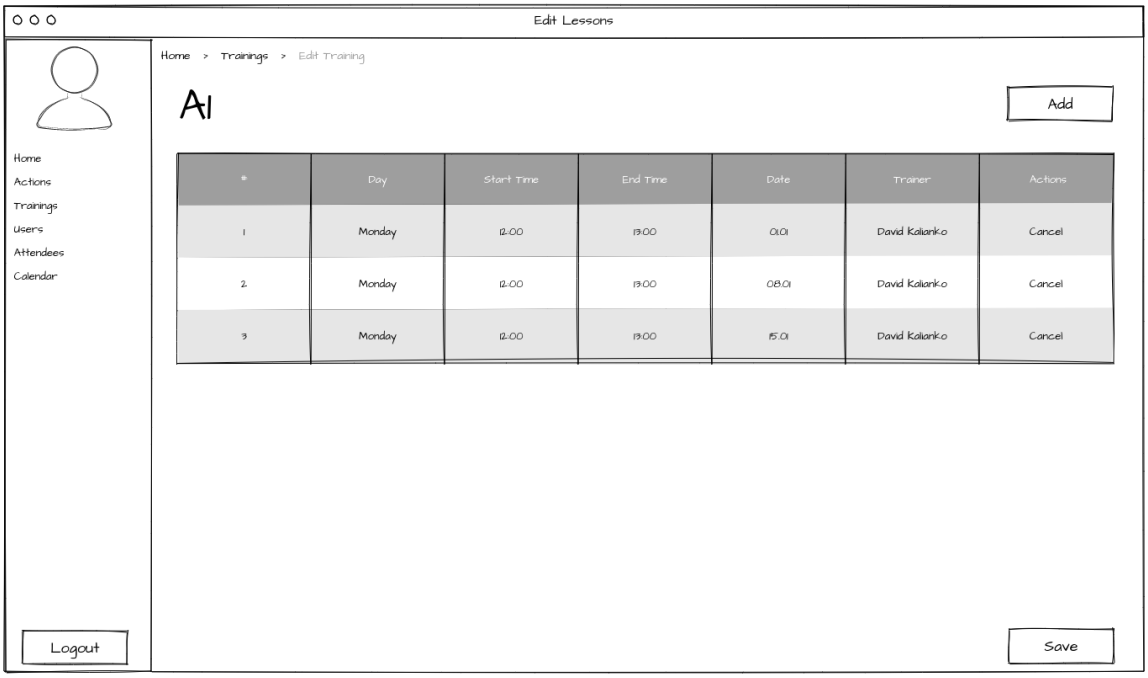
List docházejících



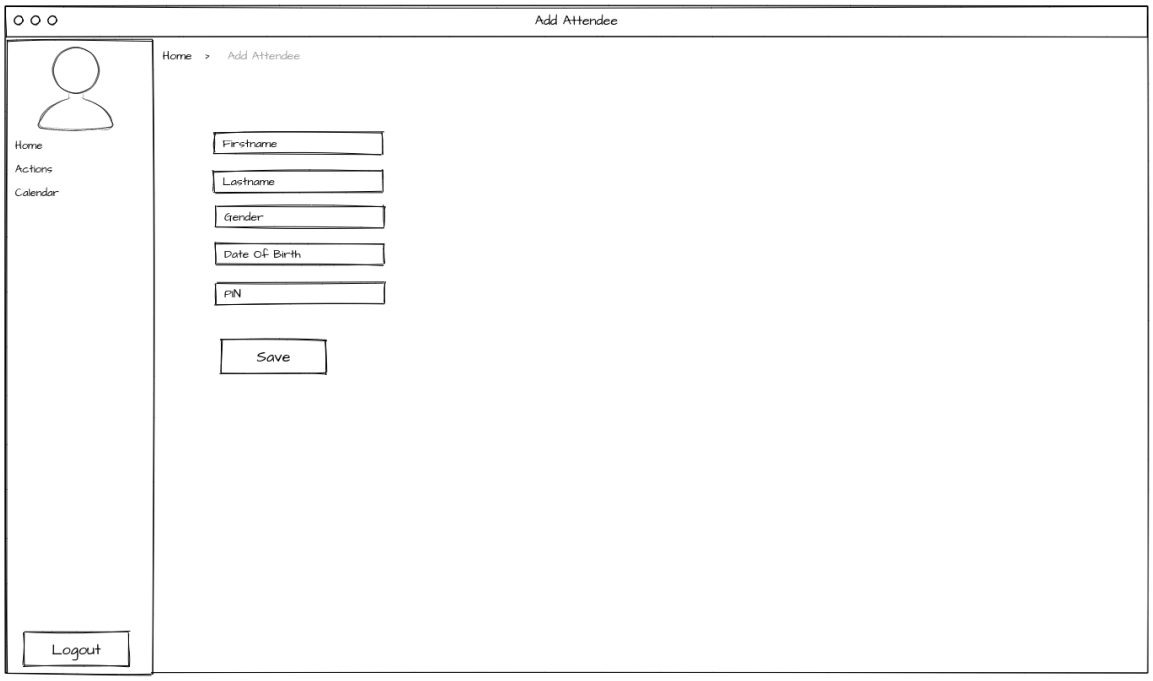
List akcí



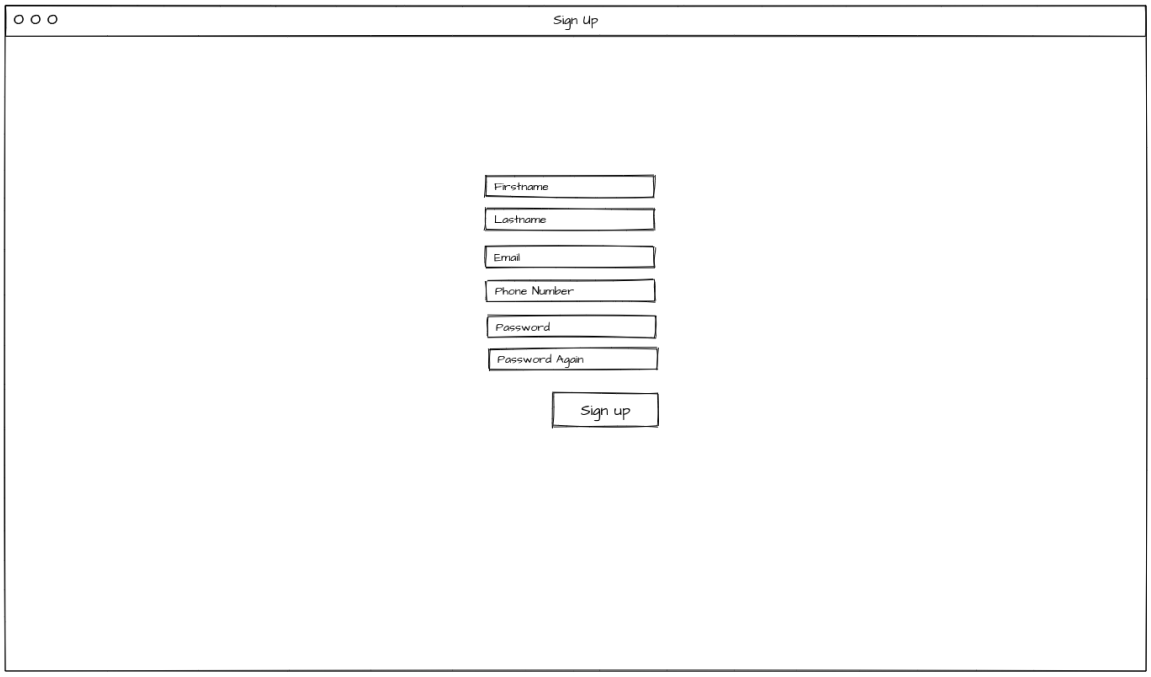
Formulář pro vytvoření události



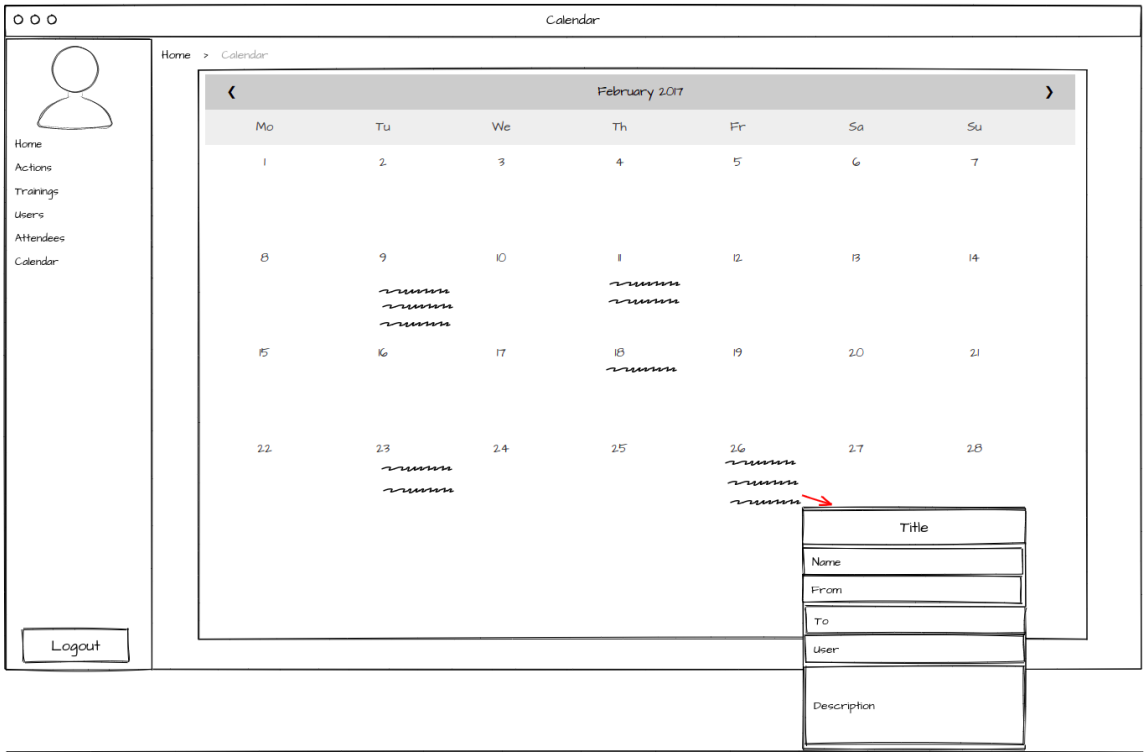
Editace termínů



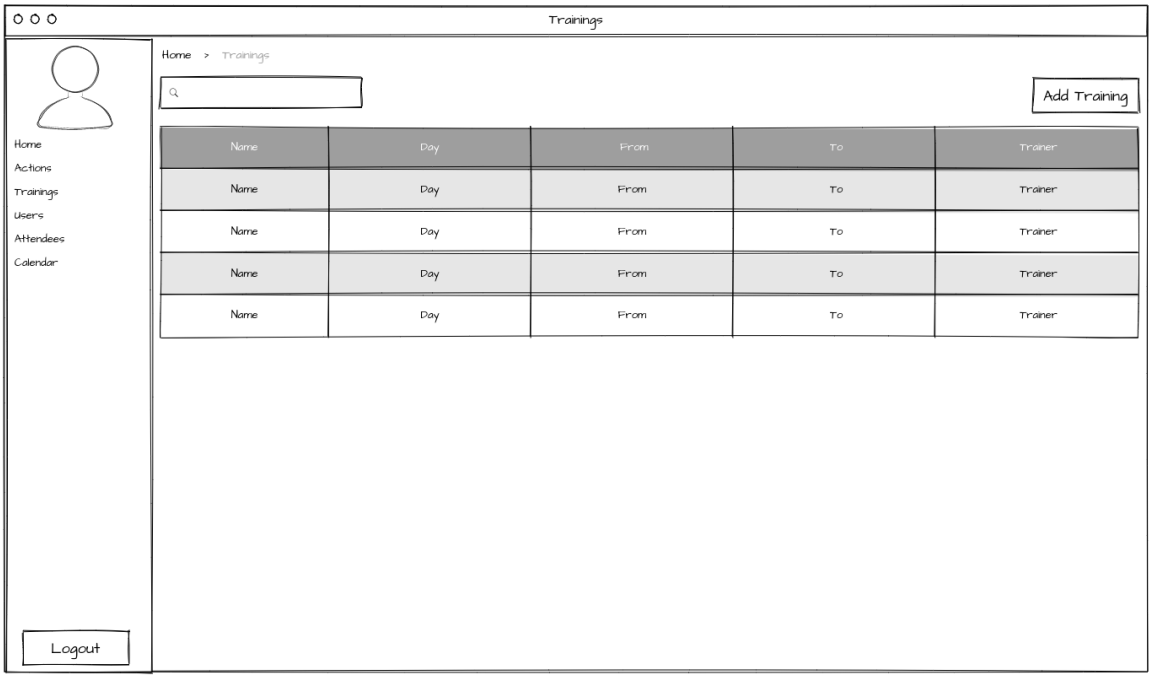
Formulář pro vytvoření docházejícího



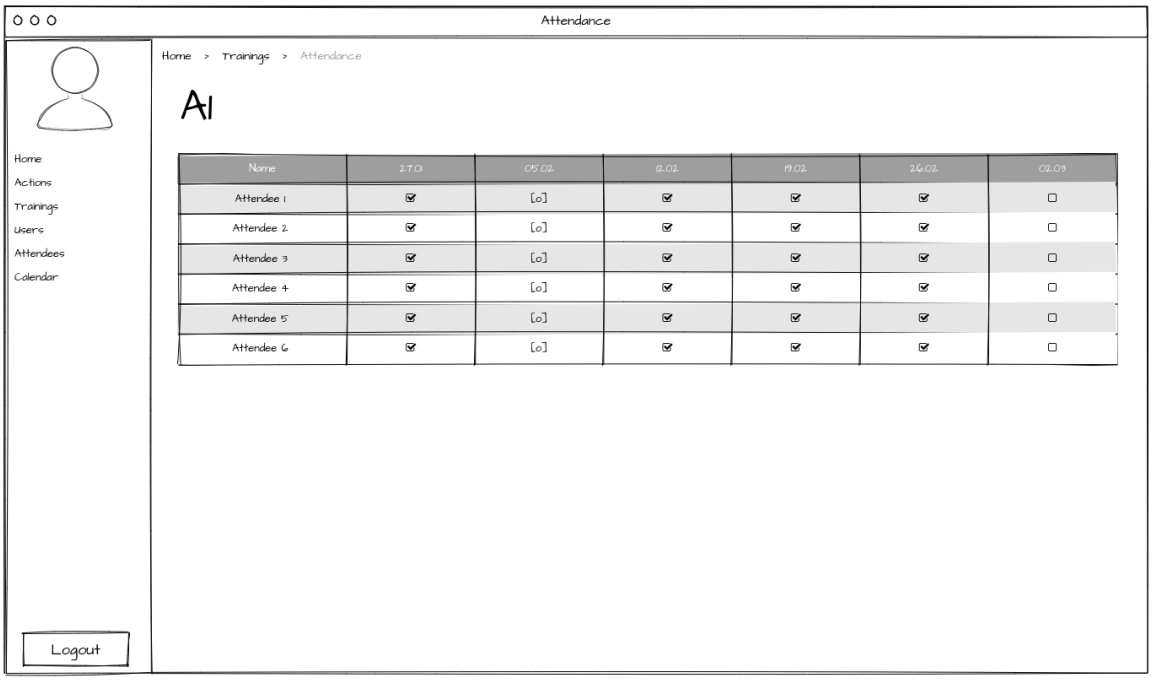
Formulář pro registraci



Kalendář



List tréninků


Attendance
Home > Trainings > Attendance
AI
Home
Actions
Trainings
Users
Attendees
Calendar
Logout
Name
27.01
05.02
12.02
19.02
26.02
02.03
Attendee 1
Attendee 2
Attendee 3
Attendee 4
Attendee 5
Attendee 6

Docházka

Uživatelské testování

Uživatelské testování – Docházející

Aplikace Stano je určena pro registraci docházejících a následnou kontrolu jejich docházky v organizaci Hybatelna.

Uživatelské testování umožňuje vyladit chyby a přehlednost celé aplikace. Aplikace je vytvářena v rámci bakalářské práce "Implementace webové aplikace pro docházkový systém." Dotazníky jsou anonymní a budou sloužit pouze jako podklad pro bakalářkou práci. V rámci testování aplikace není potřeba uvádět reálné informace, ale měly by reflektovat realitu, tzn. vkládat normální jména jako "David Neviděl" nebo smysluplné datum. Vložená data nebudou nikde dlouhodobě uchovávána.

Pro přístup k aplikaci využijte link: https://stano-fe.herokuapp.com

Jako docházející můžete kontrolovat a spravovat svoji docházku nebo docházku vašich dětí. Aplikace umožňuje sledovat nadcházející nebo právě probíhajících událostí.

Představte si, že se chcete účastnit pravidelných tréninků pro dospělé. Také by jste se rád/a účastnil nějaké akce pro pokročilé. Máte ještě dvě děti, které by se rádi účastnili tréninků. Jeden je mladší a má menší zkušenosti s parkourem. Druhý je starší a s parkourem má větší zkušenosti.

* Required

Informace o sobě

Doplňte informace o sobě, aby bylo možné data kategorizovat.

1. Kolik vám je let?

Mark only one oval.

- 15 - 20 let
- 20 - 30 let
- 30 - 40 let
- 50 - 60 let

2. Jaké máte zkušenosti s používáním webových stránek

Mark only one oval.

- Nepoužívám
- Trochu používám
- Používám často
- Každý den používám

1. Registrace na událost

Představte si, že chcete sebe přihlásit na akci nebo trénink, který pořádá organizace Hybatelna.

3. Proběhla registrace bez problému? *

Mark only one oval.

- Ano
- Ne

4. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

5. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

6. Udělal by jste něco jinak? Napište co.

2. Registrace dítěte

Představte si, že chcete svoje dítě/děti přihlásit na akci nebo trénink, který pořádá organizace Hybatelna.

7. Proběhla registrace bez problémů? *

Mark only one oval.

- Ano
- Ne

8. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

9. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

3. Omluvenka

Představte si, že onemocníte a chcete dát vědět o své absenci.

10. Dělalo vám problém najít omluvení? *

Mark only one oval.

- Ano
- Ne

11. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

12. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

13. Udělal by jste něco jinak? Napište co.

4. Zobrazení akcí

Představte si, že se chcete podívat na akce, které se budou v budoucnu konat.

14. Dělalo vám problém najít plánované akce? *

Mark only one oval.

- Ano
- Ne

15. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

16. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

17. Udělal by jste něco jinak? Napište co.

5. Úprava docházejícího

Představte si, že jste špatně vyplnili údaje o docházejícím

18. Proběhla úprava údajů bez problému? *

Mark only one oval.

- Ano
- Ne

19. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

20. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

21. Udělal by jste něco jinak? Napište co.

6. Úprava údajů

Představte si, že jste špatně vyplnili své údaje.

22. Proběhla úprava údajů bez problému? *

Mark only one oval.

- Ano
- Ne

23. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

24. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

25. Udělal by jste něco jinak? Napište co.

7. Události

Představte si, že si chcete poznamenat nějakou událost. Která se bude v budoucnu konat.

26. Proběhlo vytvoření události bez problému? *

Mark only one oval.

- Ano
- Ne

27. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

28. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

29. Udělal by jste něco jinak? Napište co.

Zhodnocení

Tato část je pro zhodnocení aplikace. Pokud máte nějaké nápady, či postřehy, které se nehodili do jednotlivých sekcí, můžete sepsat do této sekce.

30. Co se vám na aplikaci nelíbilo, s čím jste měli problém?

31. Co se vám na aplikaci líbilo?

32. Prostor pro další poznámky

Uživatelské testování – Trenér

Aplikace Stano je určena pro registraci docházejících a následnou kontrolu jejich docházky v organizaci Hybatelna.

Uživatelské testování umožňuje vyladit chyby a přehlednost celé aplikace. Aplikace je vytvářena v rámci bakalářské práce "Implementace webové aplikace pro docházkový systém." Dotazníky jsou anonymní a budou sloužit pouze jako podklad pro bakalářkou práci. V rámci testování aplikace není potřeba uvádět reálné informace, ale měly by reflektovat realitu, tzn. vkládat normální jména jako "David Neviděl" nebo smysluplné datum. Vložená data nebudou nikde dlouhodobě uchovávána.

Pro přístup k aplikaci využijte link: https://stano-fe.herokuapp.com

Jako trenér můžete kontrolovat a spravovat svoji docházku. Jako trenér máte na starosti trénink, který spravujete. Máte k dispozici seznam uživatelů a docházejících.

Představte si, že jste trenérem , který má na starosti skupinu A1 a B2.

* Required

Informace o sobě

Doplňte informace o sobě, aby bylo možné data kategorizovat.

1. Kolik vám je let?

Mark only one oval.

- 15 - 20 let
- 20 - 30 let
- 30 - 40 let
- 50 - 60 let

2. Jaké máte zkušenosti s používáním webových stránek

Mark only one oval.

- Nepoužívám
- Trochu používám
- Používám často
- Každý den používám

1. Kontrola docházky

Představte si, že chcete zkontrolovat docházku.

3. Našel jste docházku bez problémů *

Mark only one oval.

- Ano
- Ne

4. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

5. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

6. Udělal by jste něco jinak? Napište co.

2. Úraz na tréninku

Představte si, že se jednomu z dětí něco stane na tréninku a je potřeba kontaktovat zákonného zástupce.

7. Byl kontakt nalezen bez problému? *

Mark only one oval.

- Ano
- Ne

8. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

9. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

10. Udělal by jste něco jinak? Napište co.

3. Ztráta karty

Představte si, že docházející ztratil kartu pro zaznamenání docházky a chce novou.

11. Dělalo vám problém vyřešit tuto situaci? *

Mark only one oval.

- Ano
- Ne

12. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

13. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

14. Udělal by jste něco jinak? Napište co.

Zhodnocení

Tato část je pro zhodnocení aplikace. Pokud máte nějaké nápady, či postřehy, které se nehodili do jednotlivých sekcí, můžete sepsat do této sekce.

15. Co se vám na aplikaci nelíbilo, s čím jste měli problém?

16. Co se vám na aplikaci líbilo?

17. Prostor pro další poznámky

Uživatelské testování – Manažer

Aplikace Stano je určena pro registraci docházejících a následnou kontrolu jejich docházky v organizaci Hybatelna.

Uživatelské testování umožňuje vyladit chyby a přehlednost celé aplikace. Aplikace je vytvářena v rámci bakalářské práce "Implementace webové aplikace pro docházkový systém." Dotazníky jsou anonymní a budou sloužit pouze jako podklad pro bakalářkou práci. V rámci testování aplikace není potřeba uvádět reálné informace, ale měly by reflektovat realitu, tzn. vkládat normální jména jako "David Neviděl" nebo smysluplné datum. Vložená data nebudou nikde dlouhodobě uchovávána.

Pro přístup k aplikaci využijte link: https://stano-fe.herokuapp.com

Jako manažer můžete kontrolovat a spravovat svoji docházku. Můžete mít na starosti trénink, který spravujete. Máte k dispozici seznam uživatelů a docházejících. Jako manažer můžete spravovat uživatele a akce v Hybatelně, jako vytvořit, publikovat, upravit. Můžete vytvářet veřejné události.

* Required

Informace o sobě

Doplňte informace o sobě, aby bylo možné data kategorizovat.

1. Kolik vám je let?

Mark only one oval.

- ◯ 15 - 20 let
- ◯ 20 - 30 let
- ◯ 30 - 40 let
- ◯ 50 - 60 let

2. Jaké máte zkušenosti s používáním webových stránek

Mark only one oval.

- ◯ Nepoužívám
- ◯ Trochu používám
- ◯ Používám často
- ◯ Každý den používám

1. Vytvoření akce

Představte si, že chcete vytvořit jednorázovou akci.

3. Byla akce vytvořena bez problému? *

Mark only one oval.

- Ano
- Ne

4. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

5. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

6. Udělal by jste něco jinak? Napište co.

2. Zrušení tréninku

Představte si, že nastalo neočekávané zrušení tréninků v jeden den.

7. Proběhlo zrušení bez problému? *

Mark only one oval.

- Ano
- Ne

8. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

9. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

10. Udělal by jste něco jinak? Napište co.

3. Vytvoření tréninku

Představte si, že chcete vytvořit nový pravidelný trénink a zveřejnit ho.

11. Proběhlo vytvoření tréninku bez problémů? *

Mark only one oval.

- Ano
- Ne

12. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

13. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

14. Udělal by jste něco jinak? Napište co.

4. Změna trenéra

Představte si, že v průběhu tréninkového období potřebujete změnit trenéra.

15. Proběhla změna trenéra bez problémů? *

Mark only one oval.

- Ano
- Ne

16. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

17. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

18. Udělal by jste něco jinak? Napište co.

5. Zarezervování haly

Představte si, že vás někdo požádá o zarezervování haly pro soukromou akci a je potřeba vyhradit tomu volný prostor v hale.

19. Proběhlo zarezervování bez problémů? *

Mark only one oval.

- Ano
- Ne

20. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

21. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

22. Udělal by jste něco jinak? Napište co.

6. Schůze

Představte si, že chcete udělat schůzi a je potřeba o tom dát vědět pouze lidem v organizaci.

23. Proběhla akce bez problémů? *

Mark only one oval.

- Ano
- Ne

24. Jak byl úkol složitý *

Mark only one oval.

	1	2	3	4	5	
Jednoduchý						Složitý

25. Bylo vše vhodně označeno? Pokud ne, napište co vám dělalo problém.

26. Udělal by jste něco jinak? Napište co.

Zhodnocení

Tato část je pro zhodnocení aplikace. Pokud máte nějaké nápady, či postřehy, které se nehodili do jednotlivých sekcí, můžete sepsat do této sekce.

27. Co se vám na aplikaci nelíbilo, s čím jste měli problém?

28. Co se vám na aplikaci líbilo?

29. Prostor pro další poznámky

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **David Kalianko**

Studium: I1800179

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Implementace webové aplikaci pro docházkový systém**

Název bakalářské práce AJ: Implementation of a Web Application for Attendance System

Cíl, metody, literatura, předpoklady:

Cíl práce: Cílem bakalářské práce je analýza procesů firmy, rešerše existujících docházkových systému a implementace docházkového systému pro externího zákazníka.

Osnova:

- Úvod
- Rešerše docházkových systémů
- Analýza firemních procesů
- Analýza technologií pro vývoj webových aplikací
- Návrh architektury aplikace
- Implementace
- Testování
- Závěr

Garantující pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Ing. Michal Macinka

Oponent: doc. Mgr. Tomáš Kozel, Ph.D.

Datum zadání závěrečné práce: 14.1.2020