VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

TVORBA UŽIVATELSKÉHO ROZHRANÍ PRO PRŮMYSLOVÝ SYSTÉM TEST-IT-OFF

CREATION OF USER INTERFACE FOR THE TEST-IT-OFF INDUSTRY SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB TABAČEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VÍTĚZSLAV BERAN, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148323

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Tabaček Jakub**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Tvorba uživatelského rozhraní pro průmyslový systém Test-it-off**
Kategorie:	Uživatelská rozhraní
Akademický rok:	2022/23

Zadání:

1. Nastudujte postupy UX a technologii React. Seznamte se s průmyslovým robotickým systémem Test-it-off.
2. Proveďte průzkum požadavků na uživatelské rozhraní pro systém Test-it-off. Soustřeďte se zejména na analýzu případů použití uživatelské role „Operátor“. Dle zjištěných požadavků navrhněte vhodné rozmístění a grafickou podobu jednotlivých prvků. Zajistěte, aby systém byl vhodně použitelný pro funkce jako ovládání práce systému, prezentace běhových informací a výkonnostních statistik a základní nastavení konkrétní průmyslové stanice Test-it-off.
3. Navržený systém implementujte pomocí webového frameworku React a napojte na mikro-servisní architekturu systému Test-it-off.
4. Výsledek otestujte, vyhodnoťte z pohledu UX a dle potřeby proveďte potřebné změny a navrhněte možná rozšíření podporující editaci a správu úloh prováděných průmyslovou stanicí Test-it-off.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Beran Vítězslav, Ing., Ph.D.**
Konzultant:	Ing. Radek Štourač
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato práce je zaměřena na návrh, vývoj a testování webového rozhraní pro operátory robotického pracoviště. Pro řešení problému byl použitý nejpopulárnější reaktivní framework React a vědomosti získané z analýzy uživatelů a jejich potřeb. Výstupem práce je přehledné webové rozhraní, navrženo dle doporučených praktik UX, umožňující operátorům jednoduše a přehledně spouštět a monitorovat běh projektů.

Abstract

This work is focused on the design, development and testing of a web interface for operators of a robotic system. To solve the problem, the most popular reactive framework React and knowledge obtained from the analysis of users and their needs were used. The output of the work is an intuitive web interface, designed according to recommended UX practices, enabling operators to simply and easily launch and monitor the progress of projects.

Klíčová slova

webové aplikace, frontend, uživatelská zkušenost, analýza uživatelských požadavků, návrh rozhraní, testování, robotické pracoviště, typescript, react, tailwindcss, long-polling

Keywords

web application, frontend, user experience, user requirements analysis, design, testing, robotic workplace, typescript, react, tailwindcss, long-polling

Citace

TABAČEK, Jakub. *Tvorba uživatelského rozhraní pro průmyslový systém Test-it-off.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vítězslav Beran, Ph.D.

Tvorba uživatelského rozhraní pro průmyslový systém Test-it-off

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D. Další informace o samotném produktu a jeho uživatelích mi poskytli Bc. Radek Štourač a Mgr. Martin Červinka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Tabaček
9. května 2023

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Vítězslavovi Beranovi, Ph.D. za odbornou pomoc a usměrnění, které mi pomohli při vypracovávání této práce. Dále bych rád poděkoval kolegům v Kinali, kteří mi vždy rádi pomohli a nasměrovali mě správnou cestou během řešení problémů.

Obsah

1 Úvod **2**

2 Technologie pro tvorbu webových aplikací **3**
2.1 Nástroje a jazyky pro tvorbu GUI 3
2.2 Aplikační logika a technologie na straně serveru 9

3 Teorie návrhu **13**
3.1 Návrhové principy 13
3.2 Testování 14
3.3 Popis robotického pracoviště 16

4 Návrh aplikace **18**
4.1 Charakterizace uživatele 18
4.2 Analýza uživatelských potřeb 18
4.3 Iterace návrhu 21
4.4 Vytváření a úprava assetů 24
4.5 Návrh řešení 27

5 Implementace **31**
5.1 Funkční celky 31
5.2 API 31
5.3 Komunikace s API 34
5.4 Implementace v knihovně React 35
5.5 Funkční testování 38
5.6 Uživatelské testování 39

6 Závěr **40**

Literatura **41**

A Obsah přiloženého paměťového média **43**

Kapitola 1

Úvod

Robotické systémy nalézají čím dál tím více uplatnění v moderních výrobních prostředích, kde napomáhají automatizaci úkolů a zvyšování efektivity a produktivity práce. Typickým příkladem může být robotická stanice integrující různé hardwarové a softwarové nástroje, aby zajistila kontrolu kvality finálního produktu. Každá taková stanice vyžaduje ovládací systém, který volá předprogramované instrukce na dostupném hardwaru a zároveň sbírá všechny informace generované či získané během práce.

Důležitým prvkem tohoto systému je grafické uživatelské rozhraní (GUI), které operátorům poskytuje celkový přehled o dění na pracovišti, včetně důležitých událostí, detekovaných závad či problémů s hardwarem nebo s během programu. Kvalitní uživatelské rozhraní umožňuje operátorům rychle a přesně identifikovat a řešit chyby, což přispívá ke zvýšení celkové efektivity.

Hlavním cílem této práce je navrhnout a implementovat přehlednou a uživatelsky přívětivou webovou aplikaci, propojenou s existující backend implementací pomocí REST API. Aplikace bude sloužit především k řízení projektů a monitorování akcí a procesů na pracovišti. Součástí monitorování budou statistiky, které poskytnou přehled o sledovaných hodnotách jako například počet OK/NOK výrobků nebo čas cyklu.

Tato práce je rozdělena do čtyř kapitol. První kapitola pojednává o technologiích použitých pro implementaci, zahrnující základní technologie jako HTML, CSS a JavaScript, ale i pokročilé knihovny a frameworky, jako React.js, Vite, TailwindCSS a jiných. Zároveň je v kapitole zmíněn i proces napojování webové aplikace na již existující API a technologie využité k zobrazování dat uživateli v reálném čase. Druhá kapitola se věnuje dalším teoretickým částem, jako například principům UX nebo popisu robotického pracoviště, kterým bylo nutné porozumět pro úspěšné vytvoření návrhu a následnou implementaci. Třetí kapitola řeší výzkum uživatelských potřeb a jednotlivé iterace grafického návrhu, přičemž uvádí i plánované rozšíření funkcionality ve formě tzv. Asset editoru. Ve čtvrté kapitole jsou popsané klíčové části celkové implementace aplikace a některé zajímavé aspekty. Na konci této kapitoly se následně zmiňuje testování aplikace, jak prostřednictvím automatizovaných testů s využitím nástroje Cypress, tak i uživatelské testování finální implementace. V závěru práce jsou uvedeny možnosti pro budoucí rozšíření a vylepšení aplikace.

Kapitola 2

Technologie pro tvorbu webových aplikací

Tato kapitola popisuje současné technologie pro tvorbu webových aplikací, které se během vývoje využívaly. V rámci kapitoly je kladen velký důraz na popsání frontendové části, jelikož ta byla hlavním cílem práce. V případě backendu bylo potřeba pouze nastudovat existující API a navázat komunikaci.

2.1 Nástroje a jazyky pro tvorbu GUI

Při frontendovém vývoji je důležité zvolit správné nástroje a jazyky. Frontend se v kontextu webového vývoje vztahuje k návrhu a implementaci grafického uživatelského rozhraní (GUI) a uživatelského prožitku (UX) webové aplikace. Zahrnuje vizuální komponenty a interaktivní prvky, se kterými koncoví uživatelé přímo interagují. Využívá kombinaci HTML, CSS a JavaScriptu k vykreslení rozvržení, stylu a funkcionality webové stránky. Hlavním cílem frontendového vývoje je vytvořit intuitivní, esteticky přitažlivý a responzivní UX, splňující potřeby a očekávání cílové skupiny.

HTML

Hypertext Markup Language (HTML) je značkovací jazyk, který slouží jako základní stavební kámen pro vytváření a strukturování obsahu na internetu [18]. V rámci projektu se využíval na definici obsahové struktury aplikace, avšak výsledné HTML se pouze dynamicky generovalo díky použití jazyka TSX[1]. Aktuální verzí je HTML5, která byla oficiálně zveřejněna v roce 2014, a její správu a vývoj zajišťuje mezinárodní organizace World Wide Web Consortium (W3C) [8].

Jazyk HTML je založen na značkách, které se používají pro definici struktury a obsahu webové stránky. Tyto značky, často nazývané tagy či HTML elementy, jsou ohraničeny symboly „<“ a „>“ (např. `<html>`, `<button>`, `<h1>`). Většina značek je párová, což znamená, že označují začátek a konec elementu. Některé značky však mohou být nepárové, jako například `<img>` nebo `<br>` (viz obrázek 2.1). Každý tag může mít i atributy, díky kterým elementy nabývají dalších vlastností, například identifikátory, třídy či stylování. HTML

[1] Pomocí TSX (TypeScript XML) lze definovat a manipulovat komponenty webových aplikací. Ty se následně překládají do funkcí JavaScriptu pro dynamické generování a aktualizaci HTML stránek v prohlížeči.

dokumenty mají obvykle kořenový element označený jako `<html>`, který obsahuje dva hlavní elementy - `<head>` pro metadata a informace o stránce, a `<body>` pro vlastní obsah stránky.

```
1 <img src="test.png" alt="Test image" class="img-rounded" width="500" height="500" />
```

Obrázek 2.1: **Příklad kódu jazyka HTML.** Červeně zvýrazněný je tag, v tomto případě nepárový `<img />`, žlutě lze vidět atributy a zeleně hodnoty atributů. V případě párového elementu by se hodnota psala mezi hraniční tagy jako například `<h1>test</h1>`.

CSS

Cascading Style Sheets (CSS) je stylovací jazyk používaný pro popis vizuálního vzhledu a formátování webových stránek napsaných v HTML a XML[2]. V projektu byla snaha omezit psaní čistého CSS kódu. Místo toho se CSS styly aplikovaly pomocí knihovny TailwindCSS 2.1. CSS umožňuje oddělit obsah stránky od designu, což zlepšuje přehlednost a udržitelnost kódu, usnadňuje úpravy vzhledu a zajišťuje konzistentní prezentaci obsahu napříč různými zařízeními a prohlížeči [18]. CSS je standardizováno organizací World Wide Web Consortium (W3C) a aktuální verzí je CSS3.

V CSS lze definovat styly pro jednotlivé HTML elementy nebo skupiny elementů, které mají společné vlastnosti, a tyto styly se poté automaticky aplikují na všechny příslušné elementy na webové stránce. Styly mohou být definovány v rámci HTML dokumentu, v externím souboru s příponou `.css` nebo přímo v atributu HTML elementu. Doporučovaný a zároveň nejčastější přístup je pomocí externích souborů, umožňuje to snazší správu nebo úpravu stylů napříč celým projektem.

Syntaxe CSS se skládá ze selektorů, které identifikují elementy nebo skupiny elementů, na které se mají styly aplikovat, a deklarací, které definují konkrétní styly. Selektory mohou být založeny na názvech HTML elementů, třídách, identifikátorech, atributech nebo i na vztazích mezi elementy. Deklarace se skládají z vlastností a hodnot oddělených dvojtečkou a uzavřených do složených závorek viz. obrázek 2.2.

```
1 img {
2   margin: 10px 20px;
3 }
4
5 img .img-rounded {
6   border-radius: 20px;
7 }
```

Obrázek 2.2: **Příklad selektorů v jazyce CSS.** Ukázka se částečně vztahuje ke kódu HTML v předchozí ukázce. První selektor nastavuje každému elementu `<img>` na stránce odsazení od ostatních o 10px na ose y a 20px na ose x. Druhý selektor vybírá pouze obrázky s atributem třídy *img-rounded* a aplikuje jim zaoblení rohů.

Aktuální verze CSS má řadu pokročilých funkcí, pro rozvržení, mobilní optimalizaci nebo interakci bez použití jazyka JavaScript. Nástroje k vytváření responzivních a fle-

[2]XML je jazyk velice podobný HTML, umožňuje však uživatelům definovat i vlastní značky. Používá se však více pro uchovávání a výměnu dat, ne pro rozložení webových stránek.

xibilních rozvržení jsou například flexbox nebo grid, které plně nahrazují staré způsoby zarovnávání pomocí tabulek a jiných. Pro možnosti interakce CSS nabízí například pseudo-třídy, jako jsou `:hover` (myš přes element), `:focus` (aktivní textové pole) nebo `::after` (pseudo-element, který je posledním potomkem vybraného prvku - používá se k přidání kosmetického obsahu do prvku), které umožňují aplikovat styly na HTML elementy v závislosti na jejich stavu nebo interakce s uživatelem (viz. obrázek 2.3). V neposlední řadě se pro další možnosti optimalizace zavedly takzvané media queries, které umožňují vývojářům stylovat obsah dle aktuálních dimenzí obrazovky zařízení.

```css
a:hover::after {
  content: " (Hover)";
}

@media (max-width: 600px) {
  a:hover::after {
    content: " ( Hover)";
  }
}
```

Obrázek 2.3: **Pokročilé funkce CSS.** Na obrázku pseudo-prvek `::after` přidává text za odkazem, když je nad odkazem kurzor myši. Následná media query mění obsah pro zařízení s maximální šířkou 600px tak, že přidá do textu ikonu telefonu.

JavaScript

JavaScript je vysokoúrovňový, interpretovaný programovací jazyk, který je integrální částí moderního webového vývoje. Jeho primárním účelem je přidávání interaktivity a dynamického chování webovým stránkám, což zahrnuje manipulaci s DOM (Document Object Model)[3], práci s událostmi, validaci formulářů, komunikaci se serverem a mnoho dalších. JavaScript je populární a široce používaný jazyk, který je podporován většinou moderních webových prohlížečů a zařízení.

JavaScript byl v roce 1995 vyvinut Brendanem Eichem u společnosti Netscape Communications. Původně se nazýval Mocha a později LiveScript, ale nakonec byl přejmenován na JavaScript, aby zdůraznil jeho podobnost s populárním jazykem Java, ačkoliv tyto dva jazyky mají zcela odlišný základ a koncepty [18]. JavaScript byl standardizován v roce 1997 organizací ECMA International. Aktuálně je nejnovější verze ECMAScript 2022 (ES2022) [12].

JavaScript se používá jak na straně klienta (v prohlížeči), tak na straně serveru (například s pomocí Node.js). Díky této univerzálnosti se stal jedním z nejpoužívanějších a nejžádanějších programovacích jazyků na světě. Jeho popularita ještě vzrostla s nástupem knihoven a frameworků, jako jsou jQuery, Angular, React nebo Vue.js, které zjednodušují a zrychlují vývoj webových aplikací [6].

Syntaxe JavaScriptu je velmi podobná syntaxi jazyka C. Jazyk podporuje proměnné, funkce, objekty, pole, cykly a podmíněné příkazy. Jazyk se však v rámci nových ECMA standardů neustále vyvíjí aby umožňoval psaní přehlednějších a rychlejších kódů. Názornou ukázku syntaxe lze vidět na obrázku 2.4.

[3]DOM je stromová struktura, která reprezentuje webové dokumenty a umožňuje jejich manipulaci. Každý uzel stromu představuje element, atribut nebo text, a vývojáři mohou tyto uzly procházet a upravovat. [18]

```javascript
// Deklarace proměnné a přiřazení hodnoty
let greeting = "Hello, World!";

// Definice funkce pro zobrazení pozdravu
function displayGreeting() {
  console.log(greeting);
}

// Zavolání funkce
displayGreeting();
```

Obrázek 2.4: **Příklad jednoduché definice a volání funkce v JS.** Jednotlivé řádky jsou popsané v kódu.

Jedním z klíčových konceptů jazyka JavaScript je asynchronní programování. To umožňuje provádět úkoly, jako je načítání dat ze serveru nebo zpracování uživatelských vstupů, bez blokování hlavního vlákna a zpomalení prohlížeče. JavaScript používá mechanismy, jako jsou *callbacks*, *promises* a *async/await*, pro docílení asynchronního chování a zajištění hladkého běhu aplikace.

TypeScript

TypeScript je založen na standardu ECMAScript a rozšiřuje syntaxi o statické typy, což umožňuje lepší kontrolu nad kódem, lepší čitelnost a analýzu chyb. TypeScript je nadmnožinou JavaScriptu, což znamená, že veškerý platný JavaScriptový kód je zároveň platným TypeScript kódem. Kód napsaný v jazyce TypeScript je před spuštěním v prohlížeči nebo na serveru přeložen (transpilován) do čistého JavaScriptu [1].

Velká výhoda použití jazyka TypeScript v rámci této práce, je zlepšení práce s externím API. Pomocí balíčku openapi-typescript a schémy API dostupné v nástroji Swagger 2.2 lze jednoduše generovat typové definice pro data, která se mohou z API vrátit. Díky tomu lze hned zjistit, jaká data lze od požadavku očekávat a jak s nimi pracovat.

Reaktivní frameworky

V moderním webovém vývoji se stále více uplatňují reaktivní frameworky, které usnadňují vytváření interaktivních a dynamických webových aplikací. Tyto frameworky umožňují vývojářům snadno pracovat s daty a stavem aplikace, čímž se zvyšuje čitelnost a kvalita kódu. Mezi nejpopulárnější a nejrozšířenější reaktivní frameworky patří React, Angular a Vue.js [3]. Kromě těchto hlavních frameworků existují i další, které se snaží zaměřit na specifické účely jako například Svelte, Qwik, Ember.js nebo Backbone.js.
React:

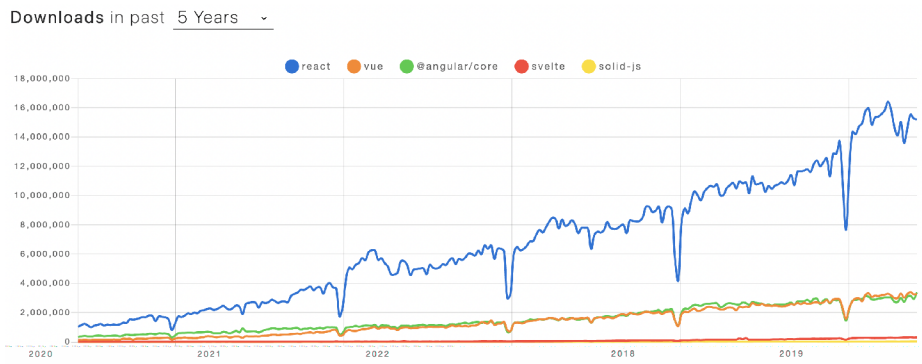
- Vyvinutý společností Facebook.
- Zaměřuje se na jednosměrný tok dat a modulární komponenty.
- Má širokou komunitní podporu a spoustu dostupných balíčků pro různé účely.
- Ideální pro vývoj velkých a komplexních aplikací.
- Učební křivka je středně obtížná, obtížnost klesá po pochopení základních konceptů.

Angular:

- Vyvinutý společností Google.
- Založený na komponentách a používá obousměrný data binding.
- Silně závislý na jazyku TypeScript a jeho funkčnosti.
- Má silnou podporu ze strany Google a rozsáhlou komunitu.
- Učební křivka je obtížná, zejména pro začátečníky.

Vue.js:

- Vyvinutý Evanem You.
- Kombinuje nejlepší vlastnosti Reactu a Angularu, jako jsou komponenty a obousměrný data binding.
- Má lehký základ a menší velikost souborů, což zajišťuje rychlejší načítání aplikací.
- Komunita je menší než React a Angular, ale stále silná a rychle roste.
- Je snadno použitelný a má lehčí učební křivku než React a Angular.



Obrázek 2.5: **Porovnání popularity nejznámějších reaktivních frameworků.** Neexistuje žádný definitivní ukazatel popularity, zde byla vybrána statistika stažení balíčků ze serveru npm 2.1. Převzato z [14].

Při zkoumání rozdílů mezi "vanilla javascriptem" (tedy bez jakýchkoliv knihoven) nebo jQuery a reaktivními frameworky je důležité si uvědomit, že tyto technologie se liší ve způsobu, jakým manipulují s DOM a jak řeší aktualizaci uživatelského rozhraní. Čistý JavaScript a jQuery používají imperativní přístup, což znamená, že programátor musí explicitně určit, jaké změny se mají provést v DOM (přidání, odstranění nebo úprava elementů). Tento přístup může být náchylný k chybám a složitější, protože vyžaduje, aby programátor pečlivě sledoval stav DOM a zajišťoval jeho konzistenci.

Na druhou stranu, reaktivní frameworky nabízejí deklarativní přístup k ovládání DOM. Místo toho, aby programátor explicitně určoval, jaké změny se mají provést, reaktivní frameworky umožňují definovat stav a závislosti mezi komponentami. Když dojde ke změně stavu, framework automaticky aktualizuje DOM, aby odpovídal novému stavu. Tento přístup zjednodušuje správu stavu aplikace a snižuje riziko chyb, protože programátor nemusí ručně řešit změny v DOM [5]. Zjednodušení správy lze vidět na obrázku 2.6.

Obrázek 2.6: **Diagramy znázorňující rozdíl v přístupu k DOM.** Rozdíl mezi jQuery (vlevo) a React (vpravo). Z diagramu lze jednoduše poznat, při kterém přístupu vzniká přehlednější a spolehlivější kód.

V rámci tohoto projektu byl po důkladném zvážení zvolen framework React na základě jeho přijatelné učební křivky, dobré podpory ze strany komunity a stabilní aktuální verzi. V projektu se také využívá hodně rozšíření Reactu, ať už oficiálních nebo komunitních. Z těch oficiálních například react-router pro jednoduchý management navigace po aplikaci nebo Recoil.js pro přehlednou správu globálního stavu aplikace pomocí tzv. atomů a selektorů. Mezi komunitní rozšíření použité v projektu patří například recharts [7] pro zobrazování grafů nebo react-i18next pro jednoduché použití překladů v celé aplikaci.

TailwindCSS a DaisyUI

Tailwind CSS je moderní CSS framework, který se zaměřuje na rychlé a efektivní vytváření uživatelských rozhraní [16]. Narozdíl od tradičních CSS frameworků, jako je Bootstrap nebo MaterialUI neposkytuje předdefinované komponenty, ale místo toho nabízí sadu nízkoúrovňových *utility* tříd, které vývojáři mohou kombinovat a upravit dle potřeby bez nutnosti psaní vlastního CSS kódu (viz obrázek 2.7). Tento přístup umožňuje větší flexibilitu a kontrolu nad vzhledem a chováním komponent, což vede k unikátním a udržitelným uživatelským rozhraním.

```
1 <button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
2     Click me
3 </button>
```

Obrázek 2.7: **Ukázka komponentu s *utility* třídami** V tomto příkladu tlačítko využívá třídy Tailwind CSS pro nastavení stylů. Tlačítko má modré pozadí, které se ztmaví při najetí myši. Text je bílý a tučný a tlačítko má také mírně zaoblené rohy.

Jedním z populárních rozšíření Tailwind CSS je DaisyUI, které nabízí sadu předdefinovaných komponent navržených pro rychlý vývoj uživatelských rozhraní [21]. Nejedná se však o velké komponenty jako v klasických CSS frameworcích, ale spíše o tzv. *atomy*, případně *molekuly*. Zahrnuje tedy komponenty jako tlačítka, formulářové pole, karty, navigace a mnoho dalšího. Tyto komponenty jsou založeny na Tailwind CSS třídách a lze je snadno přizpůsobit a rozšířit dle potřeby.

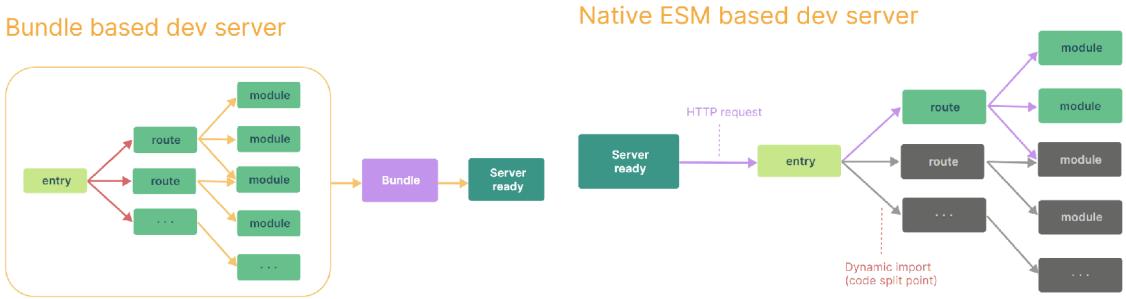
Tailwind CSS a DaisyUI také podporují různá nastavení tématu a barev, což umožňuje snadno přizpůsobit vzhled aplikací podle požadavků klienta. Díky tomuto přístupu

lze rychle experimentovat s různými vzhledy a barvami, aniž by bylo potřeba upravovat základní CSS kód.

Build

Sestavení celé aplikace bylo řešeno pomocí nástroje Vite. Vite běží na Node.js, což je běhové prostředí pro JavaScript postavené na V8 JavaScriptovém enginu od Googlu. Node.js umožňuje spouštět JavaScriptový kód na straně serveru, což usnadňuje vytváření a správu webových aplikací. Všechny balíčky Node.js aplikace jsou stahovány a spravovány pomocí NPM (Node Package Manager). NPM umožňuje stahování a instalaci balíčků a jejich závislostí do projektů a zjednodušuje tím práci s externími knihovnami [22].

Během vývoje se Vite zaměřuje na poskytování rychlého načítání modulů aplikace a "Hot Module Replacement"(HMR) - tedy obnovení komponentů v aplikaci při vývoji bez nutnosti obnovení stránky, což zkracuje čas potřebný k testování změn v aplikaci (viz. obrázek 2.8). Vite umožňuje rychlý vývojový cyklus a minimalizuje čekání na aktualizace kódu [2].



Obrázek 2.8: **Srovnání Vite a jiných nástrojů.** Vite poskytuje zdrojový kód přes nativní ESM, tedy potřebuje pouze transformovat a poskytovat zdrojový kód na vyžádání, jak to prohlížeč požaduje. Kód za podmíněnými dynamickými importy je zpracován pouze tehdy, je-li skutečně použit na aktuální obrazovce v prohlížeči. Převzato z [2].

Při sestavení aplikace Vite optimalizuje kód pro produkční nasazení, což zahrnuje minifikaci[4], odstranění nepotřebného kódu (tree shaking)[5] a další optimalizace, které zlepšují výkon výsledné aplikace. Výsledný produkční build je navržen tak, aby byl co nejefektivnější a rychlý pro konečné uživatele.

2.2 Aplikační logika a technologie na straně serveru

Backend je část webové aplikace nebo systému, která se zabývá zpracováním a správou dat, logiky a funkcí, které nejsou přímo viditelné uživateli. Backend často zahrnuje aplikační servery, databázové systémy, API a další infrastrukturu, která umožňuje komunikaci s frontendem, tedy uživatelským rozhraním. Úkolem backendu je zpracovávat požadavky

[4]Minifikace je proces zmenšování velikosti souborů (jako JS, CSS nebo HTML) odstraňováním nadbytečných znaků, jako jsou mezery, nové řádky či komentáře. Zlepšuje výkon aplikace, aniž by ovlivnila její funkčnost [10].

[5]Tree shaking analyzuje závislosti mezi moduly, identifikuje nevyužité části kódu a odstraňuje je. Tímto způsobem se optimalizuje velikost výsledného balíčku [19].

frontendu, provádět nezbytné operace, jako je načítání, ukládání či aktualizace dat, a vracet výsledky zpět.

V případě systému test-it-off se kromě klasických funkcionalit popsaných výše backend stará i o spouštění programů na robotickém stanovišti napsaných v jazyce Python a následné sbírání statistik z těchto programů. Implementace backendu však nebyla součástí této práce, takže je zde zmíněna jenom okrajově a tato část se více zaměřuje na technologie použité při komunikaci s frontendovou částí.

C# API

C# je silně typovaný, objektově orientovaný programovací jazyk, který byl vyvinut společností Microsoft jako součást .NET Frameworku. C# kombinuje prvky z jazyků jako C++, Java a Visual Basic a nabízí vývojářům širokou škálu nástrojů a funkcí pro vytváření různých typů aplikací, včetně webových, desktopových, mobilních a herních aplikací. C# je známý svou bezpečností a škálovatelností, což z něj činí populární volbu pro podnikové a komerční projekty [20].

V C# je napsané celé REST (Representational State Transfer) API. REST je architektonický styl pro vývoj webových služeb, který umožňuje jednoduchou a standardizovanou komunikaci mezi klientem a serverem prostřednictvím HTTP protokolu. API vystavuje zdroje (např. objekty, entity) a definuje operace, které lze s nimi provádět pomocí HTTP metod, jako jsou GET, POST, PUT a DELETE [9].

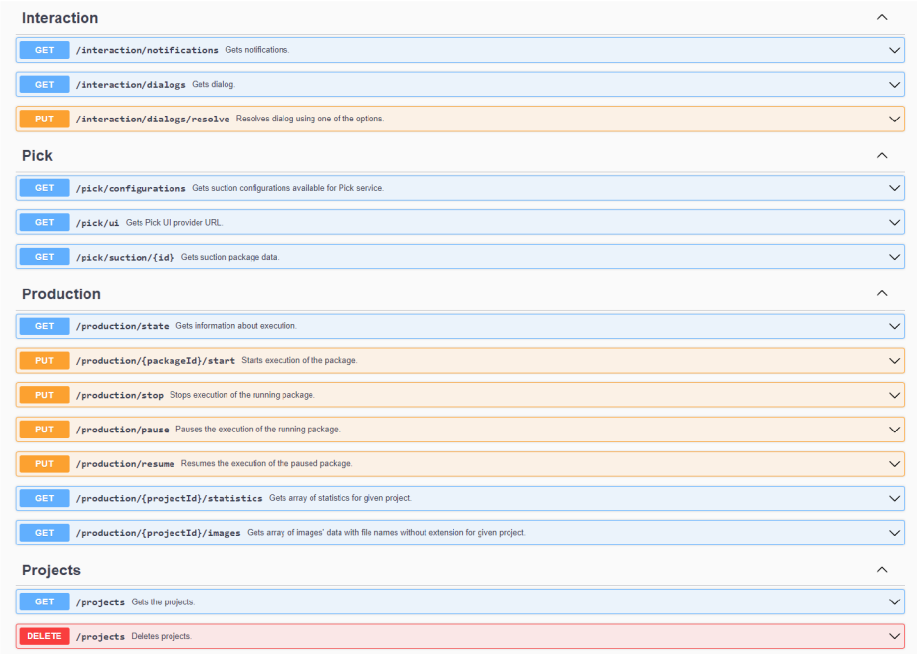
Python

Python je univerzální, vysokoúrovňový programovací jazyk s dynamickým typováním, který byl vyvinutý Guidem van Rossumem v roce 1991. Jazyk Python se vyznačuje jednoduchou a čitelnou syntaxí, což usnadňuje učení jazyka a rychlý vývoj aplikací. Jeho rozsáhlá standardní knihovna a množství externích knihoven pokrývají širokou škálu oblastí, jako jsou například webové aplikace, datová analýza, strojové učení či robotika, což z něj činí vhodný nástroj pro různé účely [4].

Python byl tedy zvolen pro tvorbu programů robotického ramene z několika důvodů. Snadná syntaxe a čitelnost kódu umožňují rychlé a efektivní vytváření skriptů, což je výhodné při prototypování či experimentování s novými funkcemi. Dalším důvodem je již zmiňovaná široká nabídka externích knihoven umožňující jednoduché a rychlé rozšíření funkcí bez vlastního programování. V neposlední řadě je výhodná platformová nezávislost jazyka Python, která umožňuje spouštět skripty na různých operačních systémech a hardwarových konfiguracích bez nutnosti zásadních úprav.

Swagger

Swagger je nástroj pro práci s RESTful API, který se zaměřuje na navrhování, dokumentování, testování a generování klientů pro různé programovací jazyky. Hlavní myšlenkou nástroje Swagger je poskytnout univerzální a snadno použitelný způsob, jak popsat API, aby byla komunikace mezi vývojáři co nejefektivnější. Swagger je založen na OpenAPI specifikaci, což je široce používaný standard pro popis RESTful API. Ukázku nástroje swagger lze vidět na obrázku 2.9.



Obrázek 2.9: **Swagger UI.** Na obrázku lze vidět jednotlivé celky dostupné na API. Každý lze rozbalit a následně získat informace o datech, které je při požadavku nutné poskytnout, a datech, které vrací, popřípadě chybových kódech.

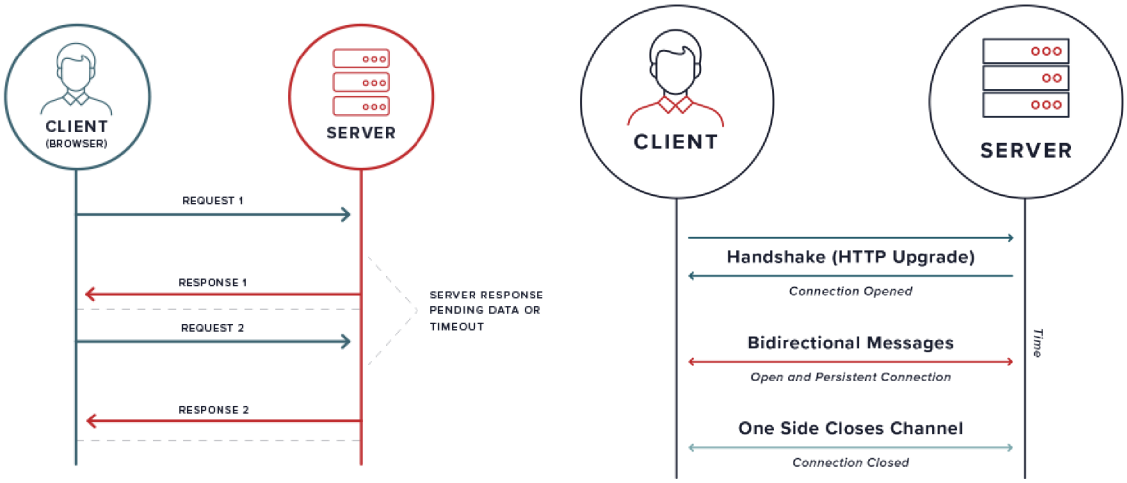
Jednou z klíčových vlastností Swaggeru je jeho schopnost generovat interaktivní dokumentaci API, která je automaticky aktualizována podle změn v API. Tato dokumentace je snadno srozumitelná a umožňuje vývojářům rychle získat přehled o dostupných koncových bodech, jejich parametrech a návratových hodnotách. Swagger UI také usnadňuje testování jednotlivých koncových bodů přímo z webového rozhraní, což zjednodušuje ladění a ověřování chování API [23].

Aktualizace dat v reálném čase

V současnosti se pro řešení problematiky aktualizace dat (například statistik) v reálném čase nejčastěji používají dvě hlavní techniky: long-polling a WebSockety. Tyto metody umožňují efektivní a rychlou aktualizaci informací v uživatelském rozhraní webových aplikací a zajišťují plynulý přenos dat mezi klientem a serverem. Každá má svoje výhody a nevýhody, které budou níže popsány.

Long-polling je technika komunikace mezi klientem a serverem, která umožňuje serveru posílat aktualizace klientovi bez nutnosti klienta neustále se dotazovat na nová data. Při použití long-pollingu klient pošle požadavek na server a čeká na odpověď. Server pak může odpovědět buď okamžitě, pokud jsou k dispozici nové informace, nebo později, až budou nové informace k dispozici. Tento přístup snižuje zátěž na server a zlepšuje výkon komunikace ve srovnání s tradičním pollingem.

Na druhou stranu, WebSockety jsou moderní technologie, která umožňuje plně duplexní, neblokující komunikaci mezi klientem a serverem. WebSockety umožňují otevřít trvalé spojení mezi klientem a serverem, přes které lze posílat zprávy v obou směrech bez nutnosti opakovaně navazovat spojení. Tato technologie je obzvláště vhodná pro aplikace, které vyžadují rychlou a průběžnou výměnu dat, jako jsou herní aplikace nebo chatovací platformy.



Obrázek 2.10: **Porovnání long-polling (vlevo) a websockets (vpravo).** Na diagramu lze vidět porovnání průběhu komunikací mezi klientem a serverem. Převzato z [11].

Kritérium	**Long-polling**	**WebSockety**
Komplexita	Jednodušší na implementaci a použití	Vyžaduje více znalostí a složitější implementaci
Kompatibilita	Podporováno ve starších prohlížečích	Méně kompatibilní se staršími prohlížeči
Výkon	Nižší výkon, zpoždění při přenosu dat	Vysoký výkon, minimální zpoždění při přenosu dat
Zdroje serveru	Může způsobit vyšší zátěž na serveru	Nižší zátěž na serveru díky trvalému spojení
Interakce	Omezené na jednosměrnou komunikaci	Plně duplexní, obousměrná komunikace

Tabulka 2.1: Srovnání long-polling a WebSocketů

Z uvedeného srovnání v tabulce 2.1 vyplývá, že volba mezi long-pollingem a WebSockety závisí na požadavcích konkrétní aplikace, kompatibilitě s prohlížeči a očekávaném výkonu. Pro aplikace, které vyžadují rychlou a průběžnou výměnu dat, jsou WebSockety obecně vhodnější volbou. Zatímco pro jednodušší aplikace, které nevyžadují tak intenzivní interakci, a kde je důležitější kompatibilita se staršími prohlížeči, může být long-polling vhodnější volbou [11].

V rámci tohoto projektu byla zvolena technika long-polling z několika důvodů. Za prvé, byla již částečně implementována v rámci stávajícího API, což umožňuje plynulejší integraci. Za druhé, díky použití long-pollingu mohou být všechny typy komunikace udržovány v souladu se specifikacemi definovanými ve swagger OpenAPI schématu. Toto rozhodnutí podporuje konzistenci získávání dat v projektu a zjednodušuje jeho správu a rozšiřitelnost.

Kapitola 3

Teorie návrhu

V předchozí kapitole byly představeny technologie a nástroje nezbytné pro implementaci. Tato kapitola se tedy zaměřuje na další důležité teoretické aspekty, mezi které patří návrhové principy UX, popis robotického pracoviště a metody průběžného uživatelského testování návrhu a implementace.

3.1 Návrhové principy

Níže popsané jsou základní a důležité návrhové principy UX (User Experience), na které se implementace a návrh zaměřovaly. GUI představují klíčovou část interakce mezi uživatelem a systémem. Je tedy důležité, aby byl kladen důraz na dodržení těchto principů. Díky tomu je interakce se systémem pro uživatele efektivní a příjemná [17].

Hierarchie informací

Zásadní pro zajištění srozumitelnosti a snadného použití GUI je definování hierarchie. Informace by měly být uspořádány tak, aby uživatelé snadno pochopili, jakým způsobem se pohybovat v rozhraní, a mohli rychle nalézt potřebné funkce či informace. Hierarchie by měla zohledňovat důležitost a četnost použití jednotlivých prvků.

Konzistence

Konzistence je klíčová pro snadné a rychlé učení uživatelů při práci s GUI. Stejné prvky a akce by měly mít konzistentní vzhled a chování napříč celým rozhraním. To pomáhá uživatelům lépe pochopit, jak systém funguje, a zvyšuje jejich důvěru v jeho použití.

Zpětná vazba

Dalším důležitým aspektem je informování uživatele o výsledcích jeho akcí. Po provedení akce by uživatel měl obdržet jasnou zpětnou vazbu, aby pochopil, zda akce byla úspěšná, nebo se vyskytl problém. Zpětná vazba může být poskytnuta například formou vizuálních prvků nebo zvuků.

Minimalismus

Minimalismus zlepšuje přehlednost rozhraní a usnadňuje orientaci uživatelů. Spočívá v omezení počtu prvků a informací na nezbytné minimum a zaměření na základní funkce a jejich

snadnou dostupnost. Zbytečné prvky a informace by měly být odstraněny, aby nedocházelo k rozptýlení pozornosti.

Srozumitelná navigace

Uživatelé by měli být schopni se snadno orientovat v rozhraní a rychle najít požadované funkce či informace. Navigační prvky by měly být jasně viditelné, srozumitelně popsané a umístěné na vhodných místech v rámci rozhraní.

Estetika a atraktivnost

Ačkoli by funkčnost měla být prioritou při návrhu GUI, estetická a vizuální stránka rovněž hrají důležitou roli. Dobře navržené a vizuálně přitažlivé rozhraní může zvýšit uživatelskou spokojenost a zlepšit vnímání produktu. Vizuální prvky by měly být v souladu s celkovým vzhledem aplikace či značky zákazníka (zadavatele).

Funkčnost na všech zařízeních

Při návrhu GUI je důležité zohlednit různé platformy a zařízení, na kterých může být aplikace používána. To zahrnuje přizpůsobení designu pro různá rozlišení obrazovky nebo různé způsoby vstupu (například myš s klávesnicí nebo dotyková obrazovka). Uživatelské rozhraní by mělo být responzivní a flexibilní, aby poskytovalo optimální zážitek pro všechny uživatele.

Jazyková lokalizace

Při návrhu by měly být zohledněny různé jazyky, formáty data a času, vědecké jednotky a další specifika daného regionu. Lokalizace zvyšuje celkovou dostupnost a přijetí aplikace či systému na mezinárodní úrovni.

3.2 Testování

Testování je zásadní součástí vývoje software. Zajišťuje jeho kvalitu, spolehlivost a správnou funkčnost. Proces testování spočívá v ověřování, zda výsledný produkt splňuje požadavky a specifikace definované v průběhu návrhu. Testování se provádí na různých úrovních, od jednotkových testů, které kontrolují jednotlivé komponenty, až po integrační a systémové testy, jež se zaměřují na celkovou funkcionalitu systému. Během testování se využívají různé metody a techniky, které mohou být manuální či automatické.

Funkční testování

Funkční (automatické) testování bylo řešeno pomocí nástroje Cypress. Cypress je moderní, open-source nástroj pro testování webových aplikací, který se vyznačuje rychlostí, stabilitou a snadnou integrací do vývojových procesů. Jeho primárním účelem je zjednodušit a zefektivnit proces automatizovaného testování webových aplikací, a tím přispět ke zvýšení kvality a spolehlivosti výsledného software. Cypress umožňuje tvorbu end-to-end, integračních a jednotkových (unit) testů v prostředí JavaScript. Jeho jedinečnou vlastností je schopnost spouštět testy přímo v prohlížeči, což umožňuje přesnější simulaci uživatelských interakcí a usnadňuje ladění chyb.

V rámci projektu byl Cypress využíván na testování funkčních bloků 5.1 před jejich přidáním do hlavní vývojové větve a následné testování při přidání každé další funkcionality, aby se ověřilo, že předchozí funkce nejsou narušeny. Jednalo se tedy hlavně o E2E[1] testy, které psal tester v týmu test-it-off.

Uživatelské testování

Uživatelské testování bylo zaměřeno na tzv. do-it-yourself metodu zkoumání použitelnosti uživatelských rozhraní (UI) podle knihy Steva Kruga [15]. Hlavním cílem testování použitelnosti je sledovat uživatele při práci s produktem a odhalovat možná zlepšení, aby bylo UI co nejjednodušší a zároveň intuitivní. Existují dva základní přístupy k testování použitelnosti: kvantitativní a kvalitativní.

Kvantitativní testování má za úkol ověřit určité tvrzení, například zda je nová verze UI lepší než předchozí, nebo zda je dané UI lepší než konkurence. Tento typ testování se zaměřuje na měření úspěšnosti (kolik lidí splnilo zadané úkoly) a čas strávený řešením úkolů. Během kvantitativního testování se obvykle snažíme minimalizovat interakci mezi uživatelem a návrhářem UI, aby byla výpovědní hodnota co nejvíce nezávislá.

Do-it-yourself testování patří do kategorie kvalitativního testování, které se nezaměřuje na prokazování lepší verze, ale na odhalení způsobů, jak UI vylepšit. Kvalitativní testování se zaměřuje na sledování myšlenkových procesů uživatelů místo shromažďování dat. Tento typ testování je více neformální, může zahrnovat změny testovacích scénářů během samotného testování a vyžaduje méně uživatelů.
V knize uvádí Steve Krug tři hlavní důvody, proč do-it-yourself testování funguje:

- **Všechny webové stránky mají problémy**, pravidelně se stává, že najdeme problémy s použitelností i na stránkách, které denně využívají miliony lidí.
- **Závažné problémy jsou obvykle snadno identifikovatelné**, avšak pro návrháře UI mohou být obtížnější k nalezení, protože je se systémem již dobře obeznámen. Pro uživatele, kteří se s daným UI setkávají poprvé, je nalezení závažných chyb snazší.
- **Sledování uživatelů pomáhá designérovi UI zlepšovat své schopnosti** a navrhovat intuitivnější rozhraní, jelikož získává zkušenosti s tím, jaké prvky uživatelé očekávají a kde.

Během implementace webové aplikace byly vytvářeny testovací scénáře na jednotlivé celky. Ty obsahovaly klíčové úkony, které by měl každý uživatel na stránce dokázat provést. Během testování byla hlavní pozornost soustředěna na oblasti týkající se:

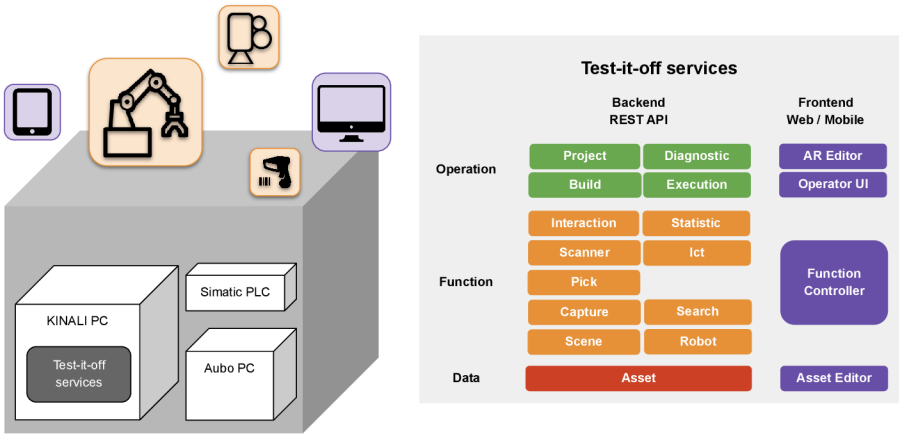
- spouštění daných projektů
- sledování statistik v různých pohledech
- řešení problémů v projektu
- vyhledávání a kontrolu logů notifikací

[1] E2E (end-to-end) testování je proces, při kterém se ověřuje, že celá webová aplikace funguje správně od začátku až do konce. E2E testy simulují reálné uživatelské interakce s aplikací, což umožňuje ověřit, že jednotlivé komponenty, služby a integrace spolu správně komunikují a aplikace splňuje očekávané chování.

Pro každou z těchto oblastí bylo vytvořeno několik testů, díky kterým se mohla ověřit přehlednost a intuitivnost UI. Před testováním byl uživatel seznámen s kontextem jeho práce, tedy "co potřebuje udělat". Nebylo mu však řečeno nic týkající se samotné implementace, pouze jeho cíle. Během testování byl kladen důraz na tzv. think-aloud metodu, kdy uživatel nahlas sděluje své myšlenky. Obrazovka s aplikací byla během testování nahrávána společně s hlasem uživatele.

3.3 Popis robotického pracoviště

Před analýzou uživatelských potřeb je nutné ujasnit si, co všechno systém test-it-off (TIO) obsahuje a které části se v rámci této práce zkoumaly.



Obrázek 3.1: **Test-it-off architektura** V levé části lze vidět příklad robotického pracoviště a v pravé části služby, které se starají o provoz.

Robotická pracoviště se skládají z různých spolupracujících hardwarových a softwarových komponent. Systém test-it-off se snaží zjednodušit složitější úlohy, jako je komunikace mezi různými protokoly, low-level programování v jazycích jako C nebo jiných proprietárních jazycích. Díky unifikaci použitím REST API je možné snadněji implementovat vyšší vrstvy systému, jako je obecné UI a jeho jednotlivé části, například Asset Editor.

Pracoviště mohou obsahovat jednoho nebo více robotů, čtečky, kamery a další zařízení. Systém je navržen tak, aby byl flexibilní a umožňoval firmám snadno měnit a upravovat konfiguraci hardwaru v závislosti na potřebách výroby.

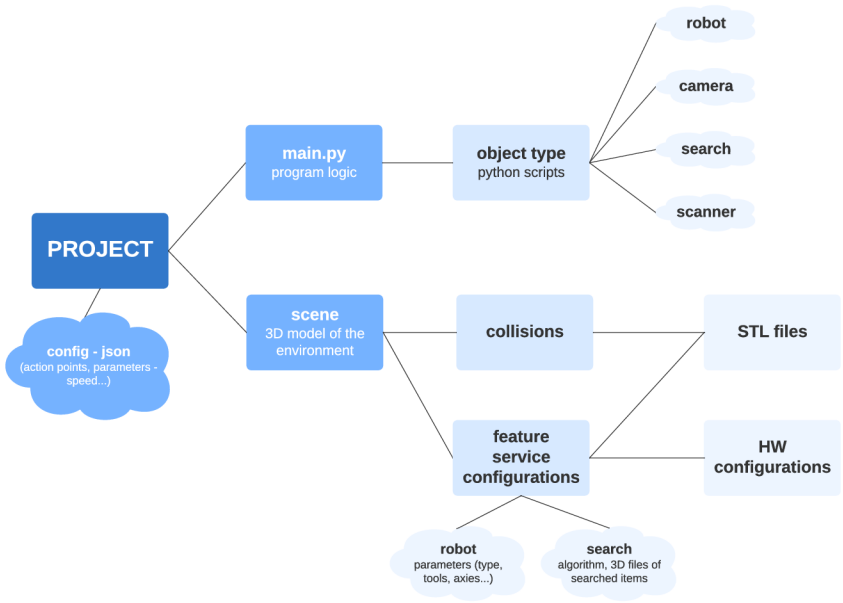
Vrchní operační vrstva systému TIO je tvořena moduly a entitami, které umožňují unifikovaným a organizovaným způsobem tvořit programy pro robotická pracoviště. Klíčový pro tuto vrstvu je koncept scén a projektů. Scéna reprezentuje fyzické rozmístění hardwaru a veškerých objektů na pracovišti. Ty jsou důležité pro definici kolizní scény, která je využívána pro plánovaní bezkolizních pohybů robota. Projekt je entita, která řídí procesy na pracovišti a koordinuje činnost různých zařízení. Více projektů může být založeno na jedné scéně. Tedy nad jednou konfiguraci HW a sestavě objektů na pracovišti můžeme definovat více projektů, které se mohou lišit různou logikou a posloupností akcí nebo jen parametry těchto akcí, jako například různá rychlost robota. Díky tomuto flexibilnímu a modulár-

nímu přístupu mohou robotická pracoviště snadno adaptovat a měnit své konfigurace podle potřeb výroby.

Jednoduchým příkladem projektu může být například sada instrukcí, kde robot nejprve přejde do určité inicializační pozice, následně hledá součástky v boxu, po nalezení dané součástky ji skenuje a vyfotí kamerou. Poté součástku vrátí zpátky na původní místo a proces se opakuje s další součástkou v pořadí.

Asset

Assetem se rozumí soubor související s činností robotického pracoviště. V zásadě se jedná o uživatelská data, která mohou být specifická pro každou aplikaci systému TIO. Typicky jsou to scény, projekty, konfigurační soubory pro služby a HW, převážně ve formátu `.json`. Dále se jako asset označují 3D modely obvykle ve formátu `.stl` reprezentující produkty hledané pomocí lokalizačního modulu, například desky plošných spojů, nebo i části pracoviště definující kolizní scénu. Každý asset také obsahuje meta informace jako například popisek, označení (tags), datum vytvoření a modifikace a v neposlední řadě seznam všech závislostí, které daný asset referencuje.



Obrázek 3.2: **Diagram assetů.** Obrázek znázorňuje příklady hlavních assetů, které může projekt obsahovat. Obdélníky znázorňují assety, jiné tvary příklady, co si pod daným assetem představit.

Akční body

Akční body slouží k definování póz (pozice a orientace v prostoru) do kterých robot najíždí. Hlavním přínosem definování akčních bodů je následná snadná úprava a přizpůsobení projektů. Pokud je potřeba vytvořit nový projekt, který je velmi podobný stávajícímu, například kvůli kontrole nové desky, může se provést klonování stávajícího projektu. Klonovaný projekt bude mít stejný kód, ale upraví se akční body, které řídí, kam se robot přesune.

Kapitola 4

Návrh aplikace

Tato kapitola se bude zabývat nejprve charakteristikou uživatelů aplikace, následně analýzou potřeb operátora (pro kterého je implementace určena) a na závěr výsledným návrhem aplikace dle předchozích znalostí.

4.1 Charakterizace uživatele

Hlavním uživatelem je **operátor**, který je zodpovědný za bezproblémový běh projektu a kontrolu procesů na stanovišti. Operátor potřebuje přijít k pracovišti, rychle nalézt požadovaný projekt a spustit ho. Po spuštění potřebuje sledovat statistiky ať už za celkový běh projektu nebo za nějakou nejnovější část, aby mohl vidět aktuální trendy ve statistikách. Statistiky si potřebuje zobrazit v různých podobách, jako například v grafech nebo průměrných hodnotách. Také potřebuje vidět log událostí - notifikací, které daný program vrací v průběhu projektu. V neposlední řadě je potřeba operátorovi poskytnout možnost řízení toku programu a řešení problémů, které můžou při běhu projektu nastat.

Uživatelem bez možnosti spouštění projektů, s povolením pouze sledovat statistiky, je **manažer**. Manažer potřebuje kontrolovat běh všech robotických stanic a jejich aktuální stavy, není však na místě u robota, takže mu není umožněno spouštět projekty.

Uživatelé s vyššími pravomocemi jsou **Advanced operátor** a **Integrátor**. Tyto role toho mají hodně společného a v rámci tohoto projektu jsou identické. Rozlišují se pouze na pozadí, kdy integrátor vytváří i skripty pro robota. Cílem těchto uživatelů je vkládat nové a upravovat stávající projekty na robotickém pracovišti. To obsahuje úpravu akčních bodů, duplikaci projektů, úpravu používaných služeb atd. Výsledkem jejich práce jsou nové dostupné projekty pro operátory. Tato funkcionalita byla pro integrátory plánovaná jako následné rozšíření uživatelského rozhraní. Z analýzy však vyplynulo, že integrátoři nepotřebují projekty spouštět. Z toho důvodu se část s editací assetů přesunula do separátního UI přímo pro ně tzv. Asset editoru popsaného v kapitole níže 4.4.

4.2 Analýza uživatelských potřeb

S hotovou výše popsanou charakterizací uživatele se lze zaměřit na přesné zkoumání uživatelských potřeb a dalších částí jako je četnost akcí nebo informační struktura.

Konkrétní uživatelské potřeby

Tato část je zaměřena na zkoumání konkrétního významu potřeby uživatele. V předchozí podkapitole byly popsány základní potřeby, avšak v následujícím odstavci budou popsány nejdůležitější aspekty těchto potřeb s využitím otázek z perspektivy operátora.

Potřeby byly zkoumány dvěma hlavními způsoby, prvním byly tzv. *user stories*. Jedná se o "příběhy" uživatelů, tedy sepsané sledy událostí nebo akce, které chce daný uživatel vykonat. Druhým bylo pozorování uživatelů a jejich zpětná vazba k původní implementaci GUI 4.4. Následně byly ještě všechny poznatky otestovány v rámci nové implementace během uživatelského testování 5.6.

- Podle čeho chci vybrat projekt?
  - hlavně dle názvu, ten slouží pro hlavní rozlišení mezi projekty
  - také však můžu na rozlišení využít popisek, ve kterém najdu víc informací o projektu
  - můžu také chtít vybrat poslední projekt který běžel, nebo jeden z posledních
- Jaké statistiky chci sledovat?
  - v první řadě potřebuji vidět aktuální hodnoty, tedy nejnovější aktualizace dané statistiky
  - v některých případech potřebuji průměrnou hodnotu
  - a pro sledování trendů statistik potřebuji grafické znázornění pomocí grafů vývoje v čase
  - na sledování krátkodobých trendů potřebuji jednoduchý způsob, jak omezit počet dat zobrazených ve statistice
- Co potřebuji neustále vidět na obrazovce při běhu projektu?
  - karty se statistikami a informace o tom, co za hodnotu právě karta ukazuje
  - název běžícího projektu a ukazatel stavu běhu
  - v případě chyby potřebuji být okamžitě upozorněn
  - také potřebuji vidět nejnovější informace o běhu projektu pomocí notifikací bez otevírání podmenu nebo vyskakovacích oken
- Také by bylo vhodné vidět některé z těchto informací napříč celou aplikací jako název běžícího projektu, notifikace, dialogy a stav projektu.

Četnost akcí

Zde byl kladen důraz na zjištění, k čemu hlavní uživatel aplikace, tedy operátor, používá aplikaci nejčastěji.

1. Nalezení, výběr a spuštění projektu
2. Sledování statistik projektu v reálném čase
3. Detekce a řešení chyb, které mohou nastat v běhu

4. Čtení logu programu - notifikací
5. Vyhledávání a řazení projektů nebo změny pohledů statistik
6. Stažení ladících informací
7. Zobrazení běžících služeb a knihoven

Struktura zobrazených informací

Projekty:

- **název projektu** - může být jakkoliv dlouhý, je tedy nutno počítat s jeho ořezáním zprava pomocí tří teček. Pro delší názvy by měl autor projektu využít popisek projektu.
- **popisek projektu** - opět může být jakkoliv dlouhý a měl by pro přehlednost podporovat nějaký styl formátování, ne pouze čistý text. Po zvážení byl vybrán markdown pro jednoduchost použití uživatelem a jeho obecnou rozšířenost.
- **možnosti řazení projektů** - projekty potřebujeme řadit dle abecedy pro jednoduché nalezení projektu a také data posledního spuštění, aby bylo jednoduché najít často spouštěné projekty. V obou případech lze řadit vzestupně i sestupně.
- **vyhledávání mezi projekty** - potřebujeme vyhledávat nejen podle názvů projektů, někteří uživatelé identifikují projekty i dle jejich popisku.

Statistiky:

- **typ hodnot** - může to být numerická hodnota s postupným průběhem nebo obrázek.
- **statistické hodnoty** - potřebné možnosti zobrazení jsou živá hodnota, průměr nebo graf. U grafů a průměru by mělo být možné vybrat ze sníženého počtu nejnovějších statistik - například posledních 100 nebo všechny dostupné.
- **grafy** - pro zobrazení grafů s velkým počtem jednotlivých hodnot slouží v rámci karty statistiky funkcionalita API tzv. downsampling. Ten řeší průměrování vícero hodnot do jedné, v případě, že se už jednotlivé body nevejdou do poměrně malého zobrazení v kartě statistiky. Pro detailnější zobrazení může uživatel graf zvětšit kliknutím na kartu statistiky a zobrazí se mu dialog na celou obrazovku s grafem.

Ostatní:

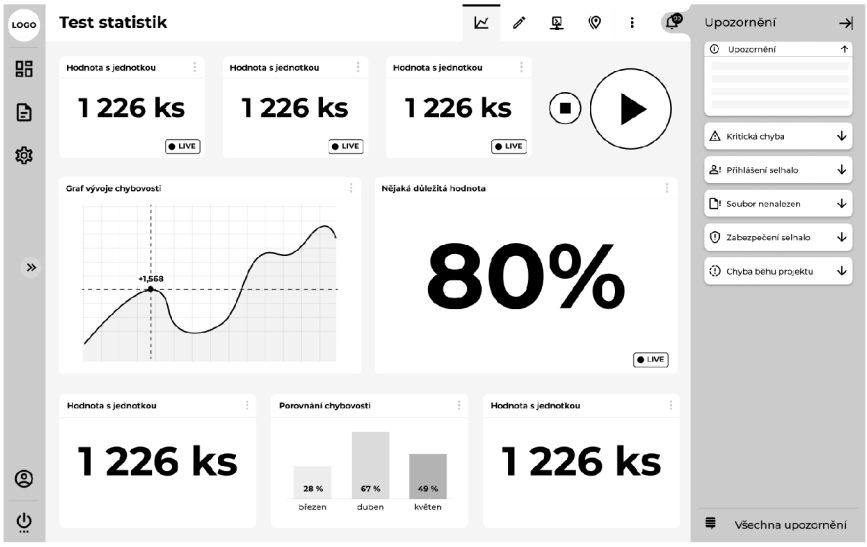
- **notifikace** - notifikace píše sám autor daného programu robota, takže obsahují pouze zprávu, důležitost a časové razítko.
- **dialogy** - řešení situace v případě více chyb v jeden moment, uživatel řeší chyby postupně vždy v novém dialogu dokud nevyřeší všechny.
- **zachování statistik a notifikací** - aktuálně tyto data mizí po spuštění jiného projektu nebo opětovného spuštění stejného.

4.3 Iterace návrhu

V průběhu vývoje aplikace bylo realizováno několik iterací návrhu, přičemž některé z nich vedly k zásadnímu přepracování konceptu. V této podkapitole bude popsaná analýza důvodů, proč určité návrhy nebyly úspěšné a jaké požadavky byly kladeny na finální řešení.

První návrh

První iterace návrhu měla spoustu problémů. Největším problémem bylo, že se jednalo o prototyp bez podkladů uživatelské analýzy, pouze na základě krátkého setkání a vysvětlení základů systému. Myšlenka, se kterou byl návrh řešen nebyla špatná, ale v závěru nesplňovala to, co se od systému očekávalo.



Obrázek 4.1: **První návrh GUI.** Barvy jsou zatím pouze v odstínech šedé, typické pro "wireframe"- skica webu.

Na obrázku 4.1 ve vrchní liště lze vidět, že je UI aktuálně na kartě statistik, ale lze se v rámci stránky dostat i na jiné karty, sloužící pro integrátory na úpravu projektů. Ikonky zleva doprava měly znamenat: statistiky, parametry, služby a akční body projektu. Tím se návrh snažil spojit funkcionalitu pro operátory i pro integrátory. To se na první pohled může zdát jako rozumné řešení, avšak po následném zkoumání požadavků se to ukázalo jako nevhodné. Integrátor by musel nejdříve hledat projekt, ke kterému jsou assety navázané, jít přes kartu statistik a následně hledat, který soubor chce upravovat. Výsledné řešení tohoto problému bude popsáno v kapitole níže 4.4.

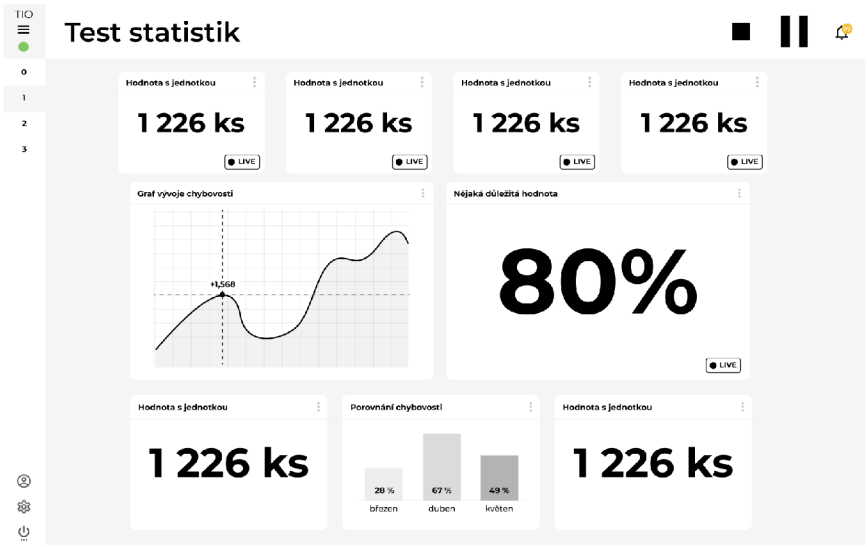
Jedním z dalších problémů identifikovaných v návrhu bylo umístění akčních tlačítek pro ovládání exekuce projektu. Ačkoli se v návrhu mohou zdát logicky umístěné mezi kartami v pravém horním rohu, předpokládalo se, že v budoucnu budou mít operátoři možnost přesouvat statistické karty podle svých preferencí. Tato flexibilita by mohla vést k nekonzistentnímu rozložení těchto klíčových tlačítek.

V neposlední řadě je nutné zmínit pozici upozornění (notifikací) v tomto návrhu. I když je pozice zprava nejčastější volbou umístění u moderních aplikací, zde bylo toto rozložení po výzkumu označeno jako nevhodné. Notifikace v rámci projektu se spíš víc podobají na log, většinou nemají nadpisy, pouze text. Ten může být dost dlouhý, tedy notifikace potřebují

zabírat hodně místa. Chyb v tomto návrhu je ještě více, za zmínku stojí ještě například tři tečky v pravém rohu karty místo jednoznačného ukazatele, co karta zobrazuje, nebo tlačítka pro spouštění projektů přímo na seznamu projektů, které jsou zbytečné při možnosti mít jenom jeden běžící projekt zároveň.

Druhý návrh

V rámci druhého návrhu došlo k revizi prvního návrhu s cílem odstranit identifikované nedostatky. Postupně se redukovali funkce, u kterých bylo určeno, že budou součástí až plánovaného asset editoru a dalších oddělených aplikací. Postupné odstraňování však mělo za důsledek "rozpadávání" původního návrhu. Ne všechny chyby byly tedy odstraněny, jako například tři tečky u karet. Začalo být zřejmé, že je nutné přepracovat návrh od začátku.

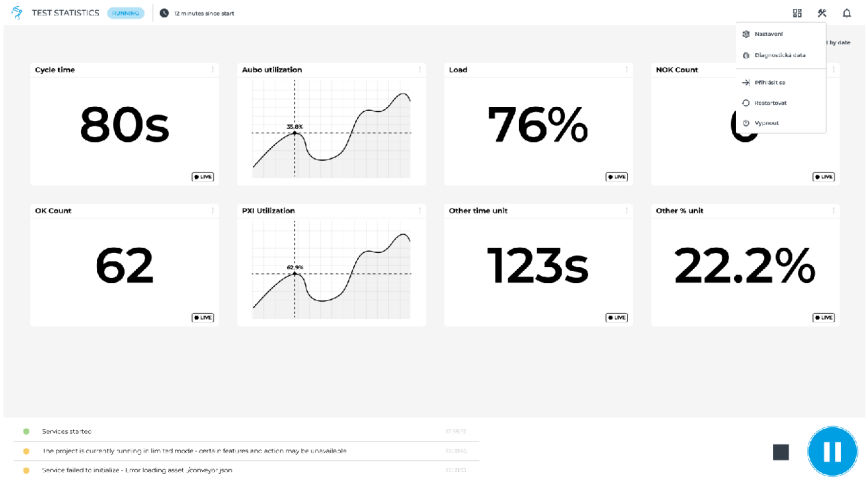


Obrázek 4.2: **Druhý návrh GUI.** Tlačítka byly přesunuty na fixní místo a název projektu byl zvětšen, aby bylo jasně poznatelné, jaký projekt běží. Notifikace se přesunuly pod tlačítko zvonečku, které by otevíralo dialogové okno.

Součástí návrhu byla myšlenka odstranění celé stránky seznamu projektů, jelikož z analýzy informační struktury 4.2 vyplynulo, že počet projektů u zákazníků by měl být v naprosté většině případů nízký, tedy by se projekty mohly přesunout například do boční lišty. Tento návrh pak ovšem narazil na problém, který vyplynul z pozdější analýzy. Někteří zákaznici využívají popisky k rozlišení jednotlivých projektů, které by se do boční lišty nevešly. Zároveň je zde pořád možnost existence zákazníka, který by rád měl velký počet projektů a UI by pro něho bylo nepřehledné.

Třetí návrh

Jedná se o první návrh po plně kompletní analýze uživatelských požadavků 4.2, tak jak tomu mělo být od začátku. U návrhu se začínalo s důrazem na vyřešení všech předchozích problémů a zachování intuitivního a dobře navrženého UX 3.1.

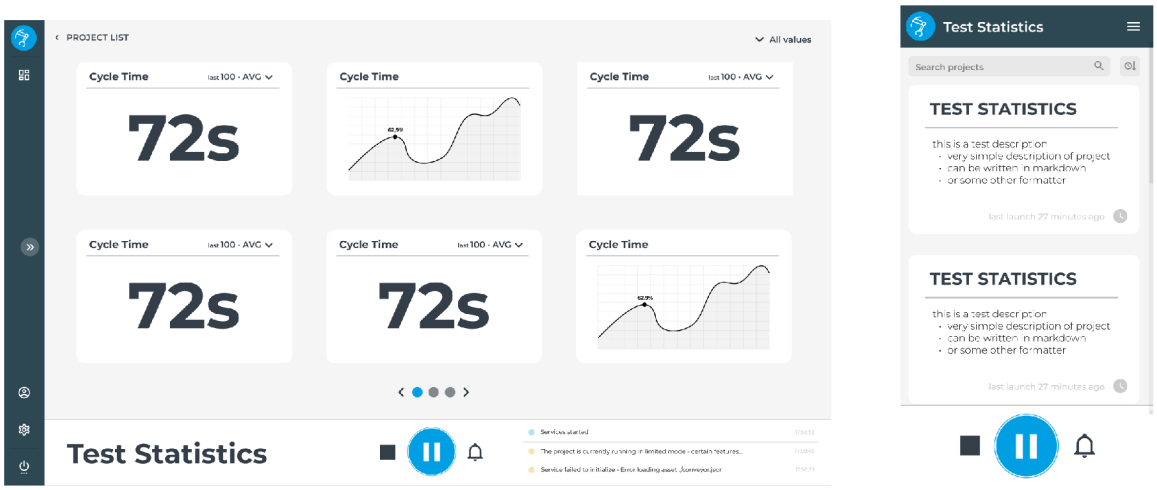


Obrázek 4.3: **Třetí návrh GUI.** Na obrázku lze vidět nový koncept spodní a vrchní lišty bez boční lišty. Na seznam projektů by bylo možné překliknout ikonkou nalevo od nastavení (v pravém horním rohu).

Tento návrh měl pořád pár problémů. Spodní lišta měla hodně volného prostoru a opět se zmenšil název projektu, který však nyní poskytoval více relevantních informací. Další nesprávný krok z pohledu UX je umístění menší tabulky s posledními notifikacemi a tlačítka na vyvolání dialogu se všemi notifikacemi na přesně opačné rohy stránky. Návrh se také plně nevěnoval kartám se statistikami, všechny tyto problémy byly však vyřešeny ve finálním návrhu.

Finální návrh

Výsledkem tří předchozích iterací a všech zkušeností sesbíraných během navrhování je čtvrtý a finální návrh UI. Boční lišta nahradila lištu vrchní a název projektu se přesunul do spodní lišty, kde může být dostatečně velký, aby nebyl problém rozeznat běžící projekt. Akční tlačítka byly přesunuty doprostřed boční lišty a v pravé části se nachází notifikace.



Obrázek 4.4: **Finální návrh GUI společně s mobilní verzí.** Podle tohoto návrhu byla řešena implementace aplikace.

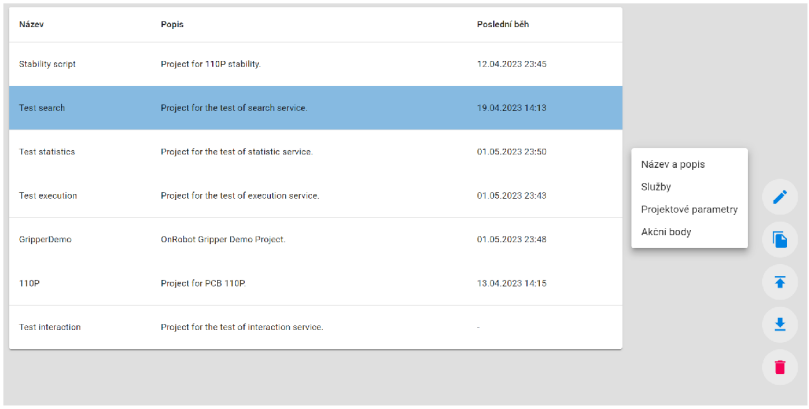
Pozornost byla také věnována i kartám projektu, které nyní přehledně sdělují, jakou hodnotu ukazují. Vrchní lištu nahradil pouze odkaz na seznam projektů v levém rohu a v pravém rohu zůstal přepínač počtu hodnot, které karty zobrazují. Součástí návrhu na dalším obrázku je i mobilní zobrazení, které však nebylo plně implementačně zpracováno. Aplikace je responzivní, ale pouze do určité míry a nedodržuje přesně navržený design.

4.4 Vytváření a úprava assetů

Tato část se zaměřuje na návrh možnosti úpravy a vytváření úloh prováděných průmyslovou stanicí. Jedná se tedy o úpravu assetů 3.3, mezi které patří například projekty, scény, konfiguračních soubory, 3D modely a další.

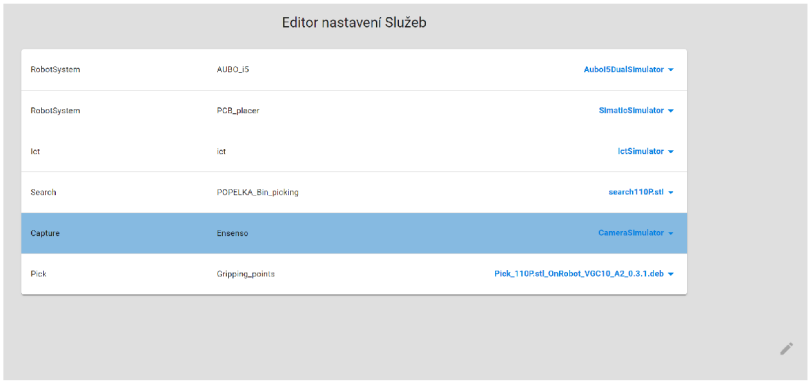
Původní způsob úpravy

V rámci původní implementace byly všechny assety navázány na projekt. Prvně si tedy každý integrátor musel najít projekt, ve kterém chce asset upravit, a následně vybrat z podmenu typů assetů z nichž nebyly ani všechny k dispozici. Nejenom, že byla původní implementace nepřehledná, ale zároveň nebyla navržena podle principu, jakým integrátoři potřebují projekty a assety upravovat. Původní způsob úpravy je vidět na obrázcích 4.5 a 4.6. Je zde vidět, že se uživatel snadno dostane i do čtyř vrstev zanoření, než se dopracuje k assetu, který chce upravit.



Obrázek 4.5: **Původní výběr projektů k editaci.** Jedná se o první stránku, kam se integrátor dostane po kliknutí na editaci.

V situaci na obrázku 4.5 je vidět, že každý projekt lze pouze zvolit. Následně musí uživatel vybrat jednu z možností v pravém dolním rohu. Možnosti nemají žádné popisky a uživatel nikdy neví, zda se akce vykoná okamžitě, nebo dostane dialog s potvrzením. Také není jasné, co každá akce provede, některé z nich otevřou dialog na existující stránce, některé způsobí zanoření do další stránky s jinou tabulkou.



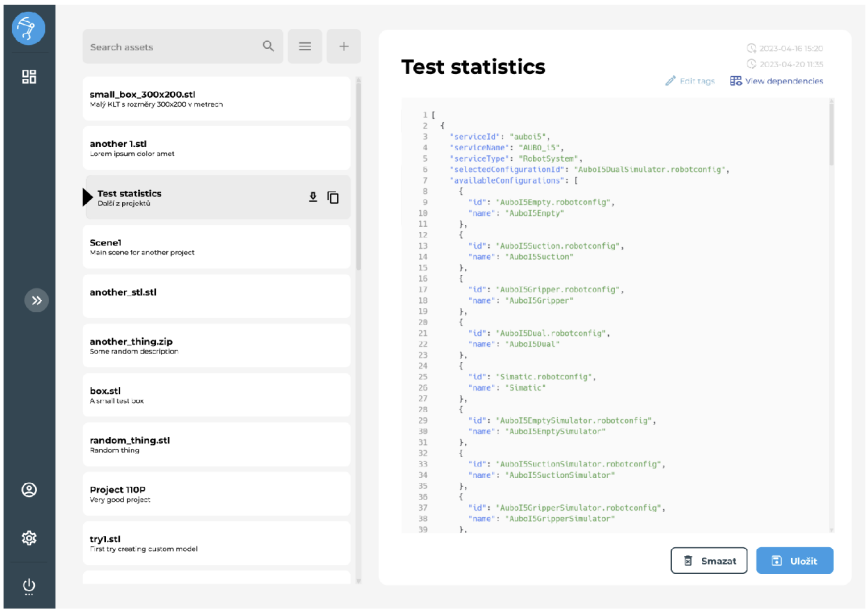
Obrázek 4.6: **Původní výběr služeb v rámci projektu.** Zde se integrátor dostane po kliknutí na úpravu Služeb v menu u projektu.

Na obrázku 4.6 lze vidět, že aktuálně zvolená služba nejde upravit (tlačítko je zašedlé). Informace o možnosti úpravy není uživateli sdělená jiným způsobem a u každé musí zjistit po zvolení v tabulce, zda jde upravit. Pokud služba upravit jde, uživatel se dostane do třetí vrstvy zanoření a otevře mu příslušný editor.

Asset editor

Asset editor je navržen jako možné rozšíření, podporující editaci a správu úloh prováděných průmyslovou stanicí test-it-off. Jedná se o separátní aplikaci sloužící pro integrátory. Byl navržen, aby integrátorům umožňoval rychlý přístup k jakémukoliv assetu a zároveň jeho rychlou editaci. Celou strukturu projektu si lze představit jako strom, kde listy jsou jednotlivé assety a projekt je kořenem stromu. Na rozdíl od původní implementace nevyžaduje asset editor od integrátora, aby postupoval tímto stromem obráceně. Integrátor vytváří a

upravuje hlavně jednotlivé assety a až na závěr může chtít vytvořit projekt, což v aktuální implementaci fungovalo přesně obráceně.



Obrázek 4.7: **Návrh asset editoru.** Na obrázku lze vidět stav editoru při vybrání assetu projektu.

Funkčnost editoru byla navržena následovně. Levá vrchní část bude umožňovat vyhledávání, filtrování a přidávání assetů. Pod touto části se nachází seznam všech assetů. Po kliknutní na asset se uživateli zvýrazní, který zvolil a bude mít k dispozici rychlé akce jako například stažení, přejmenování či vytvoření kopie daného assetu.

Pravá část obrazovky se bude dynamicky měnit dle toho, jaký typ assetu uživatel zvolil. U nejčastějších assetů, jako například u projektu nebo konfigurace scény, které jsou na pozadí `.json` objekty se na pravé straně zobrazí editor s klasickým HTML formulářem. Editor bude specificky navržen pro daný typ assetu, ale pokud by to integrátor preferoval, bude mít možnost zobrazení přepnout do "nezpracovaného" formátu, tedy textového editoru. U podporovaných 3D objektů by se na pravé straně otevřel 3D náhled. Při neznámých nebo nepodporovaných souborech lze otevřít jednoduchý textový editor s podporou zvýraznění syntaxe pro typy jako `.json` nebo `.xml`.

V pravé části může integrátor dále nalézt parametry daného assetu, jako například čas vytvoření a modifikace. Úpravu popisků a tagů může vyřešit v dialogu, stejně jako si v dialogu může zobrazit seznam assetů, které referencuje.

Další části

V kapitolách výše byly popsány řešení pro operátory a integrátory. V rámci charakterizace uživatele byl však zmiňován ještě Advanced operátor, který nevytváří assety a nové projekty. Jeho činností ale může být například duplikace jednoho projektu a změna některých parametrů, jako například akčních bodů. Této funkcionality lze samozřejmě dosáhnout i v asset editoru, avšak pro advanced operátory by byl jednodušší postup mít tzv. wizardy, které by je celým postupem provedly. Ty by fungovaly na základě oddělených kroků, tedy například prvním by mohlo být nahrání nové 3D součástky a následně druhým krokem

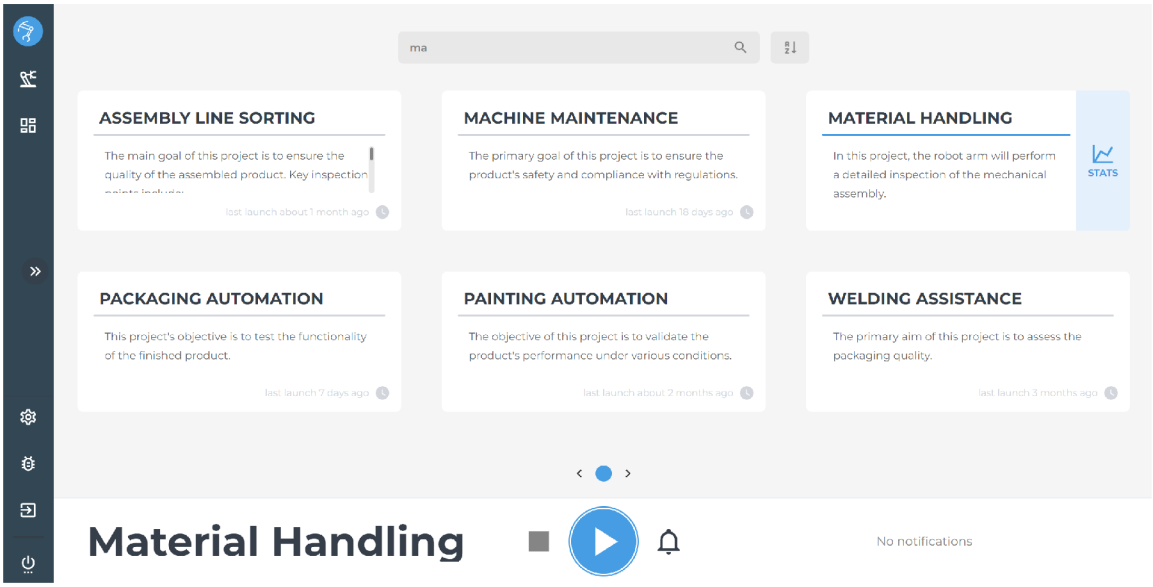
definice nových bodů, kde bude robot součástku zvedat. Toto je plánovaná funkcionalita systému test-it-off, která je však mimo rozsah této práce.

4.5 Návrh řešení

Tato část je zaměřena nejdříve na popis dvou hlavních obrazovek aplikace a následně na podrobný popis částí, které se objevují na vícero obrazovkách nebo napříč celou aplikací.

Seznam projektů

Seznam projektů je první obrazovkou, kam se uživatel dostane při úvodním spuštění. Zde mají operátoři přístup ke všem nahraným projektům, které obsahují pokyny pro robotický systém.



Obrázek 4.8: **Stránka se seznamem projektů.** Na obrázku jsou projekty seřazené od A do Z a mezi projekty se zobrazují pouze ty, které obsahují ve jménu nebo popisku "ma". Obrázek je z výsledné implementace.

Jak lze vidět na obrázku 4.8, projekty se zobrazují v mřížce karet, která se stránkuje a přizpůsobuje dle šířky a výšky obrazovky zařízení. V případě, že aktuálně neběží žádný projekt, může operátor kliknutím na kteroukoliv z karet projekt vybrat. To znamená, že se projekt nyní začne ukazovat ve spodní liště a karta projektu v mřížce se změní, aby indikovala, že je daný projekt vybrán. Plynulou animací z pravého rohu karty vyjede tlačítko s možností přepnout na detailní statistiky projektu a části karty se změní na primární modrou barvu. Naopak v případě, že projekt běží, není jakékoliv proklikávaní mezi kartami uživateli umožněno.

Posouvat mezi stránkami projektů je možné buď pomocí stránkovacích teček dole, nebo pro uživatele na dotykové obrazovce potažením prstem přes mřížku. Tedy například pro posunutí na další stránku stačí uživateli potáhnout prstem z pravé strany na levou.

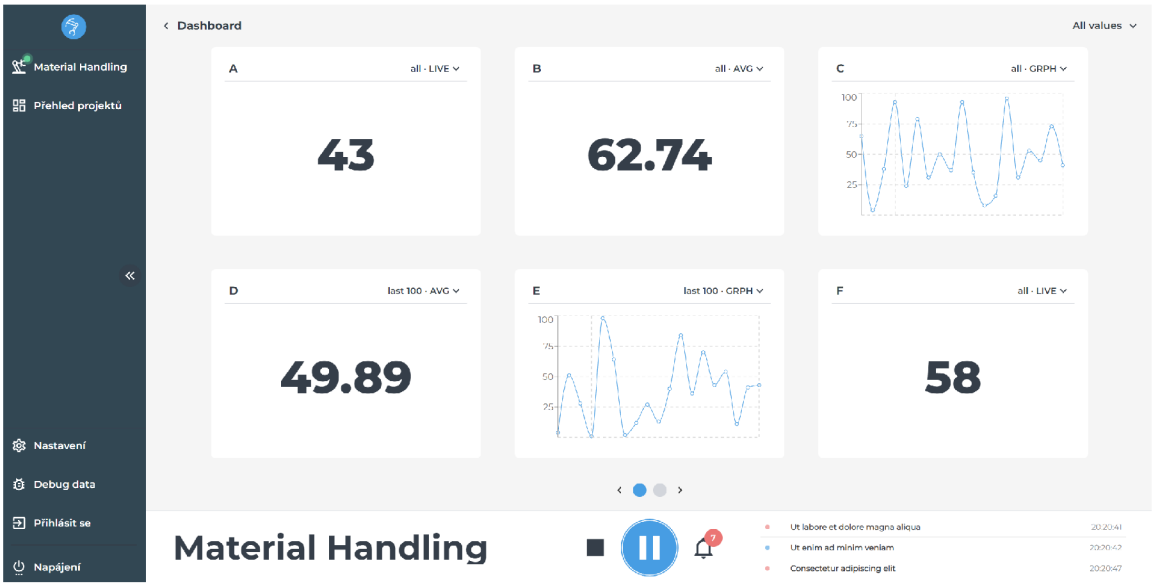
Seznam projektů je také jednou ze dvou obrazovek, na které se zobrazuje spodní lišta, což uživatelům umožňuje udržet přehled o aktuálním projektu, i když procházejí jiné pro-

jekty. V horní části obrazovky má uživatel zase k dispozici možnosti řazení projektů nebo vyhledávání podle jména nebo popisku projektu. Řadit lze dle času posledního spuštění projektu nebo abecedy, v obou případech buď sestupně nebo vzestupně. Řazení a vyhledávání je samozřejmě možné kombinovat.

Detail projektu

Na detailu projektu opět vidíme podobnou mřížku karet jako u seznamu projektů, tentokrát se však jedná o jednotlivé pojmenované statistiky projektu. Každá statistika nabízí možnost přepínání mezi různými zobrazeními, konkrétně nejnovější hodnotou, průměrnou hodnotou a grafem. Uživatelé mají také volbu mezi zobrazením posledních 100 hodnot a všech vygenerovaných hodnot, což může operátorům pomoci při detekci chyb v aktuálním trendu nebo při sledování celkového průběhu. Tato mřížka má také implementovaný stejný způsob možnosti potažení doleva nebo doprava pro změnu aktuální stránky.

Mezi posledními 100 hodnotami a všemi hodnotami lze přepínat buď globálně z pravého horního rohu, nebo individuálně pro jednotlivé karty (viz. obrázek 4.9). Uživatel má možnost prohlížet obrázek či graf v detailnějším zobrazení, a to kliknutím na obsah karty, tedy v oblasti pod nadpisem. Tímto způsobem se otevře dialogové okno s rozšířeným zobrazením daného grafu nebo obrázku přes celou obrazovku. V rámci potenciálního budoucího rozšíření je plánováno zavést možnost přeskládání statistik v libovolném pořadí. V současné implementaci se statistiky zobrazují v pořadí, jak byly přijaty z API.



Obrázek 4.9: **Stránka s pohledem na statistiky.** Lze vidět průběh statistických hodnot u běžícího projektu. Obrázek je z výsledné implementace.

Detail projektu opět zahrnuje spodní lištu, stejně jako v předchozím seznamu projektů. Rovněž je zde implementován stejný způsob stránkování s využitím teček, jako v seznamu projektů. Tečky signalizují počet stránek, přičemž aktuální stránka je reprezentována modrou tečkou. Během analýzy uživatelských požadavků 4.2 bylo zjištěno, že počet stránek s projekty či statistikami nebude natolik vysoký, aby tento systém byl nepřehledný. V případě, že by v budoucnu bylo nutné implementovat rozšíření pro automatické zkracování

prostředních a krajních stránek, například s využitím tří teček, lze takovou funkci snadno přidat do stávajícího systému.

Spodní lišta

Spodní lišta (vidět ve spodní části na obrázcích 4.8 a 4.9) zajišťuje snadný přehled aktuálně spuštěného projektu na různých stránkách aplikace, kde je tato informace relevantní. V levé části lišty je zobrazen název aktuálně vybraného či spuštěného projektu, který umožňuje uživatelům přepínat na detailní zobrazení projektu prostřednictvím kliknutí na nadpis. Střední část panelu obsahuje akční tlačítka, konkrétně tlačítka pro zastavení, spuštění či pozastavení (závislé na stavu projektu) a tlačítko pro zobrazení notifikací.

Tlačítko notifikací vyvolávající dialog 4.5 také vždy ukazuje, kolik nových notifikací přišlo od doby, co byl dialog naposledy otevřen a zároveň se tento ukazatel zabarvuje podle nejzávažnější úrovně notifikace, která nebyla přečtená. Tedy ukazatel je modrý, pokud přišly pouze informativní notifikace, oranžový pokud je mezi nimi i varování a červený pokud se mezi nimi vyskytuje chyba. Pokud byly všechny notifikace přečteny, ukazatel zmizí.

V pravém rohu se nachází tabulka posledních tří notifikací, seřazených od nejnovější nahoře, spolu s jejich úrovní (Informace, Varování, Chyba) a časovým razítkem. Notifikace jsou vždy vázané k jednomu běhu projektu, zobrazují se tedy jenom ve chvíli, kdy má uživatel vybraný správný projekt, jinak se v této části objeví text "No notifications". Pokud nemá uživatel vybrán žádný projekt, všechny části spodní lišty jsou nahrazeny textem "No project selected".

Boční lišta

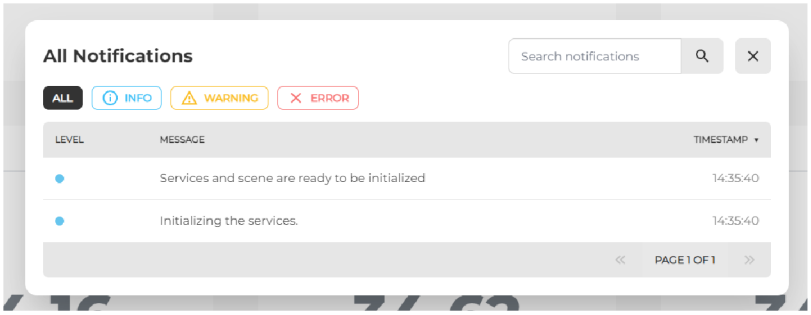
Boční lišta slouží v projektu na jednoduché přepínání mezi různými pohledy a na rychlý přístup k funkcím, které může operátor potřebovat. Ve své výchozí zmenšené podobě (viz. obrázek 4.8) jsou vidět pouze ikonky jednotlivých akcí nebo funkcí. V prostředku se nachází tlačítko na rozbalení boční lišty do její plné šířky. V plné šířce lze vidět popisky jednotlivých ikon, které nejsou ve zmenšené verzi vidět (viz. obrázek 4.9).

Ve vrchní části pod logem systému test-it-off se nachází odkazy na dva hlavní funkční celky, přehled projektů a detail zvoleného, resp. běžícího projektu. Pokud nebyl žádný projekt zvolen a nejsou dostupné statistiky z žádného předchozího běhu, je tlačítko na detail projektu zakázané. Naopak v případě aktivního projektu se u tlačítka zobrazuje pulzující indikátor jeho stavu, aby uživatel mohl rychle vidět aktuální stav napříč celou aplikací. Indikátor se také zabarvuje podle závažnosti stavu, tedy je oranžový, je-li projekt pozastaven a červený, jestli je projekt zastaven z důvodu chyby.

Ve spodní části se pak nachází více systémových funkcí, jako například odkaz na nastavení nebo stažení diagnostických dat. Také se zde nachází možnost přihlášení pro administrátory a nastavení napájení, které umožňuje operátorům celý systém restartovat nebo vypnout.

Dialog notifikací

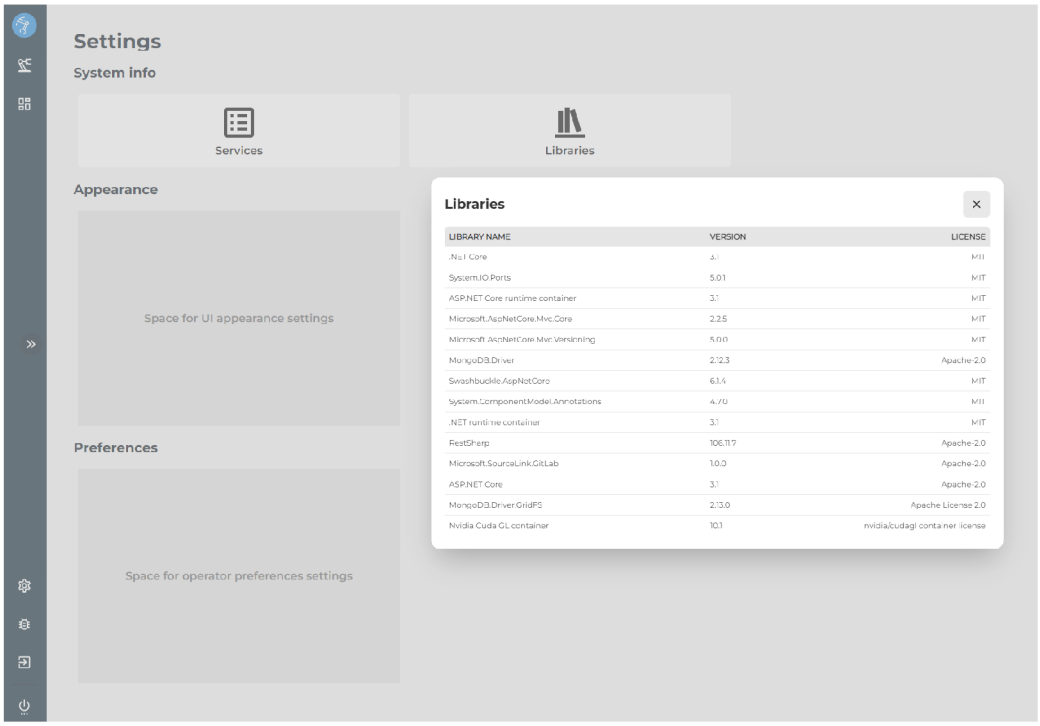
Po kliknutí na ikonu zvonečku ve spodní liště 4.5 dostává operátor možnost prohlížet seznam všech notifikací spojených s průběhem projektu. Notifikace lze filtrovat podle úrovně závažnosti či provádět vyhledávání mezi nimi. Navíc mohou uživatelé tabulku seřadit dle libovolného sloupce prostřednictvím kliknutí na jeho hlavičku. Tabulka podporuje stránkování, aby nebyl při velkém počtu notifikací dialog příliš vysoký.



Obrázek 4.10: **Notifikační okno.** Jestli nejsou žádné notifikace dostupné, dialog se po kliknutí na tlačítko zvonečku neotevře.

Nastavení

Stránka nastavení v rámci této práce obsahuje pouze seznamy služeb a knihoven, které systém TIO využívá. Tyto seznamy se nachází v separátních vyskakovacích oknech, které se zobrazují po kliknutí na tlačítka v sekci "System info". Nastavení se budou postupně doplňovat při dalším vývoji systému o funkce, které budou chtít uživatelé pravidelně měnit a nastavovat.



Obrázek 4.11: **Stránka nastavení.** Lze vidět vyhrazené místo pro budoucí nastavení v kategoriích "Apperance" a "Preferences".

Kapitola 5

Implementace

Aplikace je naprogramována v jazyce TypeScript a o rozložení stránky se stará HTML a CSS. Struktura celého projektu je koncipovaná podle "features" - tedy funkčních celků programu, podle kterých je rozdělená. Tedy například můžeme považovat každou stránku za samostatnou funkcionalitu, ale také některé komponenty napříč stránkami, jako například boční nebo spodní lištu.

5.1 Funkční celky

Finálními funkčními celky jsou: `Dashboard`, `ProjectDetail`, `Sidebar`, `Bottombar` a `Settings`. Každý funkční celek má vlastní složku s následující strukturou:

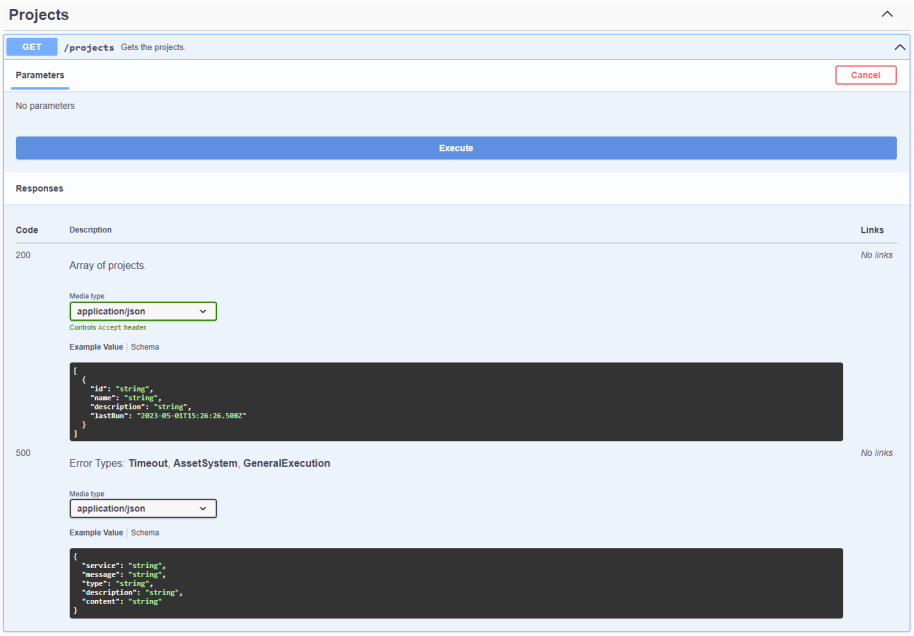
- **components/** - složka s komponenty použitými pouze v tomto celku, může se jednat o specifickou tabulku, formulář nebo jiný komponent.
- **models/** - obsahuje modely (typy) v jazyce TypeScript související s daty v rámci tohoto celku
- **ViewName.tsx** - hlavní komponent, který "zabaluje" všechny komponenty do jedné stránky a obsahuje logiku pro jejich případné podmíněné vykreslení.
- **viewService.ts** - hlavní služba, vyřizující dotazy na API a jiné funkce jako například vyhledávání a řazení.

Všechny zdrojové soubory jsou ve složce `/gui/src/`, kde `main.ts` je vstupním souborem programu. V něm se inicializuje samotný React a další jeho rozšíření použité v projektu. Jedním z nich je react-router, který se stará o přecházení mezi jednotlivými stránkami - funkčními celky. Dalším je například Recoil, který udržuje globální data v tzv. stores[1].

5.2 API

Již existující API popsaná v nástroji Swagger 2.2 je rozdělená do celků. Celky které jsou potřeba k implementaci jsou Diagnostic, Interaction, Production, Projects a Tokens.

[1]Store je ústřední úložiště, které udržuje stav aplikace a zajišťuje jeho správné aktualizace. Funguje jako jednotný zdroj pravdy pro stav aplikace, což umožňuje snadnější sledování a manipulaci s daty. Tímto způsobem store zjednodušuje komunikaci mezi komponentami a zlepšuje řízení stavu v reaktivních aplikacích.



Obrázek 5.1: **Náhled na koncový bod `/projects` v nástroji Swagger.** Zde lze také otestovat jaké data aktuálně API vrací pomocí tlačítka "Execute".

Modely pro data, které vrací API byly vygenerovány pomocí OpenAPI schémy z nástroje Swagger. Základem celé aplikace jsou projekty, ty se zobrazují na kartách v celku dashboard. Dotaz na api probíhá na koncový bod `/projects` a každý projekt má následující strukturu:

- **id** - jedinečný identifikátor projektu, většinou je odvozen z názvu projektu, ze kterého jsou odstraněny nevalidní znaky, tedy např. mezery, speciální znaky atd.
- **name** - název projektu.
- **description** - popis projektu, může být buď v čistém textu, nebo ve formátu markdown.
- **lastRun** - časové razítko posledního běhu projektu, pokud projekt nikdy neběžel, vrací null.

Pokud je potřeba zobrazit statistiky projektu, lze se dotázat na `/production/{projectId}/statistics` kdy se na dotaz vrátí:

- **name** - název dané statistiky, tedy například: "Počet NOK" nebo "Počet cyklů".
- **average** - průměrná hodnota statistiky ze všech dostupných datových bodů
- **values**[] - pole obsahující sbírané hodnoty společně s časovým razítkem sběru statistiky. Tedy v poli jsou objekty s klíči `value` (hodnota) a `created` (časové razítko vytvoření)

Na stejném místě jako statistiky je také potřeba zobrazit obrázky, ty se však ukládají jiným způsobem a lze se na ně dotázat z jiného koncového bodu `/production/{projectId}/images` který vrací:

- **name** - název obrázku, tedy informace o tom, k čemu se obrázek vztahuje.
- **data** - binární data obrázku kódované ve formátu base64[2].

Ve finále se v rámci implementace kód dotazuje na oba koncové body zároveň a tyto typy se díky společné hodnotě `name` spojí dohromady a následně stačí pouze rozlišit, zda má daná statistika hodnotu `data` - tedy jestli se jedná o obrázek nebo nikoliv.

Dalším typem jsou notifikace, které se uživateli zobrazují v průběhu projektu a lze se na ně dotázat na `/interaction/notifications`.

- **message** - tělo notifikace, může být pouze v čistém textu, žádné pokročilé formáty nejsou povoleny.
- **level** - úroveň notifikace, může nabývat hodnot Info, Warn a Error.
- **created** - časové razítko vytvoření notifikace

Velice důležité jsou také vyskakovací okna - dialogy, dostupné na `/interaction/dialogs`, pomocí kterých uživatel řeší problémy na robotickém pracovišti.

- **id** - jedinečný identifikátor dialogu, dle kterého se poté posílá požadavek zpět na API se zvolenou možností řešení.
- **title** - nadpis dialogu.
- **content** - obsah dialogu, opět může být pouze čistý text.
- **options**[] - pole možností dialogu, jedná se o pole řetězců, které se použijí jako názvy tlačítek s možnostmi řešení.
- **isActive** - pravdivostní hodnota, pokud je nepravdivá, tak i když API vrací dialog, nemusí ho operátor aktivně řešit

Pro vyřešení dialogu se odesílá POST požadavek na koncový bod `/dialogs/resolve` se dvěma parametry v těle. Prvním je řetězec reprezentující zvolenou možnost řešení a druhým je id dialogu. Na základě těchto parametrů si API dokáže přiřadit akci k danému dialogu a vykonat ji.

Posledním důležitým koncovým bodem je `/production/state` ze kterého lze zjistit aktuální stav projektu.

- **actionPointIds** - identifikátory akčních bodů projektu, které ale nejsou podstatné pro tuto implementaci.
- **activePackageId** - identifikátor aktivního projektu.
- **exceptionMessage** - pokud nastala v běhu projektu nějaká chyba, je vypsána zde, jinak je položka null.
- **state** - řetězec obsahující jednu z šesti hodnot dle aktuálního stavu projektu, jedná se o hodnoty Undefined, Running, Completed, Faulted, Paused a Pending.

[2]Base64 je metoda kódování binárních dat do textového formátu, která využívá 64 různých znaků (číslice, písmena a další speciální znaky) pro reprezentaci dat. Tato metoda je často používána pro přenos binárních souborů, jako jsou obrázky či soubory, v textových protokolech jako je email nebo HTTP.

5.3 Komunikace s API

V projektu bylo potřeba hodně koncových bodů "odebírat", tedy okamžitě reagovat na změny v datovém modelu aplikační logiky. Za tím účelem byla vytvořena funkce `subscribe()`, která umožňuje kdekoliv v kódu odebírat živé hodnoty z API v reálném čase. Také zabezpečuje na pozadí všechny nutné akce a řeší všechny možné chyby, které mohou během odebírání nastat. Následně stačí funkci importovat do jakékoliv komponenty a zavolat ji s chtěným koncovým bodem.

```
1  import { OpArgType, OpReturnType } from 'openapi-typescript-fetch'
2
3  import { paths } from './@types'
4  import fetcher from './fetcher'
5
6  export default async function subscribe<
7      K extends keyof paths,
8      V extends keyof paths[K],
9      A extends OpArgType<paths[K][V]>,
10     D extends OpReturnType<paths[K][V]>
11 >(
12     url: K,
13     method: V,
14     args: A,
15     callback: (data: D) => void,
16     signal: AbortSignal,
17     firstCall = true
18 ): Promise<void> {
19     if (signal.aborted) return
20
21     try {
22         if (typeof args !== 'object' || args === null) return
23
24         const subscribeUrl = fetcher(signal).path(url).method(method).create()
25         const { status, data: response } = await subscribeUrl(
26             firstCall ? { ...args, longPollingMilisecondsTimeout: 0 } : args
27         )
28
29         if (status == 502) {
30             await subscribe(url, method, args, callback, signal, false)
31         } else if (status != 200) {
32             await new Promise((resolve) => setTimeout(resolve, 1000))
33             await subscribe(url, method, args, callback, signal, false)
34         } else {
35             callback(response as D)
36             await subscribe(url, method, args, callback, signal, false)
37         }
38     } catch (error) {
39         if (signal.aborted) return
40
41         await new Promise((resolve) => setTimeout(resolve, 1000))
42         await subscribe(url, method, args, callback, signal, false)
43     }
44 }
```

Obrázek 5.2: **Abstrahovaná funkce `subscribe()`.**

Na prvních pěti řádcích 5.2 můžeme vidět import nutných souborů. Zde bych podotkl typy `OpArgType` a `OpReturnType`, díky kterým lze pak na řádcích 6-10 definovat přesné parametry argumentů funkce a návratového typu funkce. Kdyby se zde uvedly pouze typy `string` nebo `array`, TypeScript by nedokázal odvodit, o jaké typy se jedná. Díky tomu dokáže editor kódu programátorovi napovědět, jaké argumenty daný koncový bod přijímá nebo jaké datové typy vrací.

Z kódu lze následně vidět, že se v rámci funkce také řeší jakékoliv problémy a funkce umožňuje reagovat na přerušení - `abort()`. To je zejména potřebné při odchodu z dané stránky, všechny odběry na koncové body se totiž musí zrušit. Na řádcích 24 a 25 lze vidět

samotné odesílání požadavku na koncový bod. Na řádku 24 se požadavek vytvoří a na řádku 25 se zavolá. Pokud se jedná o první volání této funkce, nastaví se `longPollingMilisecondsTimeout` na hodnotu 0, aby se požadavek vrátil okamžitě. Následně už se požadavek odesílá s parametry, které uživatel zadal manuálně.

```
1 useEffect(() => {
2     const controller = new AbortController()
3     subscribe(
4       '/interaction/dialogs',
5       'get',
6       { longPollingMilisecondsTimeout: 60000 },
7       (dialog) => {...}
8       controller.signal
9     )
10    return () => controller.abort()
11 }, [dialog])
```

Obrázek 5.3: **Volání funkce `subscribe()`.**

Na obrázku 5.3 lze vidět názorné volání funkce `subscribe()` pro odběr koncového bodu dialogů s intervalem 60 sekund. Funkce je zabalená do hooku `useEffect()` 5.4. Zavolá se tedy při vykreslení komponenty a její návratová funkce se vykoná při opětovném zavření komponenty a odběr se přeruší. Místo tří teček na řádku 7 lze následně pracovat s návratovou hodnotou. Kód místo tří teček by se volal pokaždé, co se z API vrátí nová data.

5.4 Implementace v knihovně React

Tato část rozebírá nejdůležitější aspekty a techniky použité při implementaci výsledné aplikace. Jsou zde zmíněny základní části jako *hooks* a *routing*, ale i specifické implementace.

Management stavu

Ve frameworku React lze efektivně spravovat stav aplikace pomocí tzv. hooks. Jedná se o zásadní koncept Reactu, který byl představen ve verzi 16.8. Hooky jsou funkce, které se "připojují" k interním mechanismům Reactu a poskytují přístup k funkcím, jako je správa stavu nebo životní cyklus komponent. Existují vestavěné hooky, jako jsou `useState()`, `useEffect()`, `useMemo()` nebo `useContext()`, ale lze si také vytvořit vlastní hooky pro specifické potřeby aplikace [13].

Funkce `useState()` je základem pro lokální správu stavu v komponentech. Vrací dvojici hodnot, aktuální stav a funkci pro aktualizaci tohoto stavu. Při vytváření nového stavu pomocí `useState()` se jako parametr předává počáteční hodnota. Tato metoda je vhodná pro komponenty s lokálním stavem, který není potřeba sdílet s ostatními částmi aplikace.

`useEffect()` je další hook, který umožňuje provádět činnosti v reakci na změny závislostí, jako je načítání dat, aktualizace DOM atd. `useEffect()` přijímá dva parametry: funkci, která obsahuje logiku "efektu", a pole závislostí, které určuje, kdy má být efekt spuštěn. Pokud je pole závislostí prázdné, `useEffect()` se spustí pouze při prvním vykreslení komponenty.

Dalších užitečný hook je `useMemo()`, který umožňuje optimalizaci výkonu aplikace tím, že ukládá výsledek volání funkce a při dotazu vrátí uloženou hodnotu, pokud se závislosti

nezměnily. `useMemo()` přijímá dva parametry: funkci, která vrací hodnotu, a pole závislostí, které určuje, kdy má být hodnota znovu vypočítána.

Pro management globálního stavu aplikace je v projektu využitá knihovna Recoil.js. Ta obsahuje funkce jako `useRecoilState()` a `useRecoilValue()`. `useRecoilState()` je hook, který umožňuje číst a aktualizovat atomický stav. Tento hook vrací stejnou dvojici hodnot jako `useState()`, ale s tím rozdílem, že stav může být sdílen mezi více komponenty. Funkce `useRecoilValue()` je další hook, který umožňuje číst atomický stav nebo vypočítaný stav (tzv. selector) bez nutnosti aktualizace. Tato funkce je užitečná v situacích, kdy komponenta potřebuje pouze číst stav, ale neaktualizovat jej. Pomocí těchto funkcí můžeme vytvořit komplexní a propojený stav aplikace, který lze efektivně sdílet mezi komponentami. Ukázka všech výše zmíněných hooků je na obrázku 5.4.

```
1 import { executionStateAtom, isExecutionActiveSelector } from './stores/rootStore'
2
3 const [executionState, setExecutionState] = useRecoilState(executionStateAtom)
4 const executionActive = useRecoilValue(isExecutionActiveSelector)
5 const [statistics, setStatistics] = useState([] as Statistic[])
6 const statsCombined = useMemo(() => {
7     return [...statistics, ...images]
8 }, [statistics, images])
```

Obrázek 5.4: **Ukázka několik hooků v Reactu.**

Přecházení mezi stránkami

Pro ovládání navigace je v aplikaci použitý nástroj react-router. Konfigurace navigace je definována v souboru `src/router.tsx`. Ta obsahuje pole objektů, které reprezentují různé cesty a odpovídající komponenty pro zobrazení.

Hlavní cesta aplikace "/" zobrazuje komponentu `<App />`. Pokud by došlo k navigační chybě, je zobrazena komponenta `<ErrorPage />`. Následně jsou definované "children" neboli potomky, což jsou další cesty, které jsou zanořeny pod hlavní cestou. Ty se tedy vykreslují zároveň s obsahem komponenty `<App />`.

Mezi potomky patří například cesta `/dashboard-view`, která zobrazuje komponentu <DashboardView />, cesta `/project-detail/:projectId`, která zobrazuje komponentu `<ProjectDetail />` a přijímá proměnnou `projectId` jako parametr v cestě, a nakonec cesta `/settings`, která zobrazuje komponentu `<SettingsView />`.

Dialogy

Dialogy jsou implementovány pomocí kombinace `useState()` a propojení komponent. V komponentě, ve které je dialog importován, se použije `useState()` pro správu stavu otevření dialogu. Stav obsahuje informaci o tom, zda je dialog otevřený, a funkci pro otevření či zavření dialogu.

Před otevřením dialogu si komponenta načte potřebná data, která budou v dialogu zobrazena. Tato data mohou být získána z API nebo z globálního kontextu aplikace. Když se má dialog otevřít, načtená data spolu s pravdivostní hodnotou otevření dialogu a funkcí na jeho otevření jsou předány do komponenty dialogu pomocí parametrů komponenty.

V rámci komponenty dialogu je pravdivostní hodnota otevření použita pro ovládání viditelnosti dialogu pomocí CSS třídy DaisyUI `modal-open`. Funkce na otevření či zavření

dialogu je následně přiřazena tlačítku na zavření dialogu v pravém horním rohu. Případně se však může volat po vykonání určité akce v dialogu, aby se dialog následně automaticky zavřel.

Funkce pro dotykové zařízení

Funkcionalita dotyků a tahů je implementována pomocí funkcí knihovny React `onPointerDown`, `onPointerUp`, `onTouchStart` a `onTouchEnd`. Implementace ověřuje, zda uživatel přejel na bodech dotyku a opětovného puštění dostatečnou vzdálenost po ose x a do jakého směru a dle toho posune stránku.

```
1  const dragStartPoint = useRef({ x: 0, y: 0 })
2
3  function handleDragStart(clientX: number, clientY: number) {
4      dragStartPoint.current = { x: clientX, y: clientY }
5  }
6
7  function handleDragEnd(clientX: number, clientY: number) {
8      const dragEndPoint = { x: clientX, y: clientY }
9      const dragDistance = {
10         x: dragEndPoint.x - dragStartPoint.current.x,
11         y: dragEndPoint.y - dragStartPoint.current.y,
12     }
13
14     if (dragDistance.x > 100) setPage(page - 1)
15     else if (dragDistance.x < -100) setPage(page + 1)
16 }
```

Obrázek 5.5: **Kód reagující na potažení.** Následně se tyto funkce volají u potažení myši nebo dotyku na obrazovce.

Při pohybu myší se funkce na obrázku 5.5 volá s parametry `handleDragStart(e.clientX, e.clientY)`. Avšak při dotyku na obrazovce je potřeba vzít v potaz větší množství dotykových bodů ve stejný okamžik. Uživatel se totiž může dotýkat obrazovky vícero prsty najednou, v rámci projektu určuje první bod doteku jestli se má mřížka posunout na další stránku nebo ne. Implementace tak vypadá následovně `handleDragStart(e.touches[0].clientX, e.touches[0].clientY)`.

Responzivní mřížky

V rámci implementace bylo potřebné vytvořit dvě, téměř identické implementace reaktivních mřížek. Jednalo se buď o seznam projektů nebo statistik. Mřížky se musí správně zvětšovat a zmenšovat, tedy měnit počet sloupců a řádků karet na jedné stránce, dle výšky a šířky stránky. Toho je dosaženo pomocí kódu na obrázku 5.6 níže.

```
// Update all relevant project data when parameters change
useEffect(() => {
    const filteredProjects = filterProjects(projects, searchText)
    const sortedProjects = sortProjects(filteredProjects, sorting)

    const start = (page - 1) * maxPerPage
    const end = start + maxPerPage
    setProjectsPage(sortedProjects.slice(start, end))

    const calcMax = Math.ceil(sortedProjects.length / maxPerPage)
    setMaxPages(calcMax > 0 ? calcMax : 1)

    // If the page is suddenly (resize, filter...) out of bounds, set it to the last page
    if (page > maxPages) setPage(maxPages)
}, [projects, maxPerPage, page, searchText, sorting, maxPages])

// Update maxPerPage when window size changes
useEffect(() => {
    if (!width) return
    const calcMax = Math.floor((height - 400) / 200)
    const maxCardRows = calcMax > 0 ? (calcMax > 3 ? 3 : calcMax) : 1

    if (width < 1280) setMaxPerPage(1 * maxCardRows)
    else if (width < 1700) setMaxPerPage(2 * maxCardRows)
    else setMaxPerPage(3 * maxCardRows)
}, [width, height])
```

Obrázek 5.6: **Kód zodpovídající za mřížku.** Jedná se o mřížku pro seznam projektů, statistiky nenabízí možnost filtrace a vyhledávání.

Spodní funkce `useEffect()` nastavuje maximální počet řádků a sloupců dle výšky a šířky stránky. Následně vrchní funkce reaguje na změny proměnné `maxPerPage` a jiných závislostí jako filtrace a řazení aby vykreslila správné projekty v mřížce.

5.5 Funkční testování

Použití nástroje Cypress 3.2 pomohlo odhalit neočekávané chyby aplikace ještě před nasazením. Pomocí nástroje cypress byly psány E2E testy pro každou z jednotlivých *features*.

```
// Import Cypress commands
import { visit, get, type, click, contains } from 'cypress'

describe('Login Form', () => {
    it('Allows a user to login successfully', () => {
        // Navigate to the main page and click on the login button in the sidebar
        visit('/dashboard-view')
        get('#sidebar-login').click()

        // Type in the username and password
        get('#username').type('admin')
        get('#password').type('admin')

        // Click on the login button and check if the site has updated appropriately
        get('#login-button').click()
        get('#sidebar').contains('Odhlásit se')
    })
})
```

Obrázek 5.7: **Příklad testu na ověření funkcionality přihlášení.** Jednotlivé akce jsou popsány v kódu.

5.6 Uživatelské testování

Metody uživatelského testování popsané v kapitole 3.2 odhalily několik problémů. Mezi drobné chyby, které jsou již nyní ve finální implementaci opraveny patří například ovládání pomocí klávesnice. Funkcionalita na kterou vývojář jednoduše zapomene během implementace, jako například potvrzení dialogu pomocí klávesy Enter nebo zrušení dialogu pomocí Esc. Další z takových chyb byla nevhodná ikonka pro aktuálně spuštěný projekt, která byla příliš podobná ikoně nastavení.

Ze záznamů obrazovky bylo zjištěno, že uživatelé často klikají na logo aplikace v levém horním rohu, když jsou zmatení. Očekávali, že se kliknutím vrátí zpět na domovskou obrazovku. Tlačítko však v původním stavu nikam nevedlo.

Závažnější chyba, která byla následně opravena, spočívala v nedostatečném informování uživatelů o provedených akcích v UI. Do finální verze byly tedy přidány notifikace o akcích v pravém horním rohu. Uživatel je upozorňován na akce jako úspěšné spuštění a pozastavení projektu nebo úspěšné přihlášení. Notifikace samozřejmě upozorňují i na neúspěšné akce v rámci UI.

Důležitým zjištěním bylo rovněž, že někteří uživatelé měli problémy s orientací v aplikaci, neboť nebylo zřejmé, že projekty v seznamu se automaticky přidávají do spodní lišty, což umožňuje okamžité spuštění projektu. Některým uživatelům také nebylo jasné, že daný projekt běží na základě stylu karty projektu. Naopak bych chtěl pozdvihnout animaci karty při přechodu do zvoleného stavu. Každý uživatel reagoval na animaci správně a pochopil, že lze nyní přejít na detail projektu se statistikami.

Kapitola 6

Závěr

Hlavním cílem této práce bylo seznámit se s UX postupy a průmyslovým systémem test-it-off a následně provést analýzu požadavků na uživatelské rozhraní tohoto systému, zejména z hlediska operátora. Poté bylo nutné navrhnout grafické rozhraní (UI) s vhodným uspořádáním pro zobrazení provozních informací a výkonnostních statistik. Posledním krokem byla implementace navrženého UI a jeho integrace s existující mikroservisní architekturou systému test-it-off.

Vzhled finální aplikace vznikl po několika iteracích návrhu, které byly spojeny s výzkumem uživatelů a jejich potřeb. Všechny návrhy byly zpracovány v nástroji Figma, který nabízí intuitivní rozhraní a umožňuje snadné znovupoužití již navržených komponent. Součástí návrhu, avšak bez implementace, byla i samostatná část pro integrátory nazvaná *Asset editor* 4.4, která zajišťuje jednoduchý a přehledný způsob úpravy souborů týkajících se robotického pracoviště - assetů.

Výsledná aplikace je implementována v jazyce TypeScript, s využitím moderního frameworku React. Pro rozvržení stránky byl použit jazyk HTML a CSS. HTML kód byl generován prostřednictvím jazyka TSX a CSS kód byl napsán pomocí tříd nástroje TailwindCSS. Backend API v jazyce C# bylo implementováno mimo rámec této práce, řešilo se tedy pouze propojení mezi UI a API. Tento krok výrazně ulehčil nástroj Swagger. Díky použití opakujících se asynchronních požadavků na server, neboli *long-pollingu*, je možné uživateli zobrazit statistiky v reálném čase bez nutnosti odesílat požadavky v pravidelných intervalech.

Výsledná implementace byla otestována pomocí metody do-it-yourself. Toto testování se ukázalo být úspěšné a poskytlo řadu poznatků již během prvního testování. Některé z poznatků už byly i zakomponovány do výsledné aplikace.

V budoucnu se očekává nejvýznamnější změna v podobě integrace aplikace jako *mikro-frontend* do celého systému test-it-off. Celý vývoj a návrh proto počítal s touto možností a byl zaměřen na co největší zjednodušení implementace, až tato změna nastane. Dále by měla být provedena lepší optimalizace pro mobilní zařízení, aplikace je již nyní responzivní, avšak pouze do určité míry. Samozřejmě se bude čekat na zpětnou vazbu od zákazníků po prvním nasazení, a další úpravy a funkce budou plánovány podle jejich potřeb a požadavků.

Literatura

[1] Adams, D. *Learn TypeScript – The Ultimate Beginners Guide* [online]. [cit. 2023-05-01]. Dostupné z: https://www.freecodecamp.org/news/learn-typescript-beginners-guide/.

[2] Contributors, E. Y. . V. *Why Vite - The Problems* [online]. [cit. 2023-03-15]. Dostupné z: https://vitejs.dev/guide/why.html#the-problems.

[3] Daityari, S. *Angular vs React vs Vue: Which Framework to Choose* [online]. Březen 2023 [cit. 2023-03-27]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

[4] Foundation, P. S. *What is Python? Executive Summary* [online]. [cit. 2023-04-22]. Dostupné z: https://www.python.org/doc/essays/blurb/.

[5] Garbar, D. *Why use Front-end Frameworks* [online]. [cit. 2023-05-01]. Dostupné z: https://belitsoft.com/what-do-front-end-frameworks-do.

[6] Greif, S. *State of JavaScript - Results* [online]. 2022 [cit. 2023-04-20]. Dostupné z: https://stateofjs.com/.

[7] Group, R. *Recharts - A composable charting library built on React components* [online]. [cit. 2023-04-01]. Dostupné z: https://recharts.org/en-US.

[8] Hoffmann, J. *A Tale of Two Standards* [online]. Duben 2017 [cit. 2023-04-21]. Dostupné z: https://thehistoryoftheweb.com/when-standards-divide/.

[9] IBM. *What is a REST API?* [online]. [cit. 2023-04-21]. Dostupné z: https://www.ibm.com/topics/rest-apis.

[10] Imperva. *Minification* [online]. [cit. 2023-04-20]. Dostupné z: https://www.imperva.com/learn/performance/minification/.

[11] Inc., P. *Long Polling vs WebSockets* [online]. Únor 2023 [cit. 2023-03-25]. Dostupné z: https://www.pubnub.com/blog/long-polling-vs-websockets/.

[12] International, E. *ECMA-262 - ECMAScript® 2022 language specification* [online]. Červen 2022 [cit. 2023-04-25]. Dostupné z: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/.

[13] JavaTpoint. *React Hooks* [online]. [cit. 2023-05-01]. Dostupné z: https://www.javatpoint.com/react-hooks.

[14] Krotoff, T. *Frontend Frameworks Popularity* [online]. Únor 2023 [cit. 2023-04-25]. Dostupné z: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190.

[15] Krug, S. *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability*. New Riders, 2009. ISBN 978-0321657299.

[16] Labs, T. *Tailwind CSS* [online]. [cit. 2023-03-18]. Dostupné z: https://tailwindcss.com/docs/.

[17] Marsh, J. *UX for Beginners: A Crash Course in 100 Short Lessons*. O'Reilly Media, 2016. ISBN 978-1491912683.

[18] Meloni, J. C. *Sams Teach Yourself HTML, CSS, and JavaScript All in One, Third Edition*. 3. vyd. Pearson Education, Inc, 2018. ISBN 978-0672338083.

[19] Mozilla. *Glossary: Tree Shaking* [online]. Únor 2023 [cit. 2023-04-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Tree_shaking.

[20] Rouse, M. *C# (C Sharp)* [online]. Duben 2020 [cit. 2023-04-21]. Dostupné z: https://www.techopedia.com/definition/26272/c-sharp.

[21] Saadeghi, P. *DaisyUI - Tailwind CSS Components* [online]. [cit. 2023-03-20]. Dostupné z: https://daisyui.com/.

[22] Sheldon, R. *What is Node.js (Node)?* [online]. Listopad 2022 [cit. 2023-04-20]. Dostupné z: https://www.techtarget.com/whatis/definition/Nodejs.

[23] Software, S. *Swagger Documentation* [online]. [cit. 2023-03-25]. Dostupné z: https://swagger.io/docs/.

Příloha A

Obsah přiloženého paměťového média

Paměťové médium (CD) obsahuje následující soubory:

- **operatorui.zip** - zdrojové soubory aplikace
- **latexsrc.zip** - zdrojové soubory LaTeX použité pro vytvoření bakalářské práce
- **xtabac02_bp.pdf** - bakalářská práce ve formátu `.pdf`
- **videoDemonstration.mp4** - demonstrační video k výsledné implementaci
- **poster.pdf** - informační plakát o aplikaci