ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Systém pro podporu dotazníků pro výzkumný projekt

Bakalářská práce

Patrik Pavelka

Květen 2023
Vedoucí práce: doc. Ing. Miroslav Bureš, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pavelka** Jméno: **Patrik** Osobní číslo: **492087**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro podporu dotazníků pro výzkumný projekt

Název bakalářské práce anglicky:

Questionnaire support system for a research project

Pokyny pro vypracování:

Navrhněte a implementujte systém pro podporu dotazníků zodpovídaných účastníky studií v rámci projektu vyzkumodolnosti.cz řešeného na FVZ UNOB. Systém umožní generování dotazníků pro účastníky lékařských studií. V dotaznících bude odpovídající validace vstupů dle zadání školitele a ze získaných dat bude navržený systém vypočítávat další údaje pro zpracování v rámci studií. Systém implementujte jako webovou aplikaci. Správnou funkčnost systému ověřte sadou vhodných uživatelských testů.

Seznam doporučené literatury:

J. Nema, J. Zdara, P. Lasak, J. Bavlovic, M. Bures, J. Pejchal and H. Schvach. Impact of cold exposure on life satisfaction and physical composition of soldiers. BMJ Military Health, 2023.
Herbert Schildt: Mistrovství Java: Kompletní průvodce vývojáře. Computer Press, 2017.
Michele Riva, Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production. Packt Publishing, 1st edition, 2022.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Miroslav Bureš, Ph.D. laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

doc. Ing. Miroslav Bureš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Rád bych poděkoval doc. Ing. Miroslavu Burešovi, Ph.D., za cenné rady, připomínky a vstřícnost při konzultacích během vypracovávání bakalářské práce.

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

V Praze dne

.....

Abstrakt / Abstract

Tato práce je zaměřena na vytvoření uživatelského rozhraní a API pro sběr dat z dotazníků o spánku na projektu Výzkum Odolnosti (organismu), který svým účastníkům nabízí odborný pohled na jejich kvalitu a režim spánku.

Součástí práce je sběr požadavků od strany Univerzity Obrany, jejich zpracování do softwarové analýzy včetně funkčních a nefunkčních požadavků, popsaná architektura a datový model pro implementaci a samotná implementace a následné testování systému a běh v provozu. Data slouží pro další zpracování a to ve formě informování uživatelů o jejich spánku za určité časové období. Systém také ukládá záznamy z nositelné elektroniky, které se dají spárovat s daty z dotazníků. Aplikace slouží jako nástroj pro provádění studie o spánku.

Klíčová slova: spánek, dotazník, výzkum odolnosti, kvalita spánku

This work is focused on the creation of a user interface and API for collecting data from sleep questionnaires on the Vyzkum Odolnosti (Resilience Research) project, which offers its participants an expert view of their sleep quality and mode.

Part of the work is the collection of requirements from the Resilience Research side, their processing into software analysis including functional and non-functional requirements, the described architecture and data model for implementation and the implementation itself and subsequent testing of the system and running in operation. The data is used for further processing in the form of informing users about their sleep over a certain period of time. The system also stores records from wearable electronics that can be matched with data from questionnaires. The app serves as a tool for conducting a sleep study.

Keywords: sleep, questionnaire, endurance research, sleep quality

Title translation: Questionnaire support system for a research project (Bachelor's thesis)

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Výzkum Odolnosti je projekt, který svým účastníkům nabízí řadu metod, kterými si mohou zvýšit výkon svého těla a mysli. Účastníci výzkumu poskytují informace o vlastních výsledcích z probíhajících metod prostřednictvím vyplňování dotazníků či měření spánku nositelnou elektronikou. Výzkum Odolnosti vznikl na Univerzitě Obrany (UNOB) na Fakultě vojenského zdravotnictví. Na projektu se začalo České vysoké učení technické v Praze (ČVUT) na Fakultě Elektrotechnické (FEL) podílet v roce 2022. Cílem této spolupráce je vyvinout snadno spravovatelný systém pro sběr a zpracování sesbíraných dat z nositelné elektroniky a z vyplněných dotazníků účastníků výzkumu. Tato konkrétní práce se zabývá části dotazníků.

1.1 Již existující část řešení

V roce 2022 vznikl systém pro sběr dat z nositelných zařízení Garmin od účastníků výzkumu. Tato část je momentálně již v pilotním běhu a v době psaní této práce bylo sesbíráno již řádově tisíce záznamů o spánku. Účastníci výzkumu poskytují naší aplikaci souhlas ke sběru dat z hodinek prostřednictvím aplikace Garminu. Z nasbíraných dat si může hlavní řešitel studie generovat exporty a pozorovat data o spánku jednotlivých účastníků výzkumu. Vzniklo pro to administrátorské rozhraní a API pro sběr dat. V administračním si může výzkumník vybrat časové rozmezí, ze kterého chce vytvořit export spánku. Tento systém je pro další práci významný, neboť je v této práci rozšiřován o nové funkcionality.

1.2 Cíle této práce

Účastníci výzkumu bývají zpravidla studenti Univerzity obrany a jsou to tedy lidi, kteří se budou zabývat obranou České republiky. Motivací je této skupině lidí pomoct k zesílení mentálního i fyzického stavu za pomocí systémového řešení pro projekt Výzkumu Odolnosti. Bude vytvořena nová část existujícího řešení, která řešitelům projektu pomůže snadněji sledovat výsledky svých účastníků v jejich snažení. Účastníci budou využívat moderní webovou aplikaci s intuitivním rozhraním k poskytování informací o jejich spánku. Je potřeba rozvinout existující aplikaci právě o část s dotazníky, aby se k výzkumníkům tyto informace o účastnících dostaly.

V databázi přibude prostor pro záznamy vyplněných dotazníků od respondentů. Budeme potřebovat uživatelské rozhraní pro vyplňování dotazníků a webové API, která tato data bude od respondentů přijímat, zpracovávat a vkládat do databáze.

Cílem této práce je sesbírat požadavky pro rozšíření existujícího řešení, zanalyzovat a navrhnout další řešení. Návrh řešení bude dále implementován, budou otestovány části řešení jak automaticky, tak manuálně uživateli a v neposlední řadě bude systém nasazen do produkčního prostředí.

Kapitola 2
Analýza a návrh

V této kapitole se podíváme na analýzu a návrh systému, které vznikly na základě konzultací s řešiteli výzkumu. Analýza nám napomůže si ujasnit, jak následně systém implementovat. V kapitole naleznete analytický slovník, funkční a nefunkční požadavky nebo případy užití.

2.1 Analytický slovník

V této sekci se podíváme na pojmy vzniklé během sběru požadavků. Byly takto zvoleny, aby byla zachována konzistence napříč dokumentací a aby si dobře rozuměly obě strany během konzultacích a sběru a modelování požadavků. Slovník byl definován pro potřeby komunikace ve fázi sběru požadavků a definované pojmy se nutně nemusí dále objevovat v této práci.

- **Číselný výsledek dotazníku**
  Je číselná hodnota vypočtena na základě vzorců definovaných předpisy pro jednotlivé dotazníky. Jeden dotazník může mít jeden až několik číselných výsledků.
- **Stručný výpis z dotazníku**
  Je takový výpis, který obsahuje pouze základní informace, kterými jsou identifikace respondenta a vypočtené číselné výsledky.
- **Detailní výpis z dotazníku**
  Je takový výpis, který obsahuje veškeré informace o odeslaném dotazníku, tedy všechny informace ze stručného výpisu z dotazníku a navíc odpovědi na všechny otázky.
- **Otázky nutné pro výpočet**
  Jsou otázky, podle kterých vypočítáváme číselný výsledek dotazníku.
- **Identifikující otázky**
  Jsou otázky, za pomocí kterých se respondent identifikuje.
- **Doplňkové otázky**
  Doplňkové otázky jsou takové otázky, které nejsou potřeba pro výpočet ani identifikaci respondenta. Slouží k získání doplňujících informací pro výzkumníka. Otázka může být např. čemu se respondent poslední měsíc věnoval.
- **Ranní typ**
  Také ranní ptáče nebo skřivan. Člověk, kterému vyhovuje vstávat spíše v dřívějších ranních hodinách.
- **Večerní typ**
  Také sova. Člověk, kterému vyhovuje chodit spát spíše v pozdějších hodinách.

2.2 Typy dotazníků

V současné době se na Výzkumu Odolnosti pracuje s dotazníky vypsanými v této podkapitole. Dotazníky, které se zdají, že nesouvisí se spánkem, jsou využívány proto, aby byla nalezena korelace mezi např. životní spokojeností a kvalitou spánku respondenta.

- **PSS**

  Škála vnímaného stresu (Perceived Stress Scale, PSS) je dotazník, za pomocí kterého můžeme zjistit, jak respondent vnímá svůj stres. Dotazník PSS byl vyvinut Sheldonem Cohenem a jeho kolegy z Carnegie Mellon University.

  PSS se skládá ze 14 otázek, které hodnotí, jak často jedinec zažívá běžné příznaky stresu, jako je pocit přetížení, napětí nebo neschopnost zvládat důležité věci ve svém životě. Odpovědi se hodnotí na pětibodové škále od `nikdy` po `velmi často`.

  Ve Výzkumu Odolnosti vyplňují respondenti zkrácenou verzi dotazníku PSS, která je známa jako PSS10. Ta se skládá pouze z 10 otázek původního PSS (PSS14). Je zde používán k hledání vztahů mezi úrovní pociťovaného stresu a kvalitou spánku měřenou PSQI [1] [2].

- **MEQ**

  Morningness-Eveningness Questionnaire (MEQ) je dotazník, který se používá k posouzení chronotypu jedince. Dotazník byl vyvinut vědci Hornem a Ostbergem a slouží k vyhodnocení spánkových vzorců jedinců.

  Dotazník MEQ se skládá z 19 otázek, jako jsou kdy jedinec obvykle chodí spát, kdy se obvykle probouzí, jak dlouho mu trvá usnout nebo jaké hodiny ve dni považují za pro ně nejproduktivnější. Na základě odpovědí jsou jedinci klasifikováni jako ranní typ, nebo večerní typ, a podle jejich preference jsou v rámci Výzkumu Odolnosti zařazování do jednotlivých skupin. Tato znalost umožňuje individualizovat nastavení výzkumu podle chronotypu jedince [3] [4] [5].

- **MCTQ**

  Mnichovský dotazník chronotypu, The Munich Chronotype Questionnaire (MCTQ) je dotazník vytvořen Till Roennebergovou a Marthou Merrowk, který se podobně jako dotazník MEQ používá k posouzení chronotypu (preferovaného vzorce spánku) jedince. Je navíc doplněn otázky týkajících se aktivit ovlivňující spánek, jako jsou výskyt na denním světle, způsob dopravování do práce, či zda respondent kouří nebo pije alkohol). Dotazník se dohromady skládá ze 17 položek, které jsou děleny na návyky v pracovních a nepracovních dnech. Díky tomuto rozdělení můžeme identifikovat tzv. sociální jetlag. Pro Výzkum Odolnosti je dotazník významný jako doplněk dotazníku MEQ, který hodnotí cirkadiánní preference respondenta na základě reálného chování a při změně jeho návyků je schopen reflektovat jemné nuance [6] [7] [8].

- **Dotazník životní spokojenosti**

  Dotazník životní spokojenosti (Life Satisfaction Questionnaire, LSQ) vyvinul výzkumník Jochen Fahrenberg. Hodnotí celkovou životní spokojenost jedince. V LSQ se můžeme setkat s 10 otázkami z oblastí spokojenosti ohledně zdraví, zaměstnání, finanční situace, volného času, partnerství, vztahu k vlastním dětem, vztahu k vlastní osobě, sexualitě, svých přátel a bydlení.

  Ve Výzkumu odolnosti je dotazník použit k hodnocení subjektivního vnímání celkové či dílčí životní spokojenosti v průběhu působení výzkumných metod. Je tak z důvodu prokázání pozitivních vlastností jednotlivých výzkumných metod nebo jako indikace případných externalit, které mohou ovlivnit vnímání respondenta (ukončení vztahu, či nemoc respondenta) [9].

- **PSQI**

  Pittsburgh Sleep Quality Index (PSQI) je dotazník vyvinutý výzkumníky na Pittsburské univerzitě, který hodnotí kvalitu spánku jedince. PSQI umí hodnotit různé aspekty spánku, včetně latence, délky trvání a poruch spánku [10].

2.3 Stávající fungování projektu

V současné době se využívají pro sběr dat z dotazníku Microsoft Forms, kde výzkumník zadává otázky a definuje u nich možné vstupy. Neexistuje u nich validace vstupu a respondent u otevřených otázek, kde by správně měl být např. pouze celočíselný vstup, zadává i texty abecedy. K výzkumníkovi se dostanou výsledky z dotazníků ve formě XLS, které potom ručně přepisuje do dalších tabulek, kde má napsané vzorce pro výpočet číselných výsledků z dotazníku a zároveň opravuje chybně vyplněné kolonky. Tento manuální přístup je zdlouhavý a dá se automatizovat, a to validací vstupu, které umožní i automatické výpočty.

2.4 Funkční požadavky

V následujících sekcích můžeme vidět utvářející se stav TO-BE. Prvním nástrojem pro definování TO-BE stavu jsou funkční požadavky.

- **FR01 – Vyplnění dotazníku**
  Systém umožňuje respondentovi vyplnit dotazník v takové formě, aby se mohl co nejlépe soustředit na pravost svých odpovědí. K tomu slouží vlastnosti vytvořeného uživatelského rozhraní. Respondent na obrazovce vidí vždy pouze jednu otázku, na kterou může odpovídat. Po vyplnění odpovědi na otázku se uživatelova odpověď zvaliduje. Systém respondentovi neumožní se přesunout na další otázku až do doby, dokud nezadá odpověď na otázku ve správném formátu. K odeslání dotazníku nastává ve chvíli, kdy respondent zodpoví poslední otázku.
- **FR02 – Validace odpovědí**
  Když respondent vyplňuje dotazník, jeho odpovědi jsou validovány, a to ve chvíli, kdy chce respondent potvrdit svou odpověď. Validace probíhají podle typu otázek. Například na některé otázky je potřeba odpovědět ve formátu hh:mm, jiné otázky mohou požadovat pouze celočíselnou odpověď. Konkrétní vstupy mají konkrétní validaci.
- **FR03 – Kombinování dotazníků**
  Systém umožní uživateli vytvořit sadu otázek, která se skládá z více existujících dotazníků. Bude k tomu využito parametrů URL, kde se do parametru budou skládat kódy dotazníků, ze kterých chce uživatel vygenerovat sadu otázek.
- **FR04 – Výpočet dotazníků**
  Systém vypočítává a ukládá výsledky dotazníků na základě předdefinovaných vzorců daného dotazníku.
- **FR05 – Ukládání původních odpovědí**
  Systém mimo uložení výpočtu dotazníku ukládá také odpovědi tak, jak byly respondentem původně zadány.
- **FR06 – Identifikace respondenta**
  Při vyplňování dotazníku může respondent zadat, zda je zároveň účastník výzkumu. Jestliže je zároveň účastník výzkumu, identifikuje se svým výzkumným číslem. Jestliže respondent není účastníkem výzkumu, identifikuje se pouze alternativním identifikátorem. To může být např. jméno. Když respondent zadává své výzkumné číslo, pak systém validuje jeho správný formát. Správný formát výzkumného čísla je kombinace šesti alfanumerických znaků, které jsou v polovině odděleny podtržítkem.

2.5 Nefunkční požadavky

Nefunkční požadavky se týkají vlastností softwarového systému, které nesouvisejí se specifickými funkcionalitami. Popisují spíše, jak by se měl systém chovat z hlediska výkonu, spolehlivosti nebo použitelnosti.

Byly sesbírány následující nefunkční požadavky.

- **NFR01 – Bezpečnost přenosu dat**

  Veškerá komunikace mezi uživatelskými rozhraními a backendem probíhá přes šifrovaný protokol HTTPS.

- **NFR02 – Bezpečnost uložených dat**

  Data jsou uložena v databázi, která je zaheslovaná přísně tajným heslem. Databáze běží na serveru, ke kterému mají přístup pouze vývojáři systému.

- **NFR03 – Ochrana proti spamovému vyplňování dotazníku**

  Abychom předešli útoku spambotů na uživatelském rozhraní pro vyplňování dotazníků, tak je toto rozhraní chráněno před nejběžnějšími spamovými útoky.

- **NFR04 –Uživatelská přívětivost uživatelského rozhraní**

  U uživatelských rozhraní je dbáno na to, aby bylo pro uživatele přívětivé a jednoduché k použití.

- **NFR05 – Ochrana před dvojím odesláním dotazníku**

  Systém neumožní uživateli odeslat stejný dotazník dvakrát.

2.6 Případy užití

V této sekci definujeme uživatele, kteří budou systém používat a definujeme případy užití.

2.6.1 Definice uživatelů

Pro definování systémových požadavků je velmi užitečné si definovat, jací uživatelé budou systém používat. Definujeme 3 typy uživatelů. Jsou to účastník výzkumu, respondent a administrátor.

* **Účastník výzkumu**

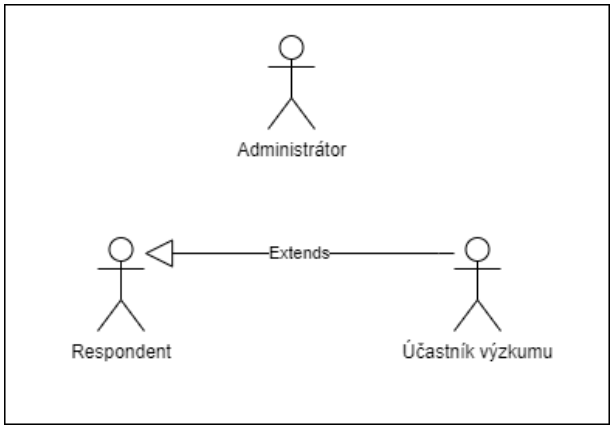
Uživatel, který se zaregistroval do Výzkumu Odolnosti a je možné ho identifikovat výzkumným číslem. Účastník výzkumu navíc může nosit nositelné zařízení od Garminu, pomocí kterých systém sbírá data o jeho spánku. Účastník výzkumu může být zároveň respondent.

* **Respondent**

Uživatel, který vyplňuje dotazník.

* **Administrátor**

Uživatel, který má přístup k datům sesbíraným od účastníků výzkumu a k vyplněným dotazníkům od respondentů.



Obrázek 2.1. Diagram aktérů

Pro uživatele nyní představím jejich případy užití. Každý aktér může být zároveň respondent, pro kterého jsou případy užití v této části systému primární.

2.6.2 Případy užití

Systém bude obsahovat následující případy užití:

UC01 - Zodpovědět na otázku

1. Respondentovi se zobrazí v prohlížeči otázka
2. Respondent vyplní či vybere svou odpověď
3. Uživatelské rozhraní zkontroluje validitu respondentovy odpovědi
4. Uživatelské rozhraní zobrazí novou otázku

UC02 - Identifikovat se v dotazníku výzkumným číslem

1. Respondentovi se zobrazí identifikační otázka
2. Jestliže je respondent účastník výzkumu, vyplní výzkumné číslo
3. Uživatelské rozhraní zkontroluje validitu respondentovy odpovědi
4. Uživatelské rozhraní zobrazí novou otázku

UC02 - Identifikovat se v dotazníku alternativním identifikátorem

1. Respondentovi se zobrazí identifikační otázka
2. Respondent zadá, že nemá výzkumné číslo a vyplní alternativní identifikátor
3. Uživatelské rozhraní zkontroluje validitu respondentovy odpovědi
4. Uživatelské rozhraní zobrazí novou otázku

UC03 - Odeslat vyplněný dotazník

1. Respondent v prohlížeči vyplní odpovědi na dotazník
2. Uživatelské rozhraní pro vyplňování dotazníků odešle vyplněné otázky
3. Systém provede výpočet nad daty od respondenta
4. Systém uloží vyplněný dotazník s vyhodnocením do databáze
5. Systém potvrdí úspěšný příjem vyplněného dotazníku

- **UC04 - Vytvořit odkaz s požadovanou sadou otázek** (Tento use case plní zejména administrátor, který je obeznámen se způsobem vytváření sady otázek z dotazníků.)

  1. Administrátor v parametrech URL vyplní kódy
  2. Administrátor zašle respondentům URL s parametry dotazníků
  3. Respondent otevře URL v prohlížeči
  4. Systém respondentovi vygeneruje sadu otázek na základě parametrů v URL

Kapitola 3
Implementace

V této kapitole se budeme zabývat volbou technologií pro jednotlivé části aplikace, obecnou architekturou aplikace, uživatelským rozhraním a přijímáním dotazníků na backendu.

3.1 Volba technologií

K volbě technologií nám mohou pomoct některé otázky. Je pro daný úkol technologie vhodná? Umí vývojáři s technologií pracovat? Tato sekce se bude těmito otázkami zabývat rozděleně pro jednotlivé základní celky celého řešení. Navíc se zamyslíme nad možnými alternativami.

3.1.1 Backend

Vytváříme webovou aplikaci a pro tvorbu webové aplikace potřebujeme webový server. Nabízí se nám několik možností, jakou technologii zvolit. Populární volbou je Node.js, kde vzniká mimo výhoda toho, že vývojář může pracovat v jazyku JavaScript jak na backendové straně, tak na frontendové straně. Dále bychom mohli zvolit jazyk Python s nástroji buďto Flask nebo Django. My jsme zvolili jazyk Java ve verzi 17. Nejjednodušší způsob, jak vytvořit v Javě webovou aplikaci, je využít framework Spring Boot, který již v sobě má webový server Tomcat a my tak můžeme rovnou začít vypisovat endpointy. Aplikace psané v Javě jsou výhodné v tom, že je můžeme provozovat na jakémkoliv fyzickém případně virtuálním stroji, na kterém běží Java Virtual Machine (JVM). Máme -li odpovědět na otázku, zda je pro daný úkol technologie vhodná, pak odpověď je ano. Můžeme snadno a rychle zprovoznit webovou aplikaci a rychle se začít zabývat business logikou, jakou potřebujeme. Další otázka zní, jestli s danou technologií umí vývojáři pracovat. I zde vznikla shoda. Všichni zúčastnění vývojáři mají s technologií Spring Boot a Java zkušenosti.

Nejbližší alternativou pro zvolenou Javu se Spring Bootem by bylo využít Spring Boot s Kotlinem, který také běží na JVM a má oproti Javě několik výhod. Výhoda je např., že stačí psát méně kódu díky jeho vylepšené syntax. Častý argument proti použití Javy mezi vývojáři je, že má příliš boilerplatu (opakující se části kódu, které jsou dlouhé a nedají se zkrátit). Toto Kotlin řeší právě vylepšenou syntaxí, které se říká syntax sugar. Velký problém programovacích jazyků, který Kotlin řeší, je přítomnost datových typů, které mohou mít hodnotu null. Tyto datové typy mohou vyvolávat tzv. null pointer exceptions způsobené tím, že chceme pracovat nebo dělat operace s proměnnými, které nemají referenci na žádný kus paměti. Kotlin tento problém řeší datovými typy, které hodnoty null nabývat nemohou. Ovšem stále zachovává i datové typy, které hodnoty null nabývat mohou. Existují případy, kdy je to potřeba.

3.1.2 Frontend

Frontend, nebo - li uživatelská část aplikace je část systému, se kterou pracuje uživatel. My jsme pro uživatelskou část zvolili prohlížeč. Tento fakt nám velice úzce omezuje

volbu programovacího jazyka. Moderní prohlížeče podporují JavaScript nebo WebAssembly (a některé podporují ještě také Javu nebo zastaralý Flash.) Jediná racionální volba pro pro daný úkol je JavaScript. JavaScript totiž splňuje dvě kritéria, které jsme si definovali v úvodu sekce. Technologie je vhodná, JavaScript je jeden z nejpoužívanějších programovacích jazyků dnes a je to průmyslový standard pro vývoj webových aplikací. Splňuje i kritérium, že vývojáři na projektu s technologií umí pracovat. Vývojáři mají s jazykem zkušenosti v mnoha podobách (protože JavaScript má spoustu nadstaveb, knihoven.) WebAssembly je naopak technologie, která v dnešní době není průmyslovým standardem. Využívají ji spíše experimentátoři. Co na technologii ovšem zaujme je rychlost operací, která před JavaScriptem vede. Nakonec zmíním, že WebAssembly vývojáři nepíší kód přímo ve WebAssembly, ale kompilují jej z jiných výše úrovňových jazyků jako je C++ nebo Rust.

Zvolili jsme tedy JavaScript. Nechceme ale psát kód ve vanilla JavaScriptu, existují nástroje, které nám vývoj zjednoduší. Přichází tedy na řadu výběr knihovny či frameworku. Pro realizaci uživatelských rozhraní byla zvolena knihovna React. Je to velmi populární a spolehlivá volba pro tvorbu projektu malého i středního rozsahu. Vývojáři s knihovnou mají zkušenost. V administrátorském rozhraní se jedná primárně o single page application (SPA), kde veškeré stránky generuje uživatelův prohlížeč. Naopak u uživatelského rozhraní pro vyplňování dotazníků je využito funkce nadstavby Next.js, která dokáže předgenerovávat Reactí kód na serveru. Využívá tedy principu server side rendering (SSR). SSR má oproti SPA řadu výhod i nevýhod. Výhody jsou např. to, že ušetříme výpočetní zátěž prohlížeče klienta a vznikne možnost si odpovědi od serveru cachovat. Dále je výhoda také to, že SSR umožní vyhledávačům (search engines) vykonávat nad naší stránkou indexaci a bude ji pak snadnější nalézt v samotném vyhledávání v prohlížeči. Naopak výhodou SPA je lepší klientský zážitek, kdy klient může svižně procházet mezi okny stránky bez přenačítání celé stránky. Nejideálnější je tyto dva přístupy v projektu kombinovat, což nám Next.js umožňuje.

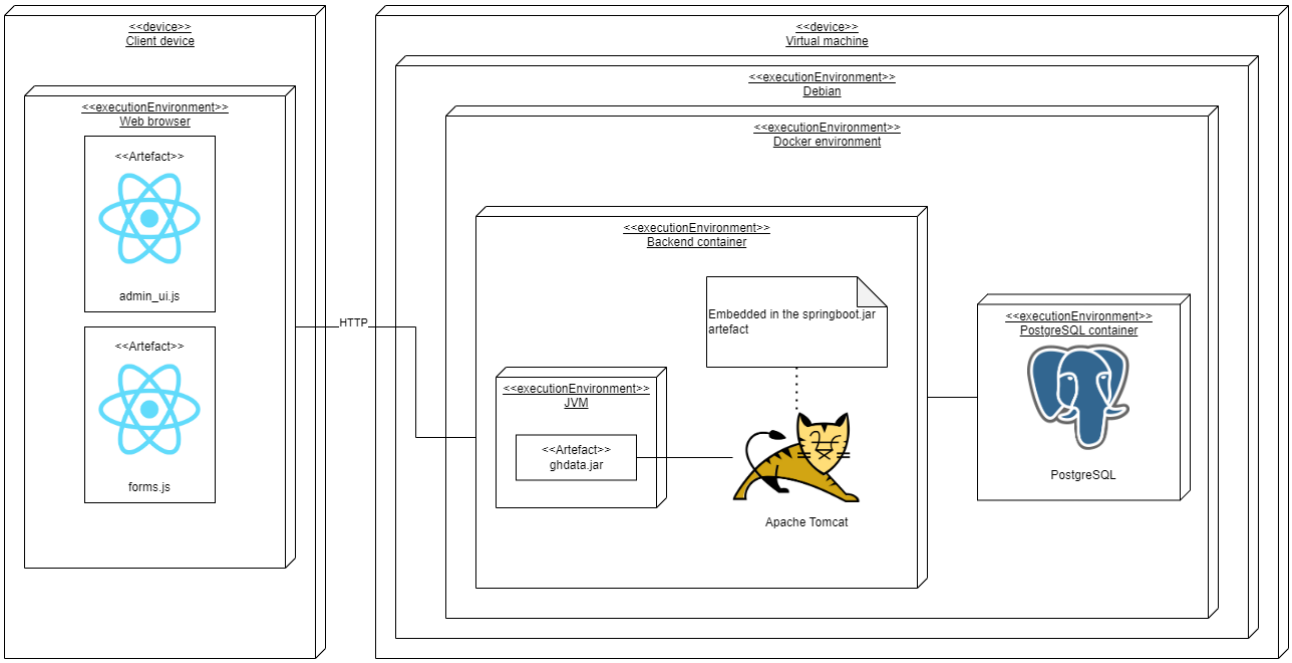
3.1.3 Databáze

Pro uchováváni dat byla zvolena open sourceová databáze PostgreSQL. Je zde splněno kritérium, že vývojáři mají s technologií zkušenost. Dále platí, že technologie je pro daný úkol vhodná, protože potřebujeme strukturovaná data rozdělená do jednotlivých tabulek, což nám relační databáze nabízí. Alternativa by byla např. MySQL nebo Oracle Database, nicméně PostgreSQL je zdarma k užívání. Další alternativou by bylo využít nerelační NoSQL databázi jako je např. MongoDB nebo Redis. Výběr nerelační databáze nepřipadá v úvahu už z důvodu, že vývojáři na projektu s těmito technologiemi nemají zkušenost.

3.2 Architektura aplikace

Na architekturu aplikace se můžeme dívat z velkého měřítka i z malého měřítka. V této sekci se podíváme na architekturu z velkého měřítka. Jedná se o architekturu klient - server. Klienty máme v našem případě dva, administrátorské rozhraní a rozhraní pro vyplňování dotazníků. Jedná se specifičtěji o monolit. To znamená, že celý backend se skládá z jednoho uzlu, který běží na jedné JVM. Pokud by to nebyl monolit, mohla by být využita např. mikroservisní architektura, kde existuje více uzlů systému, které dohromady tvoří backend. Monolit je nejjednodušší a nejzákladnější způsob jak tvořit jednoduché webové aplikace a dokáže odbavit většinu požadovaných funkcí i zátěž. Museli bychom mít pádný důvod k tomu, abychom monolit nepoužili, jako je např. očekávaná velmi velká zátěž systému.

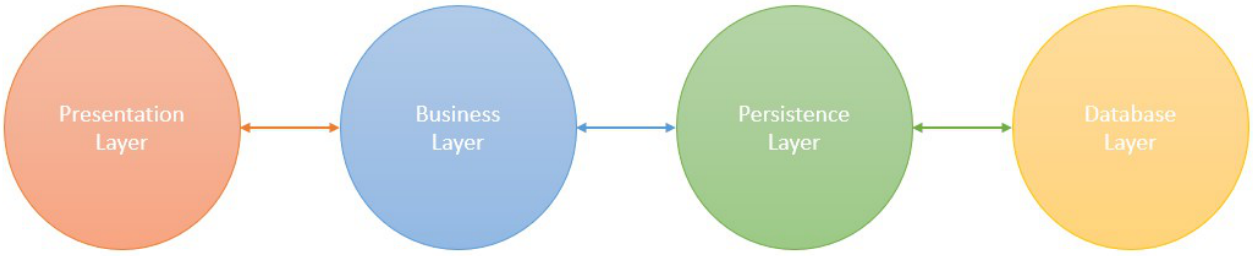
Máme tedy dva klienty, klienti komunikují s backendem. Backend komunikuje přes perzistentní vrstvu s databází. To můžeme vidět na diagramu nasazení.



Obrázek 3.1. Diagram nasazení

3.2.1 Backend

Na backendu je využita populární vícevrstevnatá architektura. V této architektuře máme několik vrstev, které jsou naskládány na sebe. Vrstvy by měly správně komunikovat pouze se sousední vrstvou. Do vrstev patří vrstva prezentační, tam máme Controllery, kde definujeme endpointy. Klient na endpointy posílá požadavky. Controllery dále předají požadavek do business vrstvy. V business vrstvě nacházíme business logiku aplikace, v tomto konkrétním případě se v ní nachází zejména výpočty nad dotazníky, export dat do XLS nebo rozcestník pro CRUD operace. Business vrstva dále komunikuje s persistentní vrstvou, která odbavuje CRUD operace. Je to vrstva, která má na starost persistenci dat a komunikaci s databází. V Spring Boot aplikacích se persistentní vrstva skládá z repository vrstvy a modelové vrstvy. V repository vrstvě jsou metody pro přístup k datům. V modelové vrstvě jsou definované entity tak, jak jsou mapované do databáze za pomocí ORM.



Obrázek 3.2. Vícevrstevnatá architektura Spring Bootu [11]

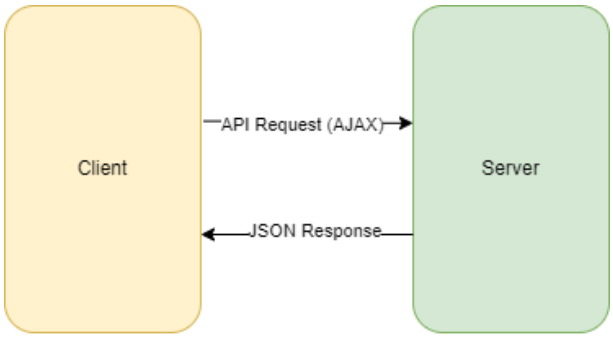
3.2.2 ORM, JPA a jeho implementace

Object Relationship Mapping (ORM) nástroje mapují entity objektově orientovaných jazyků, jako je právě Java nebo C#, do řádků databázových tabulek. V Javě je specifikováno rozhraní pro ORM v Java Persistence API (JPA). JPA je pouze rozhraní a potřebujeme k němu implementaci. Populárními implementacemi jsou Hibernate, EclipseLink nebo Apache OpenJpa. V projektu byl využit Hibernate. Implementace JPA využívají pro komunikaci s databází Java Database Connectivity (JDBC). JPA vývojářovi zjednodušuje práci s databázovou komunikací. Ovšem samotné využití JDBC by mělo za výhodu větší rychlost operací [12].

3.2.3 Frontend

U architektury frontendu by se dalo zaměřit např. na architekturu nástroje React, který je využit u obou uživatelských rozhraní. Bylo by zajímavé sledovat, jak se kód napsaný vývojářem propojuje s Document Object Modelem (DOM). Já se nicméně raději zaměřím na koncept architektury Single Page Application (SPA), jejíž princip je využit u obou rozhraní, ačkoliv jenom administrátorské rozhraní tuto architekturu plně opisuje. U rozhraní pro dotazníky dochází k předgenerování obsahu na serveru, ovšem jeho větší celky princip SPA také dodržují.

V SPA mezi aplikací v browseru a serverem probíhá plynulá komunikace - ze serveru přichází data a ta mění různé části HTML dokumentu (DOMu) [13]. Dochází k generování obsahu na stránce na základě toho, jaká přijde ze serveru odpověď na asynchronní dotaz. K provádění asynchronních dotazů na server používám knihovnu axios, která v sobě využívá XMLHttpRequest API [1]. Server typicky posílá odpověď ve formátu JSON, ale může se jednat o jakýkoliv formát dat. Můžou to být např. XML či binární data.



Obrázek 3.3. Single Page Application architektura

3.3 Rozhraní pro odesílaní dotazníků

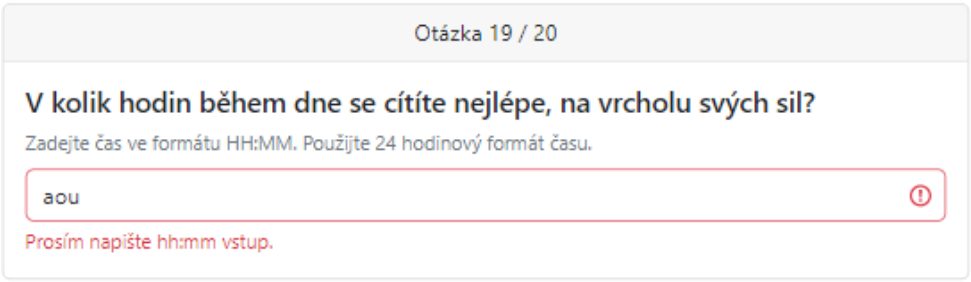
Toto rozhraní používají respondenti, aby vyplňovali dotazníky. Vzhledem k tomu, že toto rozhraní využívá několik uživatelů (respondentů) je potřeba zajistit, aby bylo rozhraní uživatelsky přívětivé. V této sekci se podíváme na vlastnosti tohoto rozhraní.

3.3.1 Validace vstupu

Každá otázka v dotazníku má validovaný vstup, která se dále odvíjí od typu vstupu. Respondent musí na otázku odpovědět a musí na ní odpovědět tak, aby jeho odpověď odpovídala požadovanému formátu. Pokud se respondent pokusí odeslat prázdný nebo

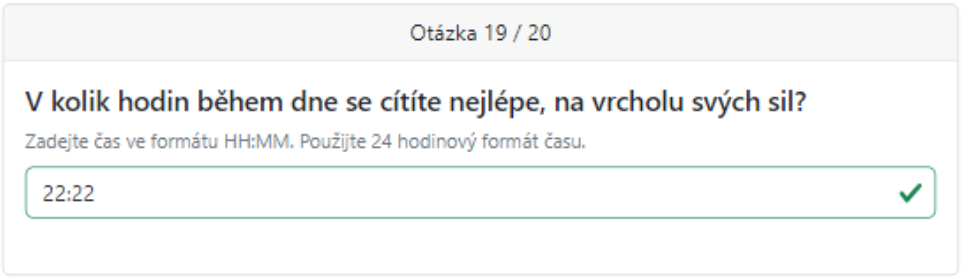
[1] https://axios-http.com/docs/intro

nevalidní vstup, dostane negativní zpětnou vazbu v podobě chybové hlášky a zčervenání vstupního pole.



Obrázek 3.4. Příklad negativní zpětné vazby

Pokud respondent zadá a odešle vstup, který odpovídá požadovanému formátu, dostane pozitivní zpětnou vazbu ve formě krátkého zezelenání vstupového pole a následným přesunutím na následující otázku.



Obrázek 3.5. Příklad pozitivní zpětné vazby

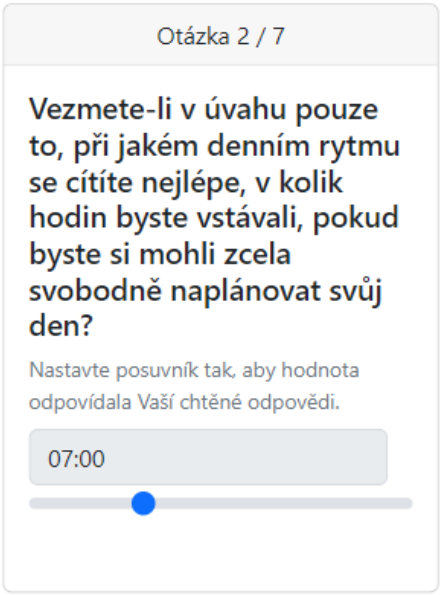
Vzhled pro pozitivní i negativní zpětnou vazbu byl realizován pomocí Boostrap 5 CSS tříd “is-invalid” a “is-valid” [2].

3.3.2 Typy vstupu

Vzhledem k tomu, že otázky v dotaznících jsou různého charakteru, vstupní pole požadují různé formáty odpovědí. Definuji tedy následující typy vstupů, které každý mají svůj typ validace.

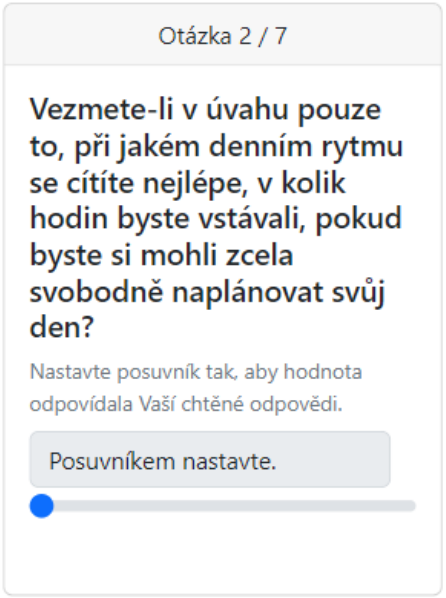
- **Vstup s posuvníkem** Vstup posuvníkem používám, když potřebuji od uživatele vstup ve formátu hh:mm, který je ohraničen nějakou dolní a horní mezí. Dolní resp. horní mez může být např. 12:00 resp. 18:00, nebo i 20:00 resp. 03:00. Implementace posuvníku tedy umí přecházet přes hranici půlnoci. K implementaci byla využita hodnota sekund, se kterou samotný posuvník pracuje. Ze zadané hodnoty je vypočítán vždy zbytek po dělení hodnotou 86400 (24 hodin v sekundách), který je pak převeden na hodnotu ve formátu hh:mm. Vstupem do React componenty HourRangeInput jsou hodnota `min` znázorňující minimální hodnotu posuvníku v hodinách a `totalHours`, podle které se vypočítá horní mez přičtením k hodnotě min.

[2] https://getbootstrap.com/docs/5.3/forms/validation/



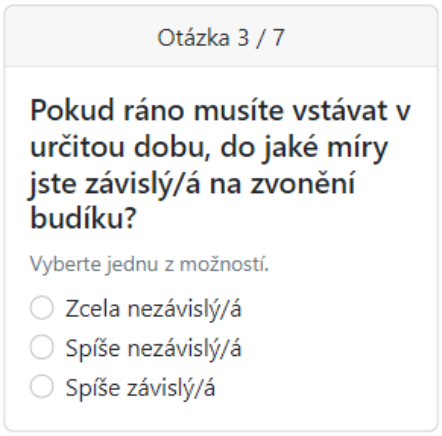
Obrázek 3.6. Ukázka vstupu s posuvníkem

Počáteční stav posuvníku nemá žádnou hodnotu. Uživatel musí s posuvníkem nejprve posunout. V případě, že se uživatel pokusí odpovědět na otázku bez posunutí posuvníku, dostane negativní zpětnou vazbu „Prosím, nastavte posuvník".



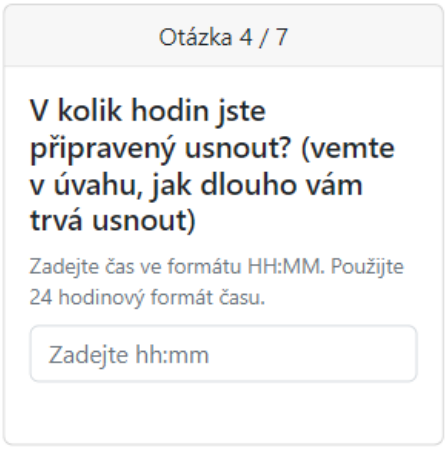
Obrázek 3.7. Počáteční stav vstupu s posuvníkem

- **Vstup s výběrem odpovědi** Vstup s výběrem odpovědi definuje množinu možných odpovědí, kterou si uživatel může vybrat. Tento druh otázky je realizován pomocí radio buttonů. Je tedy možné vybrat pouze jednu odpověď. React componenta pro tento typ vstupu přijímá na vstupu pole objektů, kde každý objekt definuje možnou odpověď a hodnotu odpovědi. Počáteční stav otázky tohoto typu je takový, kde není zaškrtlá žádná možnost. Aby mohl uživatel pokračovat, musí jednu vybrat.



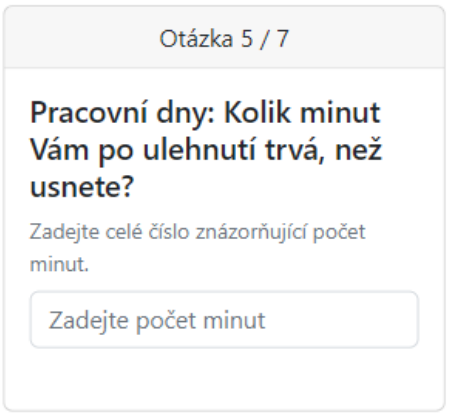
Obrázek 3.8. Ukázka vstupu s výběrem odpovědí

- **HhMm vstup** HhMm vstup je textové pole, které je validováno na vstup ve 24 hodinovém a minutovém formátu času. Validními vstupy jsou např. “3:32”, “03:32”, “23:23”. Vstupy nevalidními jsou např. “Aaa” nebo “34:20”. Negativní zpětná vazba je pro uživatele v tomto případě hláška “Prosím napište hh:mm vstup”.



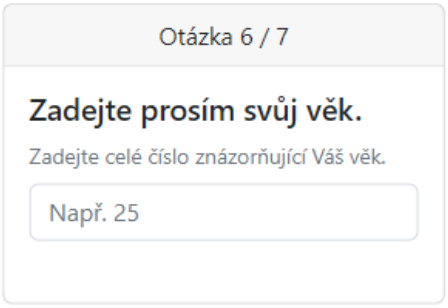
Obrázek 3.9. Ukázka HhMm vstupu

- **Minutový vstup** Minutový vstup používám, když potřebuji od uživatele zadat počet minut. Typicky to bývají otázky typu `kolik minut vám trvá usnout`. Validním vstupem je celé kladné číslo.



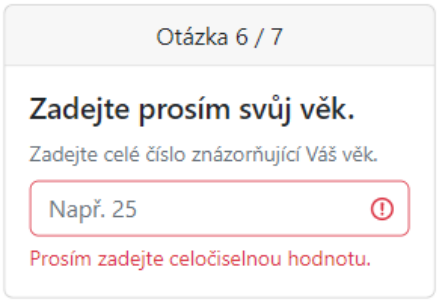
Obrázek 3.10. Ukázka minutového vstupu

- **Celočíselný vstup** Celočíselný vstup je obdobný minutovému vstupu. Rozdíl je však v tom, že u minutového vstupu jsou předdefinovány návodné popisy i chybové hlášky. U celočíselného vstupu je popisy i chybové hlášky potřeba definovat zvlášť. Proto má React componenta tohoto vstupu na vlastním vstupu atributy `description` (návodný popis) a `placeholder` (co za text se ukazuje v prázdném poli).



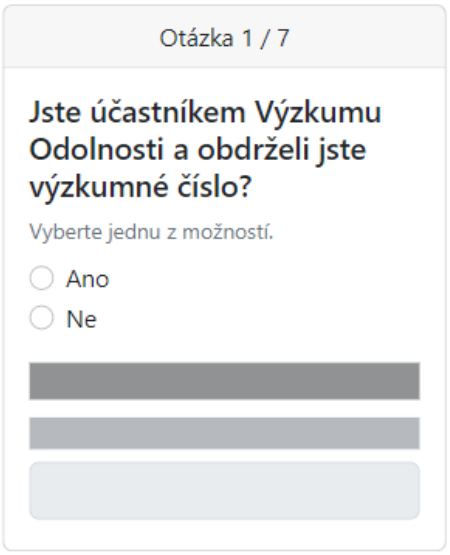
Obrázek 3.11. Ukázka celočíselného vstupu

Chybová hláška je stejná jak pro minutový vstup, tak pro celočíselný vstup a to „Prosím zadejte celočíselnou hodnotu".



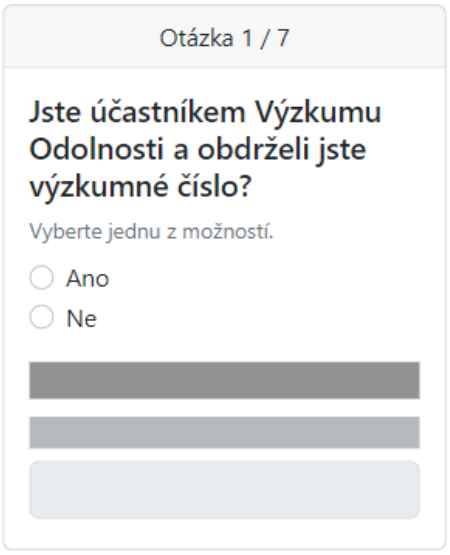
Obrázek 3.12. Negativní zpětná vazba celočíselného vstupu

- **Dvě možnosti s textovým doplněním** Tento vstup vznikl při řešení problému, jak rozlišit respondenty, kteří jsou účastníci Výzkumu Odolnosti a tudíž mají výzkumné číslo a respondenty, kteří výzkumné číslo nemají. První část otázky spočívá v tom zjistit, do které kategorie respondent patří.

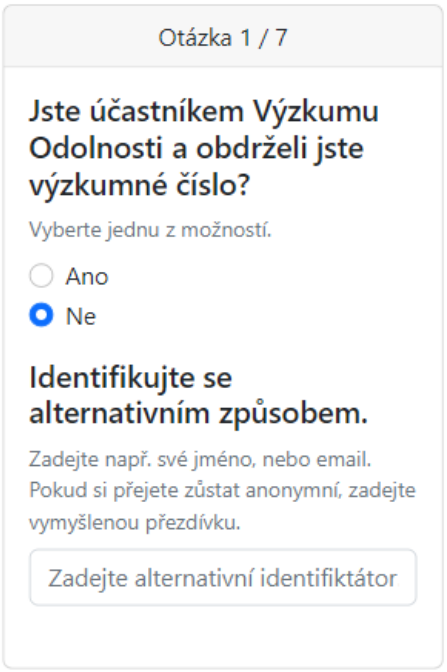


Obrázek 3.13. Ukázka vstupu dvou možností s textovým doplněním

Při rozkliknutí jedné z odpovědí se ukáže textový vstup, kde uživatel doplní další informaci. V případě, že zadá, že účastníkem výzkumu je, bude požádán o vyplnění výzkumného čísla.

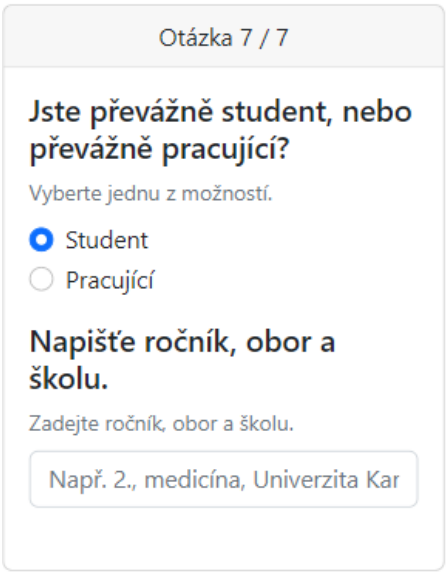


Obrázek 3.14. Výběr ano u identifikační otázky



Obrázek 3.15. Výběr ne u identifikační otázky

Uživatel vyplní textové pole, které je validováno na základě regexu, který je vstupem React componenty tohoto vstupu. Dalším příkladem použití tohoto vstupu je otázka v demografickém dotazníku.



Obrázek 3.16. Dvě možnosti s textem u demografického dotazníku

Je možné si všimnout, že se obě otázky liší textem. Liší se také validací. React komponenta tohoto typu vstupu má nejvíce vlastních vstupních atributů ze všech dosavadních komponent pro vstupy. Je potřeba definovat text, návody a možné vstupy u první otázky a text, návody, placeholder a zejména validaci u druhé otázky. Validace druhé otázky se také může lišit na základě výběru z první otázky. U příkladu s identifikační otázkou o výzkumném čísle je druhý vstup validován pro správný formát výzkumného čísla v případě, že uživatel zadal, že výzkumné číslo má. Jestliže

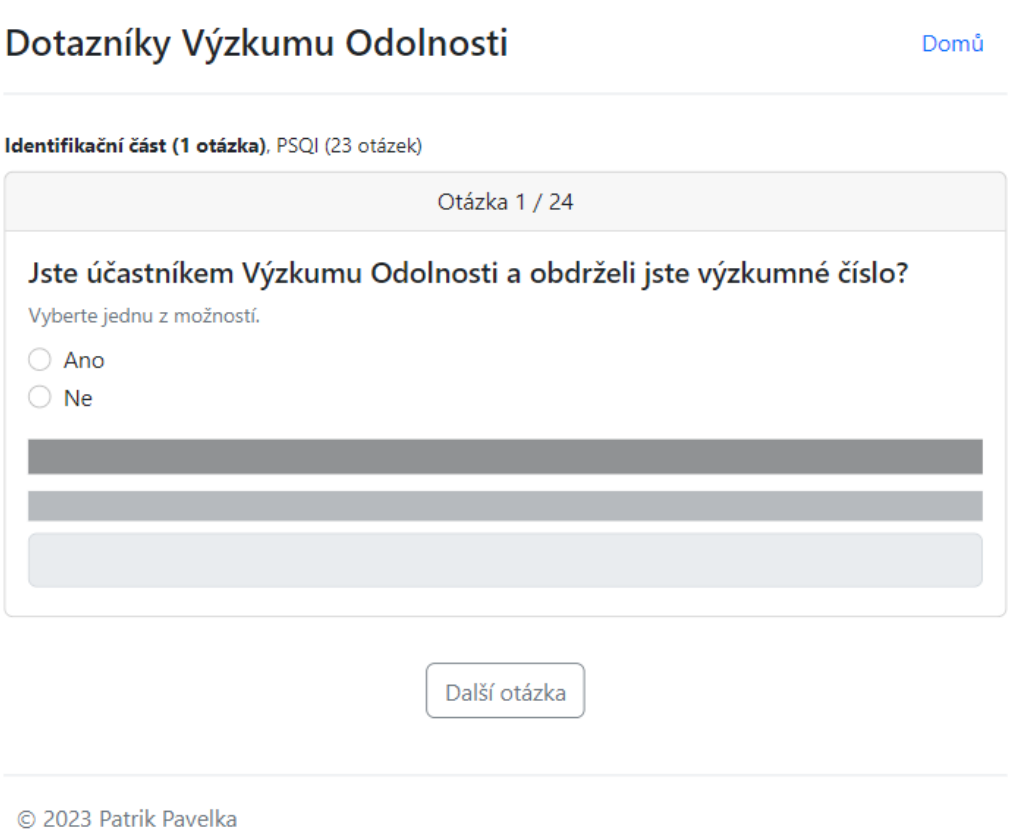
uživatel zadal, že nemá výzkumné číslo, je požádán o alternativní identifikátor. Validace vstupu alternativního identifikátoru spočívá pouze v tom, že je zadaný vstup neprázdný.

3.3.3 Generování dotazníku

Na hlavní stránce je rozcestník pro jednotlivé dotazníky. Při stisknutí tlačítka „Přejít k dotazníku" se uživatel přesměruje na odkaz, který vygeneruje otázky z daného dotazníku.

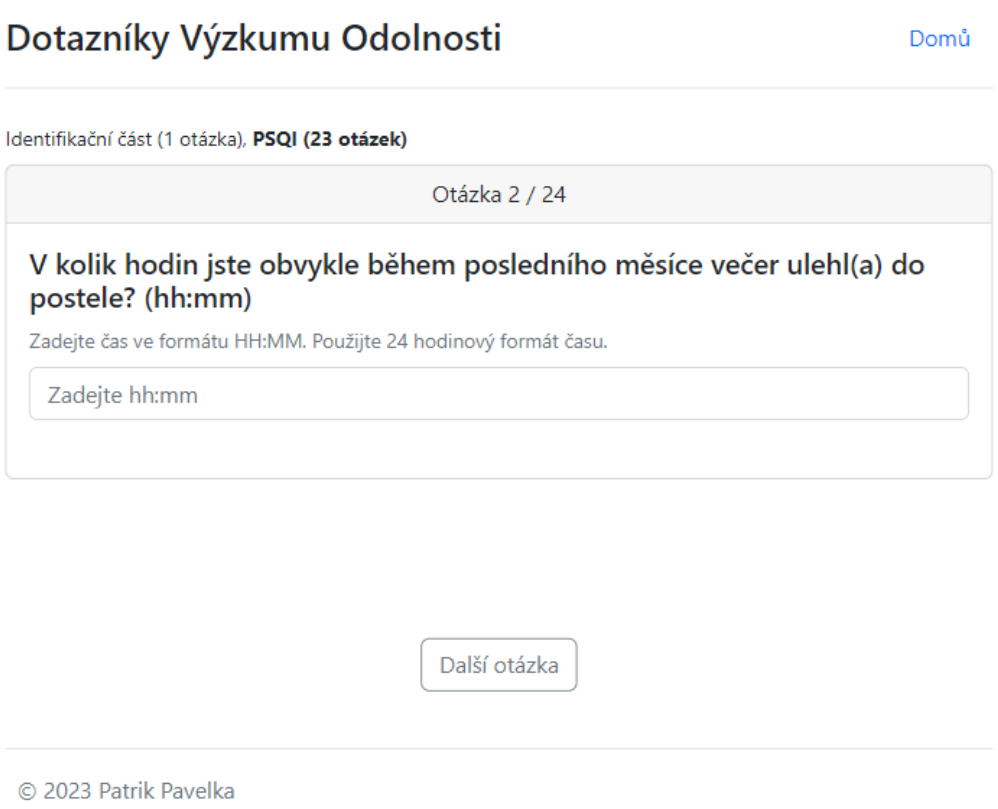
Obrázek 3.17. Hlavní obrazovka

Např. tlačítko u PSQI uživatele přesměruje na /questionnaire?q=psqi.



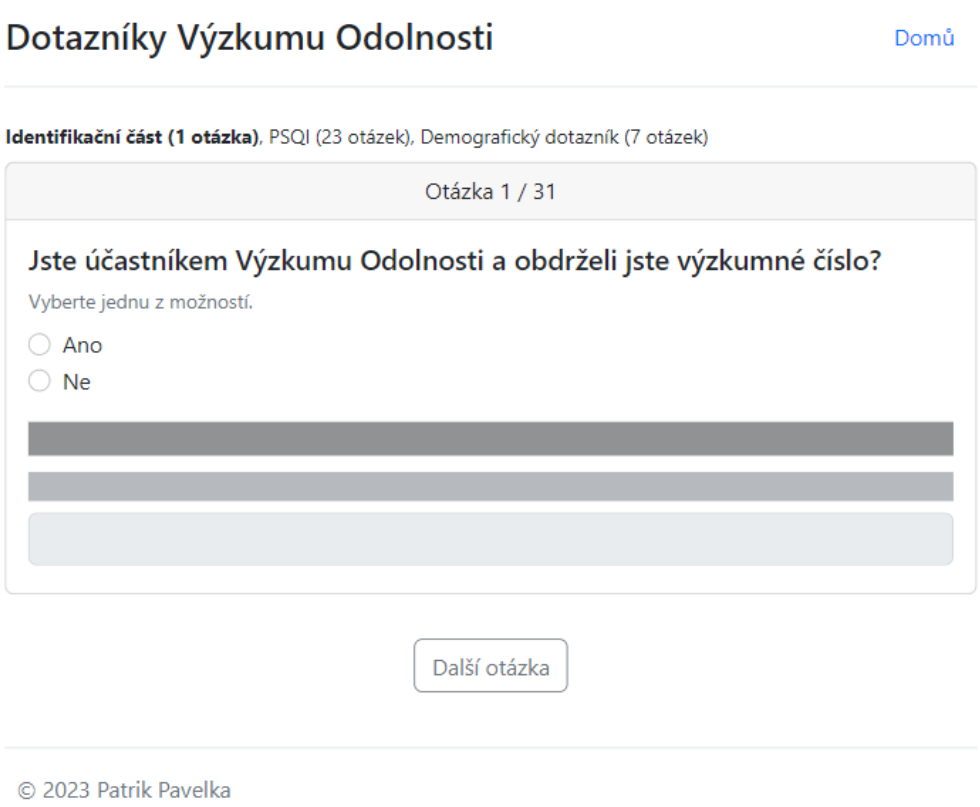
Obrázek 3.18. Vygenerovaný PSQI dotazník v identifikační části

Je možné si všimnout, že celý dotazník se nyní skládá z identifikační části (1 otázka) a samotného PSQI dotazníku (23 otázek). Při zodpovězení otázky identifikační části se uživatel přesune na otázku PSQI dotazníku. Tato skutečnost je znázorněna zvýrazněním dané sady dotazníku, ve které se uživatel nachází.



Obrázek 3.19. Vygenerovaný PSQI dotazník v PSQI části

Dotazníky jde i kombinovat. Pokud chceme, aby uživatel vyplnil PSQI dotazník a zároveň i demografický dotazník, můžeme toho dosáhnout vytvořením GET dotazu s odpovídajícím URL. Je potřeba kódy dotazníku definovat v parametru q. Možné kódy pro dotazníky jsou psqi, mctq, meq, pss, dzs, demo. Tedy kombinace PSQI a demografického dotazníku by vznikla GET dotazem s parametrem q: questionnare?q=psqi&q=demo.



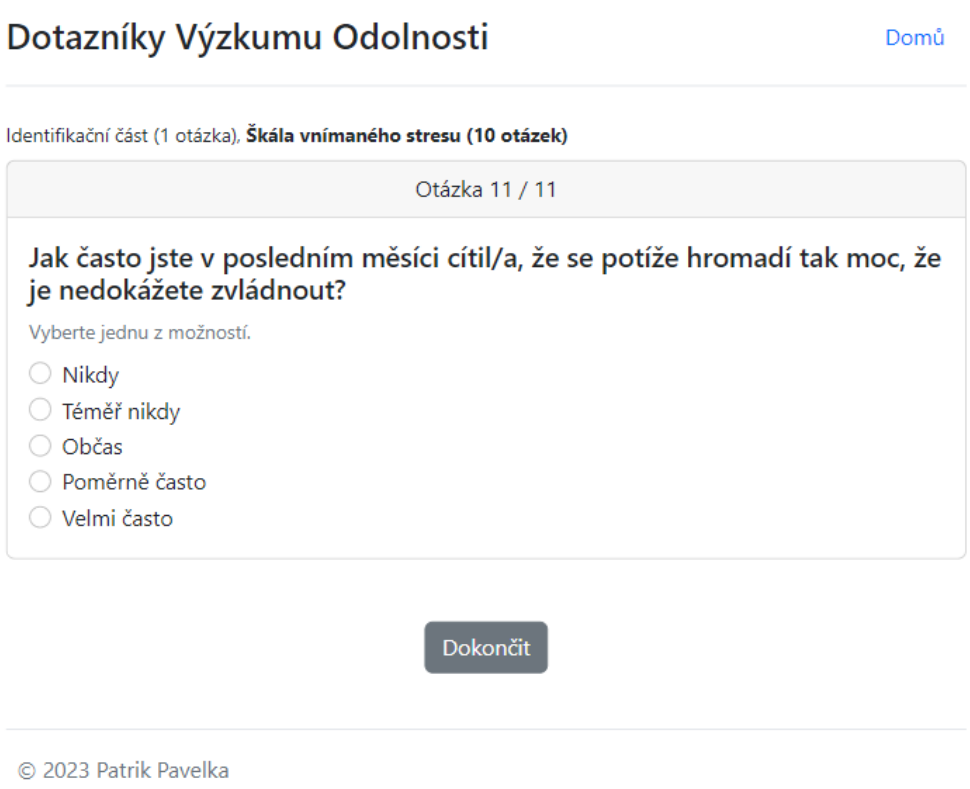
Obrázek 3.20. Vygenerovaná kombinace dotazníků

Dotazníky jde kombinovat libovolně. V případě, že se do parametru q dostane jediný kód dotazníku, pro který neexistuje žádný dotazník, skončí generování dotazníku chybovou hláškou, že dotazník se zadaným kódem neexistuje. Zadaný kód je vypsán do dokumentu. Stojí za zmínku, že zde vzniká hrozba XSS útoku, protože vstup uživatele (vstup v parametru URL) je vypsán do dokumentu. XSS ošetřuje React. Používá k tomu klasický HTML encoding.

Obrázek 3.21. Ukázka chybové hlášky při generování dotazníků

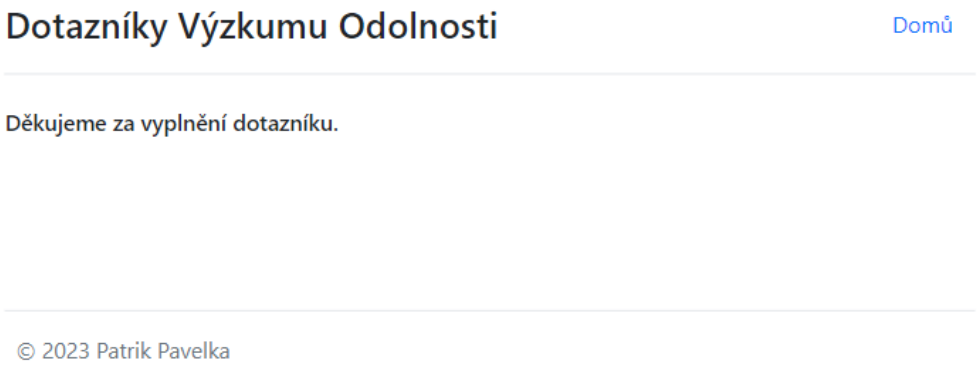
3.3.4 Odeslání dotazníku

Pokud se dostaneme na poslední otázku, změní se tlačítko „Další otázka” na otázku „Dokončit”. Změní se i barva tlačítka, aby bylo znázorněno, že je současná otázka poslední.



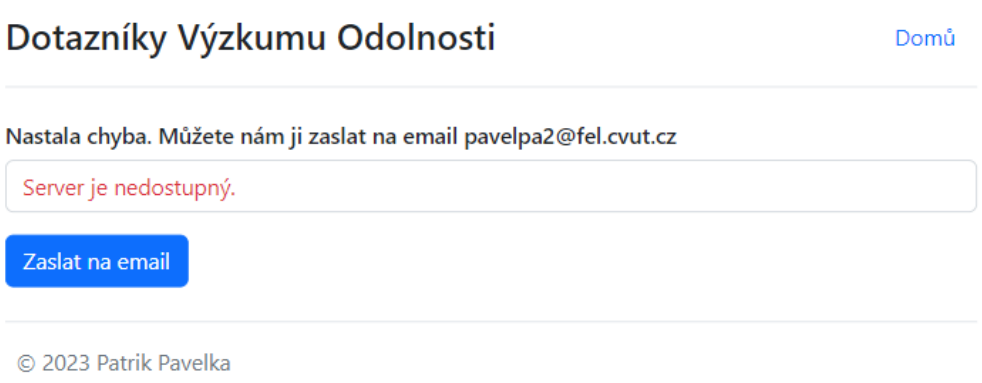
Obrázek 3.22. Změněné tlačítko Dokončit

Při kliknutí na tlačítko „Dokončit" se data z vyplněného dotazníku zašle na backend. Pokud proběhne vše v pořádku, backend odpoví 200 a uživatel je přesměrován na stránku s poděkováním.



Obrázek 3.23. Děkovací stránka

Pokud nastane nějaká chyba, např. pokud je server nedostupný, je uživateli zobrazena chybová hláška.



Obrázek 3.24. Stránka s chybovou hláškou po vyplnění dotazníku

3.3.5 Struktura dat dotazníků pro generování do UI

Strukturu dotazníků přijímá jako vstup React komponenta QuestionnareComponent. Přijímá také i celkový počet otázek jako "totalNumberOfQuestions", který již předtím vznikl při skládání samotného questionnares vstupu. Questionnares vstup komponenty je JavaScript objekt, který má jako klíče kódy dotazníků. Klíč má pak odpovídající hodnotu další JavaScript objekt, kde jsou definovány jednotlivé otázky dotazníku. Struktura tedy může vypadat např. takto:

```
{
  Id: {...},
  Psqi: {...},
  Meq: {...}
}
```

Část "id" je přítomna v každém vygenerovaném dotazníku, protože je vždy potřeba se identifikovat. Přítomnost ostatních dotazníků je dána tím, zda se nachází v q parametru GET dotazu. V samotných objektech pod klíči dotazníků nalezneme jednotlivé otázky. Např. meq vypadá následovně:

```
{
  Id: {...},
  Psqi: {...},
  Meq: {
    Q1: {
      questionType: "hourRangeInput",
      minHour: 5,
      numberOfHours: 7,
      label: "V kolik hodin se cítíte nejlépe?",
      inputId: "meqQ1InputId",
      answer: "",
      question: {
            id: 401
       }
    },
    Q2: {...},
  }
}
```

Q1 i Q2 jsou definicemi otázek. V definici je zahrnut typ vstupu, znění otázky nebo také k jaké patří otázce. K jaké patří otázce je definováno přítomností parametru id, které značí id otázky. Otázka je v pod stejným id uložena v databázi. Parametr answer uchovává odpověď od uživatele. Parametr label je samotné znění otázky, jak se objeví v dokumentu. Otázka může mít i daleko více parametrů, např. otázky se vstupem typu "twoChoiceWithText" musí obsahovat informaci o znění otázek, návodných textech nebo regex výrazů pro validaci.

3.3.6 Vytváření payloadu

Struktura POST payloadu vychází ze struktury dotazníků pro generování do UI. Vzniká tak, že se iteruje skrz každý dotazník a v něm skrz každou otázku. Z každého dotazníku vzniknou "computationVariables", neboli proměnné potřebné k výpočtu a "answers", nebo - li čitelné odpovědi k otázkám. Proměnné potřebné k výpočtu jsou JavaScript objekt, který má jako klíče kódy otázek, hodnota pod klíčem už není další objekt, ale právě hodnota potřebná k výpočtu, tedy buďto string, nebo číslo. Čitelné odpovědi k otázkám slouží k tomu, aby si výzkumník, který bude pozorovat výsledky vyplněného dotazníku, mohl přečíst, co respondent přesně vyplnil za hodnotu do pole. Payload tedy může vypadat následovně:

```
{
    "identifying": {
        "hasResearchNumber": "true",
        "researchNumber": "A23_AA2",
        "alternativeIdentifier": undefined
    }
    "meq": {
      "answers": [
         {
           "question": {
                  "id": 401
            },
                "answer": "05:00"
       },
       "computationVariables": {
          "q1": "05:00",
          "q2": "20:00",
           …
       }
     }
}
```

V tomto případě je odpověď na první otázku zapsaná jak v "answers", tak v "computationVariables". To není pravidlem pro všechny otázky, odpovědi se mohou lišit např. u otázek, kde je předdefinovaná množina možných odpovědí. Další příklad, se odpovědi liší, můžeme nalézt u otázek se dvěma možnostmi s textovým doplňkem, konkrétně u identifikační otázky. V "answer" bychom nalezli "true|A22_A22|" a v "computationVariables" bychom nalezli pouze hodnotu true. Pro výpočet je podstatné, že respondent má výzkumné číslo. Pro výzkumníka, který čte odpovědi, je zajímavá i informace jaké bylo vyplněno výzkumné číslo. V payloadu je klíč "identifying" přítomen vždy. Je potřeba identifikovat, kdo dotazník vyplnil.

3.4 Přijímáni dotazníků na backendu

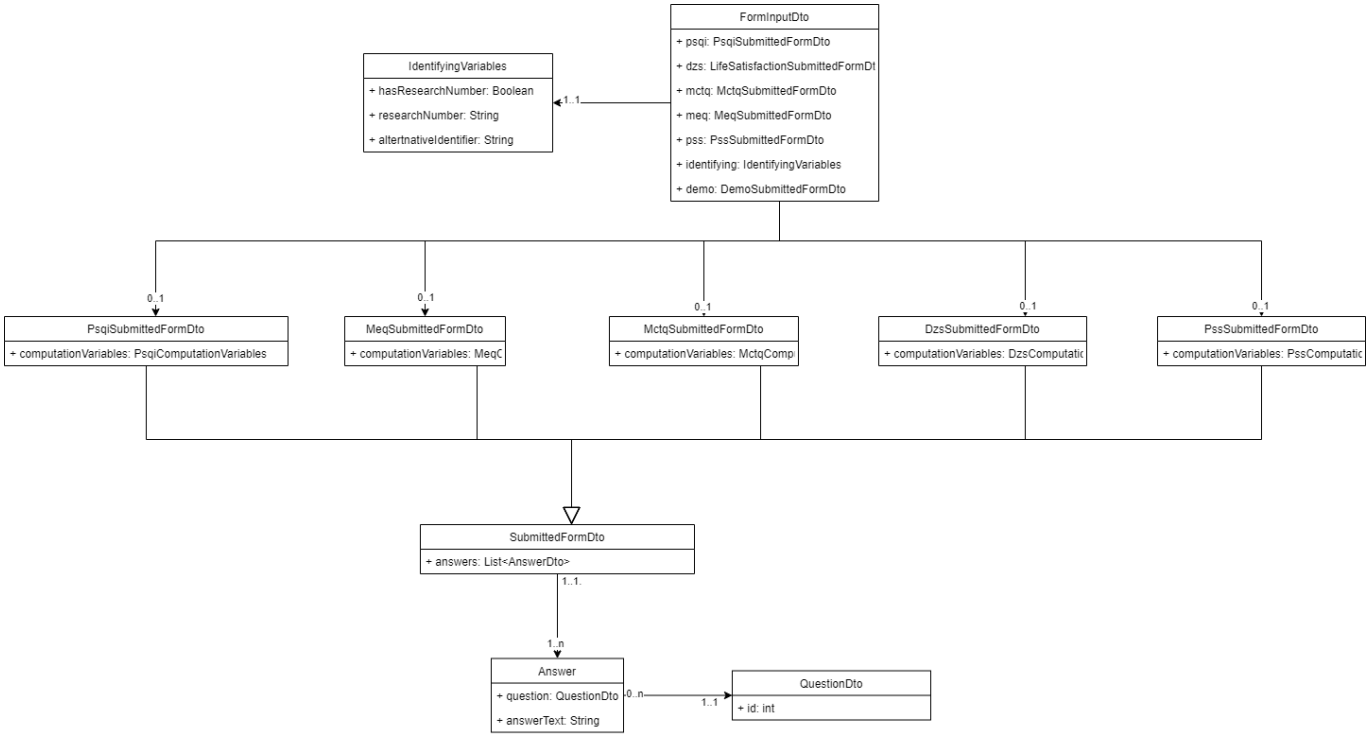
Pro přijímání dotazníků na backendu existuje jeden POST endpoint, který přijímá všechny typy dotazníků.

```
HTTP POST /form-submit
```

Payload pro tento endpoint vychází z struktury v předchozí části a lze jej vyjádřit Java objektem. Payload se mapuje na FormInputDto objekt.

3.4.1 Datová struktura DTO dotazníků

FormInputDto objekt mi umožňuje mít pouze jeden endpoint na zasílání vyplněných dotazníků. Má v sobě reference na všechny vedené dotazníky. Proto je do payloadu na frontendu možné vložit jen ty, které potřebujeme, tedy ty, které uživatel vyplnil. Ostatní zůstanou v JSON payloadu jako undefined a na backend se tedy namapují jako hodnota null. V diagramu můžeme vidět multiplicitu 0..1, hodnota je tedy nullable.



Obrázek 3.25. Diagram DTO hierarchie dotazníků

Payload musí vždy obsahovat identifying atribut, který je znázorněn strukturou IdentifyingVariables. Na diagramu to jde poznat podle multiplicity 1..1 a v Java kódu je nad atributem anotace @NotNull z knihovny javax.validation.constraints. Anotace zajišťuje, že pokud endpoint dostane FormInputDto s parametrem identifyingVariables null, backend na dotaz odpoví 400 BAD REQUEST.

3.4.2 Datová struktura dotazníků jako JPA entit

Dto z payloadu je potřeba přemapovat na JPA entity, abychom je mohli uložit do databáze. Entitní datový model je velmi podobný DTO modelu. Existuje zde jedna entita `SubmittedForm`. Tato entita představuje dotazník, jak jej respondent vyplnil. Z entity `SubmittedForm` se odvíjí několik dalších entit vztahem extends. Vztah extends definuje vztah mezi rodičem a potomkem, kde potomek dědí všechny vlastnosti rodiče a může zároveň definovat nové vlastní vlastnosti. Atributy `SubmittedForm` jsou:

- **created: LocalDate**

  Tento atribut znázorňuje, kdy byl záznam o vyplněném dotazníku vytvořen. Pro jeho realizaci byla využita anotace z balíčku javax.persistence @PrePersist nad setterem atributu. Atribut se nastaví na aktuální datum v době uložení do databáze.

- **researchParticipant: ResearchParticipant**

  Tímto parametrem se rozlišují respondenti. ResearchParticipant entita v sobě má dále atribut researchNumber, podle kterého se respondent identifikuje.

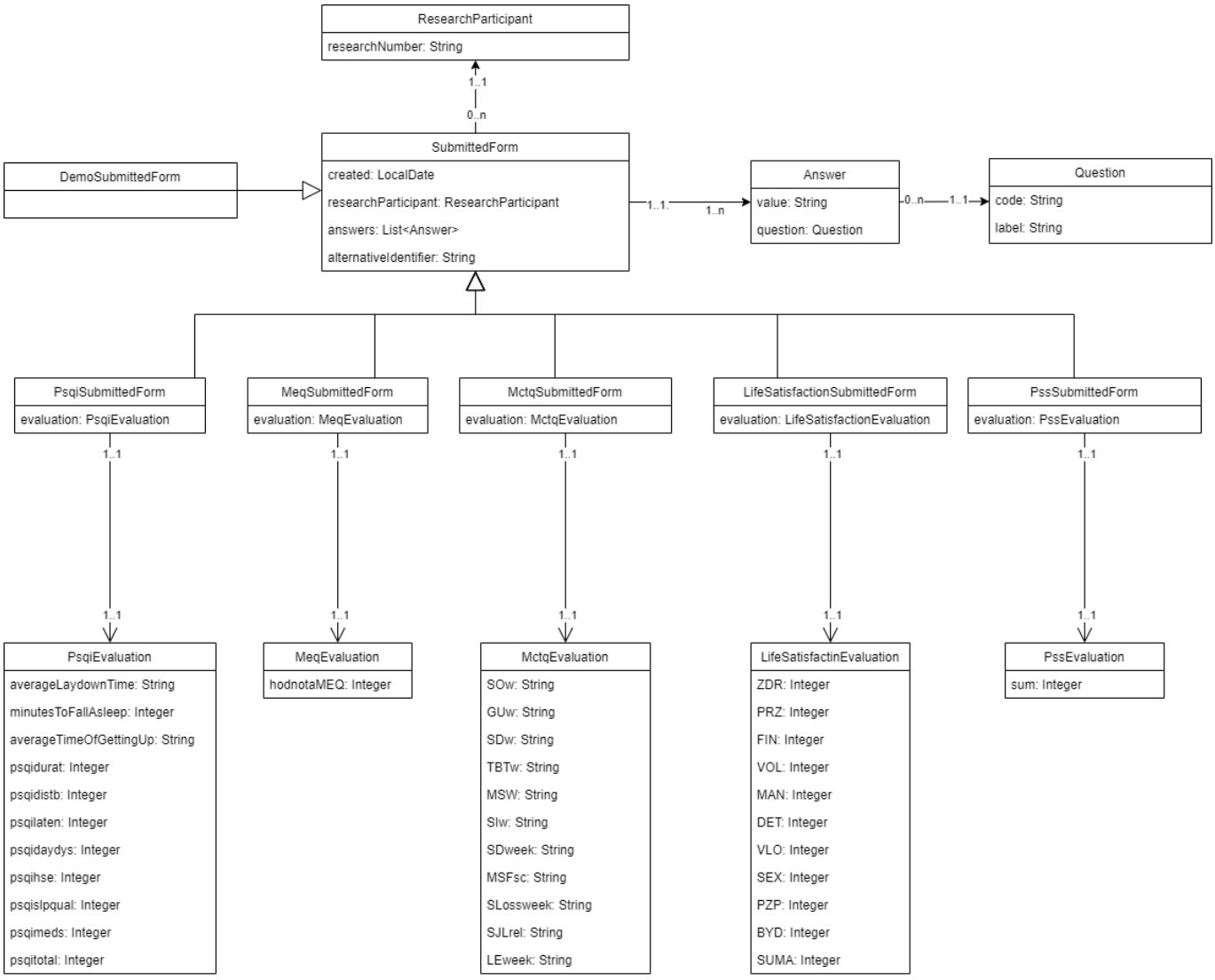
- **alternativeIdentifier: String**

  Toto je alternativní identifikátor respondenta, který při vyplňování dotazníku nevyplnil svoje výzkumné číslo, ale zadal alternativní identifikátor.

- **answers: List<Answer>**

  Je seznam odpovědí na otázky.

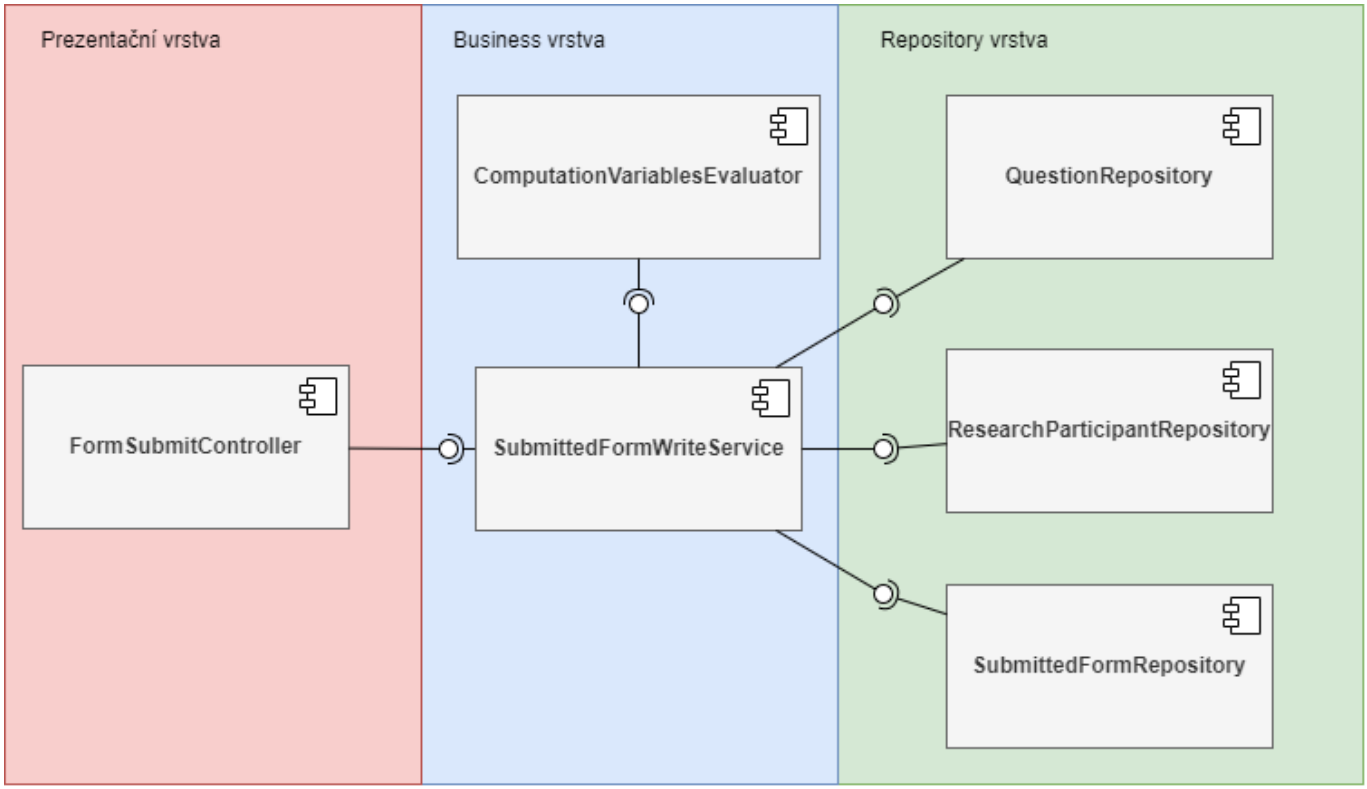
Pro každý dotazník existuje potomek entity `SubmittedForm`, protože každý dotazník má navíc svoje unikátní vyhodnocení.



Obrázek 3.26. Diagram tříd dotazníků

3.4.3 Výpočty a mapování na JPA entity

V diagramu tříd JPA entit je vidět, že každý dotazník má své vyhodnocení. Toto vyhodnocení vzniká na základě computationVariables, které jsou přítomny v DTO objektech. Musí být úplné, aby mohly vzniknout výsledky vyhodnocení dotazníků. V aplikaci existuje část, která se o tyto výpočty stará. Jmenuje se ComputationVariablesEvaluator a mapuje ComputationVariablesDto na Evaluation entity. Další krok, aby byla SubmittedForm entita úplná, je namapování AnswerDto na Answer entitu. Tuto funkcionalitu obstarává jedna metoda u všech druhů dotazníků, protože není závislá na konkrétním typu zaslaného dotazníku. V diagramu je vidět, že AnswerDto obsahuje answerText, což je to, co uživatel napsal jako odpověď. Dále obsahuje QuestionDto, které obsahuje id. Na základě tohoto id se vyhledá v databázi příslušná otázka a přiřadí se k nově vzniklé Answer entitě, která bude přiřazena k samotné SubmittedForm entitě. Mezi "SubmittedForm" a "Answer" entitou je vztah @OneToMany definovaný pomocí anotace z knihovny javax.persistence. Anotace přijíma i parametry, např. parametr cascade, který je zde nastaven na CascadeType.ALL. To nám říká, že když je do databáze uložen dotazník, budou zároveň uloženy i odpovědi.



Obrázek 3.27. Diagram komponent přijímání dotazníků

3.4.4 Definice otázek

AnswerDto obsahuje QuestionDto, to v sobě má pouze id otázky. Předdefinovávám sadu otázek ke každému dotazníku, aby se při odesílání a ukládání vyplněného dotazníku duplicitně neukládal text otázky. Předdefinování jsem docílil SQL scriptů, které do databáze vkládají znění otázek.

```
insert into question(id, code, label)
values (200, 'dzshealth0', 'Se svým tělesným zdravotním stavem jsem...'),
       (201, 'dzshealth1', 'Se svou duševní kondicí jsem...'),...
```

Na frontendu mám připravenou tu samou sadu otázek, ale do payloadu /form-submit už text otázky neposílám. Tímto předdefinováním otázek šetřím tedy nejen místo v databázi, ale i velikost jednotlivého payloadu. Kdybych tento způsob chtěl ještě vylepšit, otázky bych neuchovával otázky na frontendu. Postupoval bych tak, že při generování dokumentu v prohlížeči se frontend dotáže backendu, jaké jsou pro chtěný dotazník otázky a poté je do DOMu vygeneruje. Tento způsob má výhodu v tom, že bych pokaždé uživateli do prohlížeče neposílal celou sadu otázek (která sice není vidět, ale v současné době tam je). Nevýhoda tohoto přístupu je taková, že by musel nejprve proběhnout asynchronní dotaz na backend, kde jsou otázky uchovávány. Další nevýhodou je, že na backendu v současné době nejsou definovány typy vstupů nebo regexy pro validace. Současný způsob je tedy kompromis mezi designovým řešení v kódu a jednoduchostí implementace.

Kapitola 4
Testováni

Testování je část cyklu softwarového vývoje, kde zajišťujeme kvalitu vytvořeného softwaru. V této kapitole píšu o testování backendu a frontendu.

4.1 Backend

Hlavními částmi aplikace backendu jsou controller, servisní vrstva, a repository vrstva. Obvykle má cenu testovat pouze servisní vrstvu, kde se nachází veškerá business logika. Controller vrstva i repository vrstva je naimplementovaná velmi jednoduše. U controllerů využívám aspektů z org.springframework.web.bind.annotation, které řeší poslouchání na endpointech a mapování payloadu na objekty za mě. U repository vrstvy používám implementace org.springframework.data.jpa.repository.JpaRepository, která už úplnou většinu možných repository operací implementuje. Obecně v aplikaci využívající vícevrstevnatou architekturu stačí v kontextu jednotkových testů testovat business logiku v servisní vrstvě. V servisní vrstvě se nachází komponenta definovaná interfacem `ComputationsVariablesEvaluator`, která vypočítává vyhodnocení dotazníku, a komponenta `TimeComponent`, která převádí různé formáty času na formáty jiné. V programovém testování se tedy zaměřím na tyto dvě komponenty.

4.1.1 Jednotkové testy

Zvolil jsem psát jednotkové testy, to znamená, že testuji vždy jednu část funkcionality. V kontextu Javy se dá říct, že jednotkové testy jsou testy v rámci jedné Java třídy. Testuji tedy `ComputationVariablesEvaluator`. Zde jsem postupoval tak, že jsem použil výsledky z dotazníků z dosavadního fungování projektu, zkopíroval jsem vstupní data a mé unit testy testují evaluaci dotazníku na základě vstupu. V testech se vždy pracuje s hodnotami expected a actual, expected jsem vyčetl ze již existujících vypočtených dat a actual jsou ty, které počítá má aplikace. `TimeComponent` jsem testoval hlavně proto, abych se ujistil, že správně metodu implementuji. Předem jsem věděl, co chci z funkce na základě vstupu získat, tak jsem nejprve napsal test a poté začal implementovat. Tento postup je charakterizující pro test driven development. U TimeComponenty existuji testy na výpočet délky mezi času usnutí a probuzení, převod hhmm formátu na sekundy nebo převod epoch času na LocalDate (datový typ v Javě).

Pokrytí tříd [%]	*Pokrytí metod [%]*	*Pokrytí řádek kódu [%]*
16	26	23

Tabulka 4.1. Přehled pokrytí kódu testy

4.2 Frontend

V rozhraní pro vyplňování dotazníků jsou důležitými funkcionalitami validace vstupů a generování sad otázek dotazníků. Dále je důležité, že se na základě stavu programu

renderují do dokumentu (okna prohlížeče) příslušné komponenty nebo že se uživatelské rozhraní mění na základě stavu aplikace. K otestování používám testy.

4.2.1 End to end testy

K end to end testování používám nástroj Cypress [1]. Cypress je nástroj, který mi umožní automaticky proklikat uživatelské rozhraní a kontrolovat, že se rozhraní chová, jak má. Testuji každý typ vstupů, tedy minutový vstup, vstup s posuvníkem atp. Před každým testem je potřeba definovat popis.

```
describe('Testing identifying question', () => {
    ...
}
```

Prvním parametrem funkce describe je popis a druhým je funkce, ve kterém je definovaných testů, další vnořené describe bloky nebo beforeEach bloky, které definují, co se má stát před každým testem.

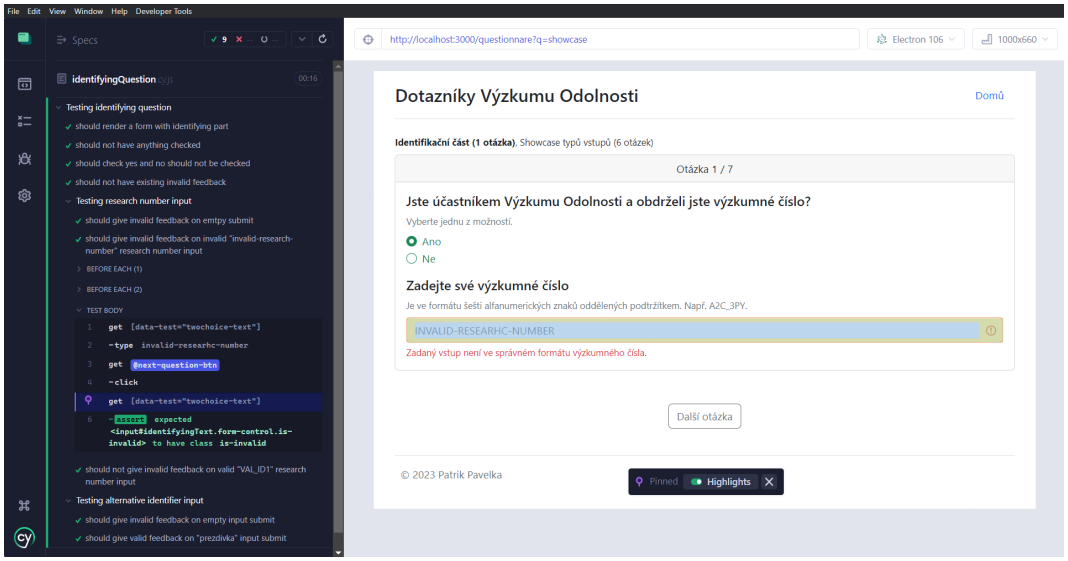
```
beforeEach(() => {
 cy.visit(Cypress.env('base_url') + '/questionnare?q=showcase')
 cy.get('[data-test="next-question-button"]', {timeout: 10000})
   .should('be.visible');
 cy.get('[data-test="next-question-button"]').as('next-question-btn')
})
```

V tomto beforeEach uvnitř describu definuji, že cypress má navštívit stránku na `base_url` (to může být localhost). Dále definuji, že má počkat, dokud se na obrazovce neobjeví element s atributem `data-test='next-question-button'`. To je zde z toho důvodu, že se UI generuje na klientské straně, proto trvá chvíli po načtení okna, než se vygenerují i samotné bloky stránky. Poslední třetí řádek definuje alias pro tlačítko 'Další odpověď.

Dále definuji testy, u testování identifikační otázky to je např., že když zadám nevalidní výzkumně číslo, tak text vstupové pole zčervená, což se projevuje tím, že se na vstupní pole přidá daná css classa.

```
it('should give invalid feedback on invalid "invalid-research-number",
 () => {
  cy.get('#identifyingHasResearchNumberYes').check()
  cy.get('[data-test="twochoice-text"]')
    .type("invalid-researhc-number")
  cy.get('@next-question-btn').click()
  cy.get('[data-test="twochoice-text"]')
    .should('have.class', 'is-invalid')
})
```

[1] cypress.io



Obrázek 4.1. Test nevalidního výzkumného čísla v Cypressu

Testuji oba možné případy průchodu u identifikační otázky, tedy možnost, že uživatel zadá že má uživatelské jméno a možnost, že zadá, že ho nemá.

Podobným způsobem testuji i další typy vstupů. Existují vždy testy, kde zadám validní vstup a nevalidní vstup a kontroluji, že validní vstup dostane pozitivní zpětnou vazbu a naopak, že nevalidní vstup dostane negativní zpětnou vazbu.

4.3 Uživatelské testy

Hlavní částí systému, se kterou interagují respondenti, je rozhraní pro vyplňování dotazníků. Pro potřebu end to end testů i uživatelských testů byl vytvořen showcase dotazník, kde je zastoupen každý typ vstupu. Můžeme u něj tedy testovat, zda jsou instrukce a zpětné vazby od systému srozumitelné pro respondenta.

Poprosil jsem 4 testerů, aby mi prošli aplikací. Sledoval jsem, kolik otázek vyplnili na první pokus ve správném formátu a naopak. Ptal jsem se jich na hodnocení na vzhled, kvalitu instrukcí a zpětnou vazbu aplikace. Na tyto otázky testeři odpovídali od 1 do 5, kde 1 je výborné a 5 je velmi špatné.

Existuje pouze jeden scénář uživatelského testu a to vyplnit showcase dotazník.

4.3.1 Tester 1

Informace o testerovi a hodnocení:

- **Věk** 32
- **Uživatelská úroveň** Střední
- **Počet chybně vyplněných otázek** 7
- **Počet správně vyplněných otázek na první pokus** 0
- **Vyplnění dotazníku bylo snadné** 2
- **Ohodnoťte kvalitu instrukcí k vyplnění dotazníku** 1-
- **Ohodnoťte celkový vzhled aplikace** 2
- **Aplikace mi dává dobrou zpětnou vazbu** 1

U identifikační otázky váhá, protože nebyl dobře obeznámen s kontextem projektu. Vyplňuje, že nemá uživatelské číslo. Ve chvíli kdy se objeví doplňující text k doplnění je testerovi otázka jasnější.

U otázky, kdy chodí spát se dlouho zamýšlí nad otázkou, neví co má doplnit, protože má nepravidelný spánek. Otázku s možností vybírá a vyplňuje. Velmi rychle vyplňuje otázku na věk a ani u poslední otázky neváhá.

Tester měl dobrou zkušenost s UI. Testerovi přišly otázky položeny příliš široce a měl problém na ně odpovídat. Tester ovšem nemá kontext projektu.

Tester nejprve nerozumí informaci o sadě, ve které části dotazníků se nachází. Nevšiml si ani změny zvýraznění při přechodu mezi sadami.

4.3.2 Tester 2

Informace o testerovi a hodnocení:

- **Věk** 19
- **Uživatelská úroveň** Expert
- **Počet chybně vyplněných otázek** 7
- **Počet správně vyplněných otázek na první pokus** 0
- **Vyplnění dotazníku bylo snadné** 1
- **Ohodnoťte kvalitu instrukcí k vyplnění dotazníku** 2
- **Ohodnoťte celkový vzhled aplikace** 1-
- **Aplikace mi dává dobrou zpětnou vazbu** 1

Tester se v aplikaci rychle orientuje. Na identifikační otázku odpovídá, zadává svůj email. Otázka s posuvníkem se mu jeho slovy líbí. Na všechny otázky odpovídá rychle až na otázku o tom, kdy usíná. Tam se zamýšlí.

Tester nemá problém s využitím aplikace ke splnění scénáře. Komentuje, že je příjemné, že se otázky týkají spánku.

U vzhledu by vylepšil placeholder čáry u identifikační otázky, jinak se testerovi líbí vzhled aplikace.

4.3.3 Tester 3

Informace o testerovi a hodnocení:

- **Věk** 22
- **Uživatelská úroveň** Expert
- **Počet chybně vyplněných otázek** 4
- **Počet správně vyplněných otázek na první pokus** 3
- **Vyplnění dotazníku bylo snadné** 1-
- **Ohodnoťte kvalitu instrukcí k vyplnění dotazníku** 1-
- **Ohodnoťte celkový vzhled aplikace** 1
- **Aplikace mi dává dobrou zpětnou vazbu** 1-

Tester snadno prochází dotazníkem. Je nespokojený se šedými placeholdery u identifikační otázky. Naschvál vyplňuje do políček odpovědi v chybném tvaru. Podle testera by mohly být chybové hlášky přesnější. Např. u hh:mm vstupu zadal 25:25, chybová hláška měla podle testera být, že je vstup mimo interval. Aktuální chybová hlaška je momentálně pouze, že má uživatel zadat hh:mm vstup.

4.3.4 Závěr uživatelského testování

Z testů vychází, že průchod aplikací je pro uživatele spíše snadný. Testeři neměli problém odpovídat na otázky správným formátem na první pokus. UI se testerům spíše líbilo. Při průchodu zpomalovaly testery spíše otázky, nad kterými se museli zamyslet, než složitost aplikace.

Na základě zpětné bych upravil šedé placeholdery u identifikační otázky, které negativně zmiňovali dva testeři. Dále bych lépe propracoval hlášky negativní zpětné vazby na základě toho, co respondent opravdu zadal. Také bych znovu prošel formulace otázek.

Kapitola 5
Nasazení

V této kapitole popíšu, jak jsem celý systém nasadil. V současné době je systém nasazen na virtuálním serveru poskytnutého od ČVUT.

5.1 Backend a databáze

K nasazení backendu a databáze jsem využil virtuální server, který jsem obdržel od školy. Vygeneroval jsem si také certifikát pro HTTPS na doménu vyzkumodolnosti.felk.cvut.cz, komunikace se serverem je tedy šifrovaná. K tomu, abych na serveru spustil backend i databázi využívám docker [1]. Docker mi umožňuje spustit běh programu v kontejnerizovaném prostředí. To v praktickém směru znamená, že na samotném virtuálním serveru nemusí být nainstalované Java, PostgreSQL, ani např. Gradle nebo podobně. Jsou nainstalované pouze na virtuálním stroji zprostředkovaným samotným Dockerem. Toto řešeni mi zjednodušuje deploy, protože je snadné definovat např. verze technologií a zejména nasazení nových verzí.

5.1.1 Postup nasazení

Nejprve je potřeba vytvořit image backendu, který je vytvořen pomocí Dockerfilu a příkazu `docker build`. Dále existuje `docker-compose.yaml` soubor, který má následující obsah:

```
#version: '1'
services:
  ghdata:
    image: ghdata:1.0
    depends_on:
      - postgres
  postgres:
    image: postgres:14.1-alpine
    environment:
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres-data:/var/lib/postgresql/data
  nginx:
    image: nginx
    depends_on:
        - ghdata
    ports:
      - 443:443
    volumes:
      - ./nginx/conf.d:/etc/nginx/conf.d
```

[1] docker.com

```
volumes:
  postgres-data:
    driver: local
```

Jsou v něm definované 3 servisy. Ghdata, postgres a nginx. Ghdata vychází z image, který jsem vytvořil. Postgres je další služba, která stahuje image postgresu verze 14.1. Poslední je nginx, což je webový server, který slouží jako reverzní proxy pro můj backend.

5.1.2 Reverzní proxy

V `docker-compose.yaml` je v odrážce volumes u nginx namapovaný /nginx/conf.d adresář na /etc/nginx/conf.d v kontejneru nginxu. Adresář nginx na hostovském počítači vypadá takto:

```
  nginx/
  |-- conf.d/
  |   |-- default.conf
  |   `-- keys/
  |       |-- fullchain.pem
  |       `-- privkey.pem
```

Nginx běžící v dockeru použije nastavení z default.conf na hostovském počítači. Hledá veškeré validní .conf soubory, které v adresáři conf.d jsou. To je vidět v defaultním konfiguračním souboru nginxu:

```
http {
    ...
    include /etc/nginx/conf.d/*.conf;
}
```

Obsah mého `default.conf` slouží k zejména k nastavení reverzní proxy a definování cest k SSL klíčům.

```
server {
  listen              443 ssl;
  server_name         vyzkumodolnosti.felk.cvut.cz;
  ssl_certificate     /etc/nginx/conf.d/keys/fullchain.pem;
  ssl_certificate_key /etc/nginx/conf.d/keys/privkey.pem;
  ssl_protocols       TLSv1.2 TLSv1.3;
  location / {
     proxy_pass http://ghdata:8090;
  }
}
```

Nginx poslouchá pouze na portu 443, který je rezervovaný pro komunikaci přes šifrovaný protokol HTTPS. Dále v `location` definuji, kam se mají přesměrovávat HTTPS dotazy, které na nginx přijdou. Je to právě na můj backend ghdata. Můžeme si všimnout pojmenováni ghdata v adrese. Je to právě ghdata kvůli tomu, že nginx existuje společně s backendem v dockerem vytvořené síti, kde existují domény vytvořené na základě jména servicy definované v `docker-compose.yaml`. V souboru jsou dále i cesty k SSL certifikátu a SSL klíči.

5.1.3 Zajištění obnovy certifikatu

Soubory fullchain.pem a privkey.pem je potřeba průběžně aktualizovat. Certifikát je platný typicky jenom cca 3 měsíce. Pro vygenerování certifikátu jsem využil nástroj `certbot`. Nástroj také sám vytvořil system.d servicu, která každý den kontroluje, zda vygenerovaný certifikát brzy nevyprchá. Jestliže zjistí, že má certifikát vyprchat, vygeneruje nový. Při vygenerování nového certifikátu také dojde ke spuštění scriptu, který vygenerovaný certifikát přemístí do složky, kde k němu má přístup kontejnerizovaný nginx a následně nginx restartuje. Obnova certifikátu tedy probíhá automaticky.

5.2 Frontend

U frontendu už je nastavený continuous deployment. Docílil jsem toho tak, že používám webhosting od Vercelu [2], který je na CD dobře připravený. Je také vhodný na deploy aplikací napsaných v Next.js, což moje uživatelské prostředí je. Na GitHubu mám větev, do které když odešlu novou verzi projektu, tak se tato verze na webhostingu automaticky deployne. Musel jsem dát Vercelu k větvi přistup k čtení. Frontend je nasazen na doméně https://dotazniky-vo.vercel.app/.

[2] vercel.com

Kapitola 6
Závěr

Cílem práce bylo vytvořit systém pro dotazníky pro projekt Výzkum Odolnosti. K tomu bylo potřeba sesbírat požadavky k systému, následně naimplementovat a otestovat uživatelské rozhraní pro vyplňování dotazníků a naimplementovat a otestovat backend, který dotazníky přijímá.

Sběr požadavků probíhal na pravidelných týdenních setkaní se stranou z ČVUT a UNOBu. Bylo potřeba si definovat, kdo vytvoří jako část v celém celku systému. Docházelo k domluvě, kdo co udělá, také byl vytvořen analytický slovník, Bylo domluveno, že respondenti budou vidět vždy pouze jednu otázku na obrazovce, že se respondenti nebudou moct vracet k otázkám zpátky, že dotazníky bude možné kombinovat nebo že bude probíhat validace vstupů.

K frontendu byla využita knihovna Next.js, která má v sobě React.js. Nejprve jsem začínal vytvářením všech možných vstupů a k nim náležitým validacím. K stavěni UI jsem používal knihovny Bootstrap 5, abych měl jistotu, že UI bude stabilní na zařízeních všech velikostí. Dále jsem pokračoval definováním souboru, který bude uchovávat veškeré otázky. Tento soubor obsahuje JavaScript objekty, které v sobě mají otázky k jednotlivým dotazníkům. Byl dlouhý proces přijít k finální podobě této struktury. Bylo potřeba uchovávat informaci nejen o tom, jak se má otázka generovat a validovat, ale také bylo potřeba myslet na to, že se následně zašle na backend k výpočtu.

Základ backendu již existoval z předchozího chodu projektu. Backend byl tedy potřeba rozšířit. Je na něm použita vicevrstevnatá architektura s frameworkem Spring Boot a je psán v Javě. Na backendu bylo potřeba zejména definovat nové entity pro ukládání dotazníků a jejich výpočtů, k tomu vytvořit repository vrstvu. Do business vrstvy jsem vložil funkcionality pro výpočet vyhodnocení dotazníků a správné mapování DTO objektů na JPA entity. Na backendu bylo také potřeba nastavit bezpečnostní funkce jako třeba CORS restrikci na cílovou doménu.

Data jsou ukládána v PostgreSQL databází, což je open sourceová relační databáze. Většina tabulek vznikla vygenerováním z hibernatu na základě definovaných entit v Javě, nicméně během práce na projektu se běžně stávalo, že jsem psal vlastní SQL scripty. Psal jsem je např., když jsem potřeboval měnit sloupečky v tabulkách, přidávat constrainty nebo přenášet data napříč tabulkami.

Otestoval jsem backend i frontend. Na backendu jsem psal jednotkové testy, které testují správnost výpočtů dotazníků. Také jsem napsal jednotkové testy pro svou TimeComponent třídu, která převádí mezi různými formáty času. Na frontendu jsem napsal automatické end to end testy pomocí nástroje Cypress.js, který proklikává uživatelské rozhraní. Na frontendu těmito testy testuji, že se vygenerovali komponenty, jak měly, nebo že se správně validuje vstup. I backend i frontend jsem zároveň testoval manuálně jak během vývoje, tak po nasazení na ČVUTí server. Byly provedeny také uživatelské testy.

Backend a databáze jsem nasadil na virtuální server. Backend stojí za nginx reverzní proxy, která poslouchá na portu 443. Virtuální server mi byl poskytnut od ČVUT. K tomu, abych celou svou aplikaci spustil, používám Docker. Napsal jsem

`docker-compose.yaml` soubor, který spouštím přes docker-compose příkaz. Ten zajistí, že se vždy nejprve spustí databáze, poté backend a poté nginx s reverzní proxy. Frontend jsem nasadil přes hosting vercelu. Vercel nabízí jednoduchý deploy pro aplikace napsané v Next.js.

Literatura

[1] Cohen, S., T. Kamarck a R. Mermelstein. A global measure of perceived stress (Globaln m eř tko vn maneho stresu). *Journal of Health and Social Behavior (Časopis o zdraví a sociálním chování)*. 1983, ročník 24, č. 4, s. 385-396.

[2] Cohen, S. a G. Williamson. Perceived stress in a probability sample of the United States (Vnímaný stres v pravděpodobnostním vzorku Spojených států). Newbury Park, CA: Sage, 1988, s. 31-67.

[3] Horne, J. A. a O. Ostberg. A self-assessment questionnaire to determine morningness-eveningness in hum (Sebehodnotící dotazník pro stanovení ranní večernosti u lidí), 1976.

[4] Roenneberg, T., T. Kuehnle, M. Juda, T. Kantermann, K. V. Allebrandt, M. Gordijn a M. Merrow. Epidemiologie lidských cirkadiánních hodin. *Sleep Medicine Reviews*. 2007, ročník 11, č. 3, s. 163-178.

[5] Duffy, J. F., D. W. Rimmer, C. A. Czeisler a D. J. Dijk. Vztah endogenních cirkadiánních melatoninových a teplotních rytmů k vlastní preferenci ranní nebo večerní aktivity. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*. 1999, ročník 277, č. 6, s. R1170-R1175.

[6] Roenneberg, T., A. Wirz-Justice a Merrow M. Life between clocks. daily temporal patterns of human chronotypes. *J Biol RhythmsFeb;*. 2003, ročník 18, č. 1, s. 80-90. Dostupné na `https://www.thoracic.org/members/assemblies/assemblies/srn/questionaires/mctq.php`.

[7] Roenneberg, T., T. Kuehnle, M. Juda, T. Kantermann, K. Allebrandt, M. Gordijn a M. Merrow. Epidemiologie lidských cirkadiánních hodin. *Sleep Medicine Reviews*. 2007, ročník 11, č. 3, s. 179-187.

[8] Kantermann, T., M. Juda, M. Merrow a T. Roenneberg. The Munich Chronotype Questionnaire: a tool for validating self-reported circadian phenotypes (Mnichovský dotazník chronotypu: nástroj pro ověřování cirkadiánních fenotypů uváděných samotnými uživateli). *Sleep Medicine*. 2008, ročník 9, č. 8, s. 846-852.

[9] Fahrenberg, J. Dotazník životní spokojenosti (LSQ). New York, NY: Cambridge University Press, 1992, s. 315-324.

[10] Buysse, D. J., C. F. Reynolds, T. H. Monk, S. R. Berman a D. J. Kupfer. The Pittsburgh sleep quality index: A new instrument for psychiatric practice and research. *Psychiatry Research*. 1989, ročník 28, č. 2, s. 193-213. Dostupné na `https://doi.org/10.1016/0165-1781(89)90047-4`.

[11] GeeksForGeeks. *Spring Boot – Architecture* . [vid. 2023-01-11]. Dostupné na `https://www.geeksforgeeks.org/spring-boot-architecture/`.

[12] GeeksForGeeks. *Hibernate – Difference Between ORM and JDBC* . [vid. 2023-01-11]. Dostupné na `https://www.geeksforgeeks.org/hibernate-difference-between-orm-and-jdbc/`.

[13] Ing. David Kadleček, PhD. *Přednáška předmětu Objetkový Návrh A Modelování na téma Patterny pro UI na FEL ČVUT* . [vid. 2023-01-11]. Dostupné na `https://cw.fel.cvut.cz/b211/_media/courses/b6b36omo/13_-_patterny_pro_ui.pptx_2_.pdf`.