ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

VYUŽITÍ FRAMEWORKU ANGULAR NA PŘÍKLADU WEBOVÉ ONLINE HRY

BAKALÁŘSKÁ PRÁCE

Dominika Pudilová
Informatika se zaměřením na vzdělávání

Vedoucí práce: doc. Ing. Zdeněk Ulrych, Ph.D.

Plzeň, 2023

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně s použitím uvedené literatury a zdrojů informací.

V Plzni dne ……

.....

vlastnoruční podpis

Poděkování

Tímto děkuji vedoucímu této bakalářské práce doc. Ing. Zdeňku Ulrychovi, Ph.D. za odborné vedení práce a cenné rady. Ráda bych také poděkovala i bývalému vedoucímu práce PhDr. Tomáši Jakešovi, Ph.D. za pomoc s výběrem práce a trpělivost při konzultacích.

OBSAH

SEZNAM ZKRATEK

SPA – Single Page Application: webová aplikace, která funguje na jedné stránce dokumentu. Obsah stránky je aktualizován JavaScriptem.

npm – Node package manager: správce balíčků pro JavaScriptové knihovny.

CLI – command line interface: rozhraní příkazové řádky.

JS – JavaScript: programovací jazyk používaný nejčastěji pro tvoření interaktivních webů.

TS – TypeScript: typový jazyk postavený na JavaScriptu.

tsc – TypeScript compiler: TypeScriptový kompilátor.

IDE – integrated development environment: vývojové prostředí, software, který umožní vývojáři pohodlnou práci.

DI – Dependency Injection: vkládání instance třídy do jiné struktury jako závislost.

DOM – Document Object Model: rozhraní v HTML nebo XML dokumentu, které nakládá s jednotlivými elementy jako se stromovou strukturou a umožňuje k nim přístup.

ř. – řádek

Úvod

Framework Angular je v současné době jedním z nejrozšířenějších frameworků pro vývoj moderních webových aplikací. Tato bakalářská práce zasazuje Angular do kontextu ostatních používaných webových frameworků, zaměřuje se na vysvětlení základních konceptů a prvků frameworku a aplikuje je na příkladu webové online hry. Práce také obsahuje rychlý úvod do jazyka TypeScript, který je třeba pro práci s frameworkem použít, a zabývá se i konzolovým nástrojem Angular CLI a jeho použitím při vývoji.

V první části práce je vysvětlen základní princip funkcionality frameworku Angular, který se skládá z dvou hlavních prvků – komponent a služeb. Dále je představeno několik dalších důležitých konceptů, jako jsou směrovače, vkládání závislostí a asynchronní zpracování operací.

V druhé části práce je představena konkrétní implementace webové online hry, která bude sloužit jako příklad aplikace frameworku Angular. Tato hra má několik funkcí, které ukáží využití výše zmíněných prvků frameworku s názorným vysvětlením na příslušných částech kódu. Součástí práce je i zdrojový kód aplikace a návod na její spuštění.

Cílem této bakalářské práce je ukázat, že i přes to, že se framework zezačátku může zdát složitý, je možné jej použít pro vlastní projekty bez větších problémů. Práce by měla vysvětlit základní koncepty, ukázat jejich využití a zároveň prokázat, že použití frameworku Angular může být efektivní a přinést mnoho výhod při vývoji moderních webových aplikací.

1 PŘEHLED FRAMEWORKŮ

Framework je softwarový podklad pro podporu vývoje aplikací a programů. Jedná se o připravenou strukturu, kterou může vývojář použít pro urychlení práce a zjednodušení postupů. Kromě připraveného lešení, na kterém lze aplikaci stavět, může framework obsahovat i podpůrné nástroje a další knihovny.

Díky frameworku není třeba se soustředit na nízko úrovňové záležitosti a vývojář se poté může věnovat vlastnímu vývoji a práci na konkrétním programu. Ve frameworku je obsažen velmi obecný kód, který je ale přizpůsoben k přivlastnění. [1]

1.1 VÝHODY A NEVÝHODY

Frameworky slibují vyšší produktivitu díky jejich designu a znovu použitelnosti kódu. Tomu napomáhá také příprava frameworku pro automatické řešení některých problémů a jeho struktura, která navádí vývojáře k využití spolehlivých standardizovaných metod a provádí jej procesem vytváření projektu. Výsledkem je konzistentní a čistý kód aplikace. [2]

Framework také za vývojáře může řešit například otázky bezpečnosti. Systém může automaticky zabraňovat databázovým injekcím či jiným útokům díky jeho struktuře nebo použitím speciálních metod. [3]

Nevýhodou frameworků bývá vysoká počáteční investice, časová nebo peněžní, ať již se jedná o zakoupení, vytvoření vlastního systému nebo školení na open source framework. Frameworky bývají dosti složité, proto namísto aby se vývojář věnoval vytváření aplikace od nuly, bude se muset vzdělat v práci s existujícím frameworkem a naučit se využívat jeho funkce. Zezačátku může být časová náročnost podobná, avšak při opakovaném použití daného frameworku je již čas šetřen u každého dalšího projektu. [4]

Další nevýhodou je velké množství kódu. Jelikož jsou frameworky obecné a abstraktní, obsahují mnoho tříd a dalších struktur, které jsou připravené na propojení a na použití

[1] MANGER, Christian, Tomasz TREJDEROWSKI a Jarosław PADUCH. Advantages and disadvantages of framework programming with reference to Yii php framework, Gideon .net framework and other modern frameworks.
[2] RIEHLE, Dirk. *Framework Design*.
[3] Pros and cons of Frameworks in web development. Dostupné z: https://www.nerd.vision/post/pros-and-cons-of-frameworks-in-web-development.
[4] RIEHLE, Dirk. *Framework Design*.

vývojářem. Výsledný produkt může být potom s použitím frameworku mnohonásobně větší než bez jeho použití. Toto může navíc ovlivňovat i rychlost aplikace. [5]

Výhodou i nevýhodou může být verzování a aktualizace frameworku. Nová verze přináší starosti s aktualizací kódu, protože některé funkce se mohou stát zastaralými a vyžadují zásah do již funkčního programu a další strávený čas. Výhodou je ale využití nových technologií, implementace bezpečnostních prvků a zavedení nových funkcionalit a schopností frameworku. [6]

1.2 TYPY FRAMEWORKŮ

Na trhu lze najít velké množství frameworků pro různé typy použití. Kromě webových frameworků, na které se dále zaměříme, existují i frameworky pro tvoření mobilních aplikací, různé enginy, testovací frameworky a další. V dnešní době je již většina frameworků otevřená a zdarma, ale najdou se i placené platformy. Důvodem je nezájem o placené frameworky a malá komunita, zatímco otevřené frameworky jsou široce oblíbené a komunitou je do nich běžně přispíváno. [7] [8]

Webové frameworky jsou využívány pro stavbu webových aplikací, které využívají technologie jako internetový prohlížeč, databáze, webový server, HTML dokumenty či HTTP protokol. Kombinací vzniká aplikace, která je přístupná na internetu a umožňuje zobrazit uživateli statický i dynamický obsah. Webové aplikace nemusí být však pouze dostupné na internetu, mohou se provozovat i lokálně anebo je možné je stáhnout na počítač či mobilní zařízení pro off-line použití. [9]

Webové aplikace je možné rozdělit podle místa, kde je kód vykonáván, čili na straně serveru, nebo na straně klienta. Na webovém serveru jsou často provozovány aplikace schopné vygenerovat a poskládat dokumenty, které jsou následně odeslány na prohlížeč a zobrazeny jako webová stránka.

[5] MANGER, Christian, Tomasz TREJDEROWSKI a Jarosław PADUCH. Advantages and disadvantages of framework programming with reference to Yii php framework, Gideon .net framework and other modern frameworks.
[6] DEED, Ian. Pros and Cons of Web Development Frameworks.
[7] What Is a Framework? (Definition and Types of Frameworks). Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-a-framework.
[8] BHUSAL, Subash. Are frameworks always open-source? And If no, what are some examples of non open-source frameworks?.
[9] What are Progressive Web Apps?. Dostupné z: https://web.dev/what-are-pwas/.

Dalším typem back-endové aplikace je API, které je vytvořeno pro komunikaci s front-endovou aplikací. Jedná se o rozhraní, které odpovídá na dotazy a provádí žádané akce, například při zaslání autentizačních údajů může rozhodnout, zdali má uživatel právo přistoupit ke zdroji, a odeslat zpět token s oprávněním. Takový typ aplikací lze stavět s jazyky jako je PHP, Java a Python, ale hojně se dnes používá i JavaScriptový framework Express.js. [10]

Naopak front-endové aplikace jsou vykonávány na straně klienta a obsluhují pouze jednu stránku, kterou v případě potřeby aktualizují pouhým přepsáním její části. Takové aplikace se nazývají Single Page Applications (SPA) a jsou vykonávány výhradně JavaScriptem. SPA obecně zrychlují čas načítání a zpříjemňují uživatelskou interakci. Front-endem statických webů může být i pouhé HTML a CSS. [11]

1.3 BOOTSTRAP

Bootstrap je oblíbený jednoduchý framework pro stavbu webových stránek. V základu se jedná o sadu předdefinovaných CSS tříd, které při aplikování mění vzhled stránky, ale Bootstrap umí pracovat i s JavaScriptem a tím dosáhnout zajímavých efektů.

Bootstrap umožňuje využít mřížky pro strukturování obsahu stránek, a to pomocí CSS tříd. Bootstrap rozkládá stránku na 12 sloupců, přičemž vývojář si vybírá, kolik sloupců budou jednotlivé elementy na stránce zabírat. Kromě rozložení mohou tyto třídy pomoci i s responzivitou díky přednastaveným velikostem prvků pro různé velikosti obrazovek. [12]

Kromě responzivity se Bootstrap zabývá i přístupností – podporuje používání ARIA atributů, které přizpůsobují HTML čtečkám a dalším asistenčním technologiím. Tyto atributy jsou obsažené v dokumentaci ke každému prvku a jsou vyžadovány. [13]

Bootstrap obsahuje velké množství komponent, které může vývojář ve svém projektu použít, například tlačítka, odznáčky, vyskakovací okna, tooltipy, připravené navigační lišty, načítací animace a další. Dokáže pomoci i s tvorbou přehledných formulářů. Všechny prvky jsou stylizované do sady barev, kterou lze přizpůsobit dle vlastní paletky.

[10] 2021 Developer Survey. Dostupné z: https://insights.stackoverflow.com/survey/2021.
[11] SPA (Single-page application). Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/SPA.
[12] Grid system. Dostupné z: https://getbootstrap.com/docs/5.3/layout/grid/.
[13] ARIA. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA.

1.4 JAVASCRIPTOVÉ FRAMEWORKY

Mezi webovými frameworky jsou ty JavaScriptové v dnešní době nejpoužívanějšími. Mezi nejpopulárnější patří React.js, po něm Angular a pak Vue.js. Jedná se o srovnatelné frameworky, přičemž každý může být vhodný pro jiné použití. React je široce známý a používaný, zatímco Vue je jednoduchý a lehký. Angular je masivní, ale je dobře škálovatelný a vhodný pro velké aplikace. [14] [15]

Pro začínajícího vývojáře může být vhodný Vue, jelikož používá JavaScript a má nejjednodušší syntaxi. React ale může být díky své popularitě nejvíce žádaný u zaměstnavatelů. Angular je z těchto frameworků nejtěžší na naučení, protože používá nadstavbu JavaScriptu – TypeScript, musí se tedy vývojář učit nový jazyk. Na druhou stranu je Angular důkladně dokumentovaný s návody a ukázkami, je tedy také velmi přístupný pro nováčka.

Angular, na který bude práce dál zaměřena, je také výjimečný pro jeho nativní obousměrné vázání dat. Angular tedy umožňuje tok dat mezi rodičovskými prvky a jejich potomky, zatímco běžně u ostatních frameworků je pohyb dat povolen pouze jednosměrně z rodiče na potomka a podobné funkcionality lze dosáhnout jen s pomocí knihoven třetích stran. [16]

[14] 2021 Developer Survey. Dostupné z: https://insights.stackoverflow.com/survey/2021.
[15] SAKS, Elar. *JavaScript frameworks: Angular vs React vs Vue*.
[16] SAKS, Elar. *JavaScript frameworks: Angular vs React vs Vue*.

2 WEBOVÝ FRAMEWORK ANGULAR

Angular je open source platforma pro vývoj webových aplikací postavená na TypeScriptu. Kromě frameworku pro vytváření škálovatelných front-endových aplikací nabízí Angular také množství knihoven a řadu developerských nástrojů pro vývoj a testování kódu. [17]

Framework je vyvíjen Angular týmem ve firmě Google, ale přispívá k němu i komunita a některé korporace. Tento tým provádí nejen vývoj, ale zpracovává i důkladnou dokumentaci a spravuje blog, který se soustředí na novinky v okolí platformy Angular.

Základním stavebním kamenem Angularu jsou tzv. komponenty. Komponenta se skládá z HTML šablony, TypeScriptové třídy a kaskádových stylů. Tyto tři technologie se podílí na struktuře, obsahu, funkci a vzhledu jedné komponenty, která může představovat jakoukoli viditelnou i neviditelnou část rozhraní webové aplikace. [18]

K Angularu je dostupná detailní dokumentace včetně důkladného návodu pro vytvoření vlastní aplikace. Kromě oficiálních dokumentací a návodů lze také najít pomoc na internetových fórech a v článcích. Nejlépe dostupným návodem je nejspíše videokurz přímo od autorů, který je zdarma a lze jej najít na webu Angular University.

2.1 ANGULARJS A ANGULAR 2+

AngularJS je předchůdcem dnešního Angularu tak, jak ho známe. Byl vytvořen v roce 2010 jako nový JavaScriptový klient a byl dále rozšiřován, avšak brzy již nedokázal pokrýt potřeby nových webových aplikací.

Celý framework byl proto přetvořen, tentokrát v TypeScriptu, a v roce 2016 byl vydán Angular 2. Tento nový framework je jednodušší, čistější a výkonnější než jeho předchůdce. [19]

AngularJS je nyní zastaralý a používaný jen ve starších aplikacích, které nepřešly na novou technologii Angular 2+. Podpora pro AngularJS byla ukončena 31. prosince 2021.[20]

[17] What is Angular?. Dostupné z: https://angular.io/guide/what-is-angular.
[18] What is Angular?. Dostupné z: https://angular.io/guide/what-is-angular.
[19] HAMEDANI, Mosh. AngularJS vs Angular 2 vs Angular 4 | Mosh.
[20] THOMPSON, Mark. Discontinued Long Term Support for AngularJS.

Nový Angular je jeho autorským týmem udržován a pravidelně aktualizován s novou *major* verzí vycházející každých 6 měsíců. [21]

2.2 ANGULAR CLI

Angular CLI jedním z hlavních prvků platformy Angular. Jedná se o konzolový nástroj, který je při vývoji doporučeno používat, jelikož usnadní mnoho práce a urychlí některé postupy. Angular CLI dokáže vytvářet, spouštět a testovat celé projekty, umožňuje generovat komponenty i další typy souborů jednoduchými příkazy. Práce vývojáře je z velké části ulehčena, protože se nemusí starat o neustálé vytváření souborů a opisování opakujícího se kódu. [22]

Angular CLI je třeba nejdříve nainstalovat, a to nejjednodušeji pomocí npm [23]. Po nainstalování Angular CLI je možné jej používat přímo z příkazové řádky. Program je vyvolán použitím příkazu **`ng`** a jako parametr se udává samotná akce, kterou chceme provést, např. **`new`**, **`generate`**, **`test`**. Tyto příkazy dále často vyžadují vlastní parametry, jako třeba název projektu nebo typ souboru, který chceme generovat. [24]

Příkaz pro Angular CLI, kde bychom chtěli vygenerovat nový projekt a podpůrné soubory, bude vypadat následovně: **`ng new moje-aplikace`**. Pokud bychom chtěli vygenerovat novou komponentu pod jménem Account, použijeme **`ng generate component account`**. Všechny příkazy lze najít v oficiální dokumentaci Angularu.

2.3 VYGENEROVANÉ SOUBORY

Po založení nového projektu pomocí příkazu **`ng new <nazev-aplikace>`** bude ve složce s názvem aplikace vygenerováno množství souborů a složek, které budou využívány frameworkem Angular či jeho částmi.

Složka **`src`** bude obsahovat veškeré zdrojové soubory aplikace – všechny komponenty, testy, třídy, modely, služby a další. Druhým adresářem, který se v projektu objeví, je **`node_modules`**. Jedná se o složku obsahující všechny stažené knihovny a závislosti, které

[21] Angular versioning and releases. Dostupné z: https://angular.io/guide/releases.
[22] Getting started with Angular. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_getting_started.
[23] Node package manager: správce balíčků pro JavaScriptové knihovny
[24] CLI Overview and Command Reference. Dostupné z: https://angular.io/cli.

projekt bude mít. Angular defaultně některé své moduly zakomponuje do projektu a automaticky stáhne, a to právě do **node_modules**. Tato složka bude pravidelně aktualizována s novými zdrojovými soubory knihoven, ale i s jejich novými verzemi. Pokud se adresář v projektu nenachází, je třeba nejdříve spustit příkaz **npm install**, který potřebné závislosti stáhne.

Aplikace vygeneruje a připraví JSON soubory angular.json a package.json. Jsou si podobné, přesto každý obsluhuje jinou část aplikace. Angular.json bude obsahovat různá metadata k projektu, jako např. instrukce pro build, konfigurace některých základních nástrojů či nastavení globálních stylů a skriptů. Package.json bude popisovat, které balíčky, knihovny a dependence budou potřebné pro různé části projektu. Kromě seznamu závislostí obsahuje i název projektu a jeho verzi, jež si potom přebírají ostatní aplikace, např. při publikaci do npm registru. [25] [26]

2.4 TYPESCRIPT

TypeScript (zkr. TS) byl již dříve rychle zmíněn, zaslouží si však vlastní kapitolu. Jedná se o open-source vysokoúrovňový programovací jazyk, který byl vyvinut firmou Microsoft v roce 2012 a je jí dodnes udržován. TypeScript je nadstavbou JavaScriptu, která navíc přináší přídavnou vrstvu typové kontroly. Jelikož TS vychází z JS, je veškerý funkční JavaScriptový kód i funkčním TypeScriptovým kódem. TS kód je při spuštění transpilován[27] na čistý JS kód pomocí **tsc** – TypeScript kompilátoru. [28]

Výhodou TypeScriptu nad JavaScriptem je jeho striktní typová kontrola, která dokáže odhadnout datový typ proměnné a při jejím nevhodném použití vývojáře upozornit. Díky těmto vlastnostem dokáže IDE vhodně napovídat a obecně zabraňovat vzniku kódu, který by při spuštění nemohl fungovat. TS dále nabízí i nové datové typy **any**, **unknown**, **never** a **void**. [29]

[25] Workspace npm dependencies. Dostupné z: https://angular.io/guide/npm-packages.
[26] Package.json. Dostupné z: https://docs.npmjs.com/cli/v9/configuring-npm/package-json.
[27] Transpilace – proces převedení jednoho vysokoúrovňového jazyka do jiného vysokoúrovňového jazyka.
[28] JO FOLEY, Mary. Microsoft takes the wraps off TypeScript, a superset of JavaScript.
[29] TypeScript for JavaScript Programmers. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

Při udávání typu proměnné se píše ve formátu **`proměnná: typ`**. Běžně se také určuje typ vrácené hodnoty funkce, a to opět přidáním typu za dvojtečku.

```
public static getGenderIcon(gender: number) : IconDefinition {
  if (gender > 0) return faMars;
  return faVenus;
}
```

Obrázek 1 Ukázka striktního typování TypeScriptu na statické funkci (zdroj: vlastní)

Výše vyobrazená funkce přijímá hodnotu typu **`number`** a vrací ikonu typu **`IconDefinition`**.

Pro přípravu objektu lze v TypeScriptu použít **`interface`**, který umožňuje definovat jeho vlastnosti a jejich datové typy. Příkladem použití interface v praxi může být User, který bude mít ID, jméno a nepovinnou vlastnost s datem vytvoření účtu.

user.ts

```
1  export interface User {
2    id: number;
3    name: string;
4    createdAt?: Date;
5  }
```

Obrázek 2 Definice objektu za použití interface (zdroj: vlastní)

U vlastnosti **`createdAt`** byl umístěn otazník, který v TS indikuje nepovinnou hodnotu. Může tedy při vytváření User objektu zůstat prázdná. [30]

Při udávání datových typů umožňuje TS použít sjednocení více typů pomocí znaku svislé čáry [31]. Tímto lze TypeScriptu dát najevo, že datovým typem proměnné bude např. **`number`** nebo **`undefined`**, ale můžeme tak naznačit i výčtový typ, tedy výběr z několika předem daných hodnot.

```
mates: Array<any> | null = [];
```

Obrázek 3 Ukázka zápisu typu s použitím svislé čáry (zdroj: vlastní)

V případě, že budeme chtít proměnnou s těmito typy použít, TS nás nenechá vložit jinou hodnotu než tu, která bude typům odpovídat.

[30] More on Functions. Dostupné z: https://www.typescriptlang.org/docs/handbook/2/functions.html.
[31] Nazývána svislítkem, pipou či rourou, znamená běžně propojení výstupu jednoho programu do vstupu dalšího. V tomto případě se však jedná spíše o význam „nebo“. Značena |.

2.5 ZÁVISLOSTI

Závislost (také *dependence*) je označení pro softwarovou knihovnu, na níž je aplikace závislá, tzn. aplikace nebude schopna fungovat, pokud nebude požadovaná závislost dodána.

Angular využívá systém závislostí a samotný framework je rozdělen do několika balíčků, z nichž ty nejdůležitější jsou při generování projektu pomocí konzolového nástroje Angular CLI automaticky připravené. Závislosti jsou definovány formou jednoduchého objektu, který propojuje knihovnu a její verzi. Seznam závislostí je k nalezení v souboru package.json.

Kromě objektu **dependencies** můžeme naleznout v package.json i **devDependencies**. Rozdíl mezi nimi je takový, že v **devDependencies** budou balíčky potřebné pouze pro vývojářskou verzi programu, zatímco **dependencies** označují nezbytné knihovny i pro produkční verzi. [32]

Běžným příkladem vývojářské závislosti může být sada pro psaní a spouštění testů, Angular CLI nebo také samotný TypeScript. Obecnou závislostí může být jakákoli knihovna, např. Bootstrap, ikonové sady nebo RxJS. [33]

Framework Angular je modulární – je rozdělen do několika menších částí, které je možno nainstalovat jako závislosti. Určitě tedy v seznamu závislostí defaultně nalezneme **@angular/core** obsahující základní funkcionality frameworku, můžeme ale přidat i další moduly, např. **@angular/router** pro nastavení navigace v aplikaci či **@angular/forms**, které nabízí nástroje pro pokročilou práci s formuláři. [34]

Závislosti musí být při vývoji staženy, čehož se dá dosáhnout použitím příkazu **npm install**. Nato bude započata instalace závislostí, které budou nalezeny v package.json s nejvyšší možnou verzí. Zdrojový kód a další případné soubory budou po dokončení instalace k nalezení ve složce **node_modules**.

[32] Package.json. Dostupné z: https://docs.npmjs.com/cli/v9/configuring-npm/package-json.
[33] RxJS – knihovna pro reaktivní programování přinášející Observables pro přístupnější asynchronní kód
[34] Workspace npm dependencies. Dostupné z: https://angular.io/guide/npm-packages.

2.6 SPUŠTĚNÍ A NASAZENÍ APLIKACE

Aplikace je od začátku připravena pro jednoduché a rychlé spuštění z konzole. Spustit projekt pro lokální vývoj můžeme použitím Angular CLI příkazu: **`ng serve`**. Pokud jsme nezměnili nastavení v konfiguračním souboru angular.json, bude defaultně aplikace spuštěna na lokálním počítači s portem 4200 a bude k nalezení na adrese http://localhost:4200/ ve webovém prohlížeči. Je-li port obsazen nebo pokud chceme použít jiný port, je možno jej uvést rovnou v příkazu následovně: **`ng serve --port 4400`**.

Když je aplikace dokončena a je čas ji nahrát na vzdálený server, je možné použít příkaz: **`ng build`**. Výstupem bude celá transpilovaná aplikace včetně obrázků, ikon a dalších souborů, kterou bude možno defaultně najít v adresáři **`/dist/název-aplikace`**. Pro nasazení stačí překopírovat obsah tohoto adresáře na vzdálený server a správně nastavit přesměrování na soubor **`index.html`**.

Jelikož **`index.html`** je vstupem do aplikace postavené na Angularu, bude stačit, když webový server přesměruje veškerou komunikaci na tento HTML soubor. Zadá-li uživatel adresu s parametrem či cestou, např. http://mojestranka.cz/user/12, webový server v tuto chvíli nesmí hledat konkrétní soubor, ale naopak musí odpovědět se souborem **`index.html`**, odkud si již Angular a jeho Router URL přeloží a správně nasměrují.

Toto přesměrování se nastavuje v rámci konfigurace webového serveru a není závislé na Angularu, přesto je možné najít toto nastavení v oficiální dokumentaci.

3 TEORETICKÉ ZÁKLADY PRO WEBOVOU APLIKACI VYUŽÍVAJÍCÍ FRAMEWORK ANGULAR

Angular je oblíbený framework, který je však složitý na naučení. Následující kapitoly jsou zaměřeny na základní části frameworku a vysvětlení jejich funkcí. Tyto znalosti poté budou působit jako podklad pro vytvoření webové hry.

3.1 KOMPONENTY

Komponenta (anglicky *component*) je základním stavebním prvkem aplikace postavené na Angularu. Skládá se z HTML šablony a TypeScriptové třídy, které budou v projektu ve formě dvou stejnojmenných souborů ve formátech HTML a TS, přičemž součástí komponenty může být i CSS soubor se styly. Propojení těchto souborů v jeden celek probíhá za pomocí dekorátoru **`@Component`**, čímž je frameworku řečeno, že se jedná o komponentu. Šablona popisuje strukturu komponenty, zatímco TypeScriptová třída definuje proměnné a provádí logiku. [35]

Dekorátor **`@Component`** navíc obsahuje metadata o komponentě. Aby bylo možno ji použít, musí zde vývojář definovat její selektor. Dalšími běžnými metadaty jsou **`templateUrl`**, která obsahuje cestu k souboru šablony, a **`styleUrls`**, kde lze najít jednu nebo více cest k souborům se styly. Pokud je komponenta vygenerována pomocí Angular CLI, bude defaultně tyto tři vlastnosti obsahovat. [36]

[35] Component. Dostupné z: https://angular.io/api/core/Component.
[36] Beginning our Angular todo list app. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_todo_list_beginning.

not-implemented.component.ts

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'not-implemented',
5    templateUrl: './not-implemented.component.html',
6    styleUrls: ['./not-implemented.component.css']
7  })
8  export class NotImplementedComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit(): void {
13   }
14
15 }
```

Obrázek 4 Ukázka komponenty vygenerované pomocí Angular CLI (zdroj: vlastní)

Šablonu i styly lze také zapsat přímo v dekorátoru, a to za použití vlastností **`template`** a **`styles`**, ve kterých se bude nacházet rovnou HTML a CSS kód.

Angular defaultně využívá Shadow DOM, čímž lze dosáhnout toho, že styly budou aplikovány pouze v kontextu komponenty a neprosáknou mimo ni. Je tedy možno používat opakující se ID či jiné selektory, aniž by se navzájem ovlivňovaly napříč komponentami. Tento nebo jiný způsob zapouzdření lze vyjádřit v dekorátoru za použití vlastnosti **`encapsulation`**. Pokud Shadow DOM není podporován prohlížečem, lze jej také emulovat, případně může být zapouzdření úplně vypnuto, čímž se styly komponenty stanou globálními. [37]

Úkol komponent je převážně vizuální. Dynamické komponenty získávají data zvenčí, dle potřeby je mohou přetransformovat a pak je zobrazí na klientu. Komponenty reagují na uživatele a posílají jeho případný vstup hlouběji do aplikace.

3.2 SLUŽBY

Služba (anglicky *service*) je další nedílnou součástí aplikací využívajících frameworku Angular. Jedná se o funkční část aplikace, která se nezabývá vzhledem, nýbrž čistou

[37] View encapsulation. Dostupné z: https://angular.io/guide/view-encapsulation.

funkcionalitou. Na rozdíl od komponent nemají služby v aplikaci vizuální reprezentaci, neobsahují žádnou šablonu a skládají se pouze z TypeScriptové třídy.

Služby jsou úzce spojené s pojmem Dependency injection – služby jsou vkládány do jiné struktury jako závislosti. Aby framework věděl, že se jedná o službu a že ji bude vkládat do ostatních komponent, je třeba při její definici použít specifický dekorátor.

Nové služby lze generovat jednoduše příkazem **`ng generate service <jméno-služby>`** v Angular CLI. V praxi se takto definované služby používají například k získávání dat z API či jako globální pomocné třídy, ale můžou být obecně použity prakticky k jakémukoliv účelu.

3.3 DEPENDENCY INJECTION

Dependency injection (zkr. DI), přeloženo jako *Vkládání závislostí*, v Angularu je proces, kdy framework použije jednu instanci objektu napříč celou aplikací a dodává ji tam, kde je potřeba. Aby vývojář mohl označit třídu jako závislost, kterou budou ostatní služby či komponenty vyžadovat, musí použít dekorátor **`@Injectable`**.

Aby komponenta mohla získat tuto instanci, musí být vyžádána v konstruktoru závislé třídy. Při definici je možné použít klíčové slovo **`private`**, díky kterému bude závislost dostupná v celé třídě jako neveřejná vlastnost.

Ve chvíli, kdy se aplikace spouští, jsou injectable (injektovatelné) služby vytvořeny jednou a tato jedna instance se poté dodává tam, kde je vyžádána. Angular tuto architekturu hojně využívá a doporučuje, protože vede k přehledné struktuře kódu, zvýšené optimalizaci a zamezuje nadbytečným instancím tříd. [38]

Služby mohou být také závislé na jiných službách, avšak je třeba se vyvarovat zacyklení. Tato situace může nastat, když služba přímo nebo nepřímo závisí sama na sobě. První služba může záviset na druhé, která naopak závisí na té první. V tento moment vzniká zacyklení dependencí, před kterým ale Angular varuje kritickou chybou NG0200.

[38] Understanding dependency injection. Dostupné z: https://angular.io/guide/dependency-injection.

creature.service.ts

```
1  import {Injectable} from '@angular/core';
2  import {HttpClient} from "@angular/common/http";
3  import {environment} from "../../environments/environment";
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class CreatureService {
9
10   constructor(private http: HttpClient) { }
11
12   getAllCreatures() {
13     return this.http.get( url: environment.API_URL + "creature");
14   }
15 }
```

Obrázek 5 Ukázka služby, která přes DI využívá jiné služby pro svoji funkčnost (zdroj: vlastní)

Na obrázku výše (Obrázek 5) lze vidět službu, která je sama **`Injectable`** (vložitelná) a zároveň závisí na službě **`HttpClient`**. V jejím konstruktoru je využita syntaxe s klíčovým slovem **`private`**, díky které je potom dále modul dostupný pod názvem **`http`**, jako lze vidět ve funkci **`getAllCreatures()`**, bez potřeby další definice. V případě, kdy jiná komponenta bude vyžadovat výstup této funkce, bude moci získat instanci **`CreatureService`** přes DI obdobným způsobem.

3.4 DEKORÁTORY

Dekorátor (anglicky *decorator*) je aktuálně experimentální funkcionalitou TypeScriptu, která se používá pro anotaci či k přidávaní metadat k třídám a jejich vlastnostem. Obecně jsou dekorátory obyčejná funkce, která bude zavolána společně se strukturou, ke které je připojena. [39]

Angular dekorátorů využívá k označení specifických tříd či vlastností, přičemž tyto funkce už jsou pro vývojáře připravené a stačí je jen použít. Z dekorátorů již známe **`@Component`** pro označení komponent a **`@Injectable`** pro označení tříd pro DI, hned zezačátku se ale

[39] THEROX, Orta, Dinanjanan RAVINDRAN, Michael ESTEBAN, Lucas COSTA, Jack BATES a Iván OVEJERO. Decorators.

také setkáme s **`@NgModule`**, který se používá pro označení modulu, jeho metadat a jeho závislostí.

Komponenty mohou přijímat i vysílat své proměnné jiným komponentám. Jedná se o jeden ze způsobů, jak lze posílat data mezi komponentami v případě, kdy jsou do sebe vnořené.

Očekává-li komponenta data, bude k jedné z jejích proměnných připojen dekorátor **`@Input`**. Až data dostane, bude k nim moci přistoupit vyvoláním právě této proměnné.

Pokud komponenta data odesílá, bude na její proměnnou zavěšen dekorátor **`@Output`**.

3.5 DIREKTIVY

Direktivy (anglicky *directives*) jsou v Angularu speciální třídy, které přidávají vlastní chování existujícím elementům a lze je použít k manipulaci DOMu. Dělí se na direktivy strukturální a atributové, přičemž strukturální direktivy mění strukturu dokumentu a atributové mění atributy elementů. Posledním typem jsou komponentové direktivy, které spolupracují se šablonou a umožňují použití komponent.

Strukturální direktivy mohou měnit strukturu dokumentu, např. přidávat elementy nebo odebírat, a je třeba je označit znakem hvězdičky. Direktiva **`ngFor`** dokáže element duplikovat jako ve for cyklu, což je praktické v případě, kdy je potřeba vypsat více stejných či podobných prvků, např. pro vypsání prvků pole. Direktiva **`ngIf`** se používá v případě, kdy je třeba vybrat jeden z elementů na základě podmínky. Vhodné pro použití např. na přihlašovací stránce, kdy přihlášenému uživateli chceme ukázat jiný obsah než anonymnímu uživateli.

Atributové direktivy dokážou měnit vzhled nebo chování elementů v DOMu. Příkladem takové direktivy je **`ngClass`**, která dokáže přidat či odebrat CSS třídu na základě dané podmínky, nebo **`ngStyle`**, která podobným způsobem pracuje se styly elementu. [40]

Direktivy je také možné tvořit dle vlastních potřeb.

3.6 ASYNCHRONNÍ PROGRAMOVÁNÍ

Framework Angular využívá asynchronních prvků, převážně typy Observable a Promise. Tyto asynchronní typy se liší od běžného synchronního programování tím, že běžící program

[40] Attribute directives. Dostupné z: https://angular.io/guide/attribute-directives.

nebude čekat na jejich dokončení. Namísto toho je připraven skript, který bude proveden při dosažení určitých podmínek. Asynchronní akci je možné simulovat např. JavaScriptovou funkcí **`setTimeout()`**, která po uplynutí času provede danou funkci.

Promise je speciální objekt obsahující hodnotu, která může a nemusí být známa v době jeho vytvoření. Na Promise je třeba navázat funkci **`then()`**, v níž se definuje handler, který bude po dokončení asynchronní akce proveden. Promise může být ve třech stavech, a to pending (čeká), fulfilled (dokončený) anebo rejected (zamítnutý). Dokončený Promise vrátí handleru v **`then()`** hodnotu, zatímco rejected vrátí chybu, kterou je možno odchytnout a ošetřit. [41]

Observable je reprezentací toku dat, která jsou závislé na čase. V Observable může proudit jedna nebo více informací, ale i žádné. Pro získání těchto informací je třeba použít funkci **`subscribe()`**, která začne poslouchat na toku dat a v případě, že nějaká dostane, zavolá danou handler funkci. Observable je zvláštní tím, že její kód není proveden, dokud nad ní není zavolán **`subscribe()`**, což může být zdrojem některých chyb a nedorozumění při jejich používání. Observables jsou frameworkem Angular defaultně používány při HTTP dotazech, jelikož odpověď ze serveru je asynchronní. [42]

Promise i Observable jsou prvky asynchronního programování, ale Promise se zabývá jednou hodnotou, zatímco Observable jich může obsahovat více. Na rozdíl od Promise je možné výstup Observable i různě modifikovat, a to za použití knihovny RxJS. [43]

Nevýhodou Observable je nutnost zavolání funkce **`unsubscribe()`** ve chvíli, kdy již nejsou potřeba, a to i v komponentách, které automaticky zanikají např. při přecházení mezi stránkami.

3.7 POSÍLÁNÍ DAT

Ve frameworku Angular je přesně daný tok dat, a to z rodiče na potomky. Každá komponenta může posílat svá data hlouběji, avšak probublávání dat zpátky k rodičům je povoleno pouze za pomocí událostí (eventů).

[41] Promise. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.
[42] Observables in Angular. Dostupné z: https://angular.io/guide/observables-in-angular.
[43] Observables compared to other techniques. Dostupné z: https://angular.io/guide/comparing-observables.

3.7.1 INTERPOLACE

V rámci jedné komponenty lze propisovat hodnoty proměnných z TypeScriptové třídy do její HTML šablony pomocí interpolace řetězců. Používá se k jednoduchému vypsání řetězcové reprezentace proměnné do stránky. Výhodou je, že je-li hodnota v proměnné změněna, Angular zařídí její aktualizaci i v šabloně. Pro označení interpolace se používají zdvojené složené závorky s proměnnou uvnitř, jako lze vidět na následujícím obrázku (Obrázek 6). [44]

```
You are already logged in as {{ username }}.<br>
```

Obrázek 6 Použití interpolace pro výpis proměnné v šabloně komponenty (zdroj: vlastní)

3.7.2 PROPERTY BINDING

Property binding (přeloženo jako *vazba vlastnosti*) se používá k posílání dat z komponenty jako atribut HTML elementu. Vazbu lze použít v případě, kdy chceme provázat data komponenty s prvkem šablony. Na následujícím obrázku (Obrázek 7) lze vidět příklad, kde je atribut **`title`** vyplněn hodnotou na základě proměnné **`available`**.

```
<span [title]="available ? 'available' : 'unavailable'">●</span>
```

Obrázek 7 Atribut title je elementu přidělen pomocí property binding (zdroj: vlastní)

Property binding lze také použít k poslání dat z jedné komponenty do druhé, která data očekává díky použití dekorátoru **`@Input()`**. [45]

3.7.3 EVENT BINDING

Event binding (přeloženo jako *vazba událostí*) se používá k navázání dané funkce k události v DOMu. Na klasické události jako **`onclick`**, **`onchange`** či **`onkeypress`** lze reagovat připravenou funkcí v třídě komponenty. Událost je třeba zapsat bez předpony „on-“ a v kulatých závorkách, které určují, že se jedná o událost. [46]

```
<button (click)="logout()">Log out</button>
```

Obrázek 8 Navázání funkce na událost onclick (zdroj: vlastní)

[44] Displaying values with interpolation. Dostupné z: https://angular.io/guide/interpolation.
[45] Property binding. Dostupné z: https://angular.io/guide/property-binding.
[46] Event binding. Dostupné z: https://angular.io/guide/event-binding.

Vazbu je možné použít i pro komunikaci mezi rodičovskou komponentou a jejím potomkem. V potomkovi je třeba vlastnost třídy označit dekorátorem **`@Output`** a použít typ **`EventEmitter`**. Jedná se o speciální typ Observable, který umožňuje přes jeho funkci **`emit()`** odesílat data. V HTML rodiče se na místě, kde se volá dětská komponenta, použije event binding jako ve výše uvedeném příkladu, avšak namísto názvu události se použije název vlastnosti s **`@Output`** dekorátorem. [47]

3.7.4 TWO-WAY BINDING

Vazba Two-way binding (přeloženo jako *obousměrná vazba vlastnosti*) je rozdílná od všech předchozích způsobů tím, že data tečou nejen z komponenty do její šablony, ale je zde možnost získat data i z živé webové stránky zpět do komponenty. Nejčastěji se využívá pro přečtení uživatelského vstupu, např. ve formulářích. Pro její zápis se používá tzv. „banana in a box“ syntaxe, kdy požadovanou proměnnou zapíšeme uvnitř kulatých a hranatých závorek. Zápis lze vidět v následující ukázce na obrázku (Obrázek 9). [48]

```
<nav-top [(rightMenuCollapsed)]="rightMenuCollapsed"></nav-top>
```

Obrázek 9 Ukázka použití two-way binding (zdroj: vlastní)

3.8 ROUTOVÁNÍ

Routování neboli směrování je systém, pomocí kterého může uživatel navigovat mezi různými stránkami a zobrazovat je. Při vytváření nového projektu příkazem **`ng new moje-aplikace`** budeme programem Angular CLI dotázáni, zdali chceme do projektu rovnou zakomponovat Angular routing. Pokud jej zvolíme, bude vygenerovaný projekt obsahovat i připravené routování ve formě použitého modulu **`RouterModule`**. [49]

`RouterModule` musí být naimportován v hlavním modulu aplikace, tedy bude obsažen v poli **`imports`** v **`app.module.ts`**. V **`RouterModule`** bude obsaženo pole cest (routes), které budou mapovat adresu URL na komponentu. Při zadání konkrétní adresy bude potom uživateli zobrazena komponenta, která je na ni navázaná. Route je jednoduchý objekt se dvěma vlastnostmi: **`path`** obsahující část URL a **`component`**, značící komponentu, která bude při navštívení URL zobrazena.

[47] Observables in Angular. Dostupné z: https://angular.io/guide/observables-in-angular.
[48] Two-way binding. Dostupné z: https://angular.io/guide/two-way-binding.
[49] Angular Routing. Dostupné z: https://angular.io/guide/routing-overview.

V routách lze použít i proměnné parametry, např. ID v URL. Jelikož každé ID bude jiné, lze použít speciální zápis s dvojtečkou.

Nakonec je třeba naznačit, kde se budou dané komponenty zobrazovat, čehož lze dosáhnout pomocí elementu **<router-outlet>**. Uvnitř tohoto elementu se budou vykreslovat dané komponenty a jejich obsah, běžně se tedy dává hned do kořene aplikace **app.component.html**. [50]

3.8.1 ROUTE GUARD

Pro omezení přístupu neautorizovaných uživatelů k některým komponentám je možno použít systém Route guard. Každá route může být chráněna vlastní funkcí, kterou si uživatel může sám napsat.

Route guard lze vygenerovat pomocí příkazu **ng generate guard <jméno>**. Jedná se o obyčejnou službu, která ale implementuje rozhraní **canActivate** a obsahuje stejnojmennou funkci. Funkce bude sestavena tak, aby vracela boolean, ať už synchronně či asynchronně. Bude-li výsledek pravda, bude uživateli vyžádaná stránka zobrazena. Guard může vracet také **UrlTree**, čímž bude spuštěna nová navigace. [51]

3.8.2 ROUTER-LINK

Pro navigaci mezi stránkami se v systému Angular nepoužívá kotva s atributem **href**, ale s direktivou **routerLink**. Požadovaná cesta se může zapsat jako řetězec, ale i jako pole řetězců, kde skládáme různé parametry cesty dynamicky. [52]

3.9 HTTPCLIENT

Pro komunikaci s API se v Angularu využívá **Injectable** služba **HttpClient**. Služba umožňuje poskládat HTTP dotaz včetně nastavení vlastní hlavičky a odeslat jej na daný server. Hlavičku je možné upravovat před odesláním, což může být vhodné např. pro připojení autentizačního tokenu.

[50] Router tutorial: tour of heroes. Dostupné z: https://angular.io/guide/router-tutorial-toh.
[51] Router tutorial: tour of heroes. Dostupné z: https://angular.io/guide/router-tutorial-toh.
[52] Router tutorial: tour of heroes. Dostupné z: https://angular.io/guide/router-tutorial-toh.

Pro získávání dat stačí použít funkci **`get()`** třídy **`HttpClient`**, zatímco pro odesílání dotazů a další modifikaci zdrojů na serveru lze použít funkce odpovídajících HTTP metod: **`post()`**, **`put()`**, **`patch()`** a **`delete()`**.

Funkce třídy **`HttpClient`** přijímají hlavně adresu API, ale funkce odesílající data vyžadují i objekt ve formátu JSON obsahující dané informace. Dotazy na server jsou asynchronní, proto funkce třídy **`HttpClient`** vrací odpověď ve formě Observable. Výslednou hodnotu si již můžeme v Angularu zpracovat dle vlastních potřeb, avšak dokud nezavoláme **`subscribe()`**, nebude dotaz proveden. [53]

[53] Communicating with backend services using HTTP. Dostupné z: https://angular.io/guide/http.

4 NÁVRH APLIKACE A POPIS ARCHITEKTURY

V následujících kapitolách budou vysvětleny detaily k připravené webové online hře. Nejdříve bude hra obecně popsána včetně jejího základního fungování v kontextu frameworku a poté bude popsáno několik situací, které při vývoji nastaly, a jejich řešení. Nakonec bude kapitola obsahovat detailní návod pro spuštění aplikace.

4.1 FUNGOVÁNÍ HRY

Webová hra bude v první řadě umožňovat registraci a přihlášení uživatelů, aby každý mohl přistupovat ke svým zdrojům. Autentizace bude probíhat pomocí JWT tokenu, který uživatel získá ze serveru při úspěšném přihlášení. Token bude uložen do prohlížeče a bude obsažen v každém dalším dotazu na server.

Po přihlášení se uživatel může pohybovat mezi různými stránkami, každá nabízející jinou aktivitu.

Cíl hry není pevně dán. Hra neobsahuje ani všechny funkcionality a bude se v budoucnosti dále rozšiřovat. Uživatel při registraci dostane mazlíčka, o kterého se bude starat. Toto zvíře bude mít vrozené vlastnosti jako druh, rasu, pohlaví i genetické vlastnosti. Některé vlastnosti bude možno trénovat. Jakákoliv aktivita bude vyčíslena v množství energie, které zvíře bude muset spotřebovat. Hráč tedy musí dohlížet na to, aby jeho subjekty měly pro vyžadovanou aktivitu dost energie a při jejím nedostatku nechat zvíře odpočinout.

Mazlíček může uživateli získávat cenné materiály prostřednictvím expedic a také může produkovat nové potomky, přičemž nová vejce je třeba nejdříve inkubovat pomocí různých drahých kamenů. Vylíhnuté mládě má šanci získat vlastnosti rodičů, které se mohou zlepšit, ale úplně stejně i zhoršit. Vrozené vlastnosti mají vliv na rychlost expedic a ovlivňují tím jejich účinnost. Ukázku přehledu zvířete, jeho genetiky, energie a nástrojů pro práci s ním lze najít v příloze (Příloha 3).

4.2 STRUKTURA STRÁNEK A HIERARCHIE

Uživatel bude moci navigovat mezi stránkami, jako je hlavní stránka, přehled zvířat, detail jednoho zvířete i seznamu vlastněných vajec. Hierarchii je třeba rozmyslet již zezačátku, protože velmi často se bude jedna stránka rovnat jedné či více komponent. Abychom je tedy mohli začít generovat, musíme vědět, co na komponentě bude a jak se bude jmenovat.

Diagram (Příloha 1 a Příloha 2) naznačuje předběžné rozdělení aplikace na komponenty a různé stránky. Ve žlutém rámečku lze vidět ty stránky, které bude moci navštívit pouze přihlášený uživatel. Komponenty ve spodní části diagramu jsou oddělené od ostatních stránek, protože jsou na nich nezávislé – jedná se o navigační sloupce, zápatí a inventář, které všechny budou zobrazitelné na více než jedné stránce.

V diagramu lze také vidět propojení stránek a složení jedné stránky z více komponent. Stránka s detailem zvířete bude rozdělena na tři části, přičemž hlavní část bude v komponentě **`Creature`** a po stranách budou dva podobné bloky z komponent **`ActionsLeft`** a **`ActionsRight`**. Tyto bloky jsou od hlavní komponenty odděleny pro lepší přehlednost, protože budou samy obsahovat větší množství logiky. Ukázku **`Creature`** komponenty včetně panelů, ovládacích prvků a zobrazení genetiky zvířete lze najít v přílohách (Příloha 3).

4.3 VZHLED

Aplikace bude používat klasické kaskádové styly pro vzhled stránek s pomocí Bootstrapu pro zjednodušení některých prvků tvoření front-endu.

Uživatel se bude pohybovat po aplikaci pomocí běžných tlačítek. Na levé straně bude navigační lišta s tlačítky pro přepínání mezi různými místnostmi – stáje, vajíčka, dashboard a na pravé straně se budou nacházet administrační nástroje jako nastavení účtu a odhlášení. Nad samotným obsahem bude také lišta zobrazující uživatelův účet a jeho prostředky. Ukázku rozložení je možné vidět v příloze (Příloha 4).

Některé funkcionality vyžadují vyskakovací okno, jako například výběr expedice či sbírání odměn z expedic. S vytvořením modálních oken může dobře pomoci Bootstrap, protože tuto funkcionalitu umí a jeho knihovna pro Angular je také podporuje. Takové modální okno lze vidět v příloze (Příloha 9).

Bootstrap bude použit v celé aplikaci také pro rozložení obsahu do mřížky a budou se používat i jeho ostatní prvky, jako třeba tlačítka, navigační lišty a dropdown elementy.

4.4 SLUŽBY

Předem je jasné, že v aplikaci bude třeba využívat služeb. Naše komponenty potřebují zdroj dat a úkol s jejich získáváním bude delegován na příslušné služby. Aplikace bude komunikovat se vzdáleným API, ze kterého bude daná data získávat.

Pro práci s uživatelskými daty bude existovat **`UserService`**. Tato služba bude získávat a měnit data spojená s uživatelem. Abychom oddělili dvě podobná témata, vytvoříme ještě **`AuthService`**, službu, která se bude zabývat registrací a přihlášením uživatele a také nakládáním s autentizačním tokenem.

Získaný token bude uložen na lokálním počítači do cookie, teoreticky bychom proto mohli vytvořit i **`CookieService`** pro ukládání, čtení a dekódování těchto malých bloků dat.

Mezi službami můžeme najít také **`AuthHeaderInterceptor`**. Jedná se o speciální strukturu, která bude kontrolovat každý odeslaný dotaz a přidávat k němu hlavičku s autorizačním tokenem.

Projekt bude obsahovat i množství služeb navázaných na konkrétní elementy hry, např. **`CreatureService`** pro pracování se zvířaty či **`ResourceService`** pro transakce se surovinami. Zajímavým využitím formátu služby může být i **`TimerService`** – třída, která bude umět odpočítávat ubývající sekundy z časového rozsahu.

4.5 BACK-END A API

Angular je front-endový framework, neumí tedy defaultně komunikovat s databázemi a ani se nezabývá dlouhodobým ukládáním dat. Umí však získávat data z API, ta transformovat dle potřeb a zobrazovat je uživateli.

API je obecně rozhraní pro získávání dat z jiné aplikace, v našem případě bude hra komunikovat s PHP skriptem na veřejném serveru, který bude na základě přijatých parametrů a dat odpovídat s relevantními daty získanými z databáze. Komunikace bude probíhat ve formě HTTP dotazů pomocí metod GET, POST, PATCH a DELETE.

Server bude kontrolovat správnost a konzistenci přijatých dat a v případě, že budou správná, odešle odpověď ve formátu JSON s vhodným kódem. Pokud přijatá data nebudou validní, nastane neočekávaná chyba nebo se nedokončí transakce v databázi, odešle server

odpověď s textem chyby a kódem 500. V případě, že front-end vyžádá nevhodnou metodu nebo odešle nevhodná či prázdná data, dostane odpověď s kódem ve formátu 4xx. Úspěšné dotazy budou vracet kódy 200 a 204.

Pokud bude API dosti konzistentní ve svých odpovědích, bude možné využít kódy a obsah JSON objektů pro odchycení chyb a zobrazování chybových hlášek či jiných zpráv ze serveru.

V případě, kdy bude uživatel provádět různé akce, jako třeba poslání zvířete trénovat, musí server dostat správné informace, jinak k trénování nemůže dojít. Na pozadí proběhne odeslání dotazu na API obsahující ID zvířete a vlastnost, kterou chce uživatel trénovat. Odpověď z API bude obsahovat informaci o tom, zdali byla akce úspěšná a dopočítaný čas, kdy bude zvíře opět dostupné. Angular si tuto odpověď přebere a adekvátně zobrazí výsledek uživateli, například vykreslí potvrzující hlášku.

4.6 NÁVOD NA SPUŠTĚNÍ

Hotová aplikace je dostupná k přečtení a ke stažení v repozitáři na portálu GitHub. Aplikace je ke stažení zadarmo a získaný kód může být použit pro další rozvíjení znalostí frameworku či jako opora při vytváření vlastní obdobné aplikace.

4.6.1 INSTALACE NPM

Nejjednodušeji lze správce balíčků npm nainstalovat skrze Node.js. Na oficiálních stránkách Node.js stáhneme poslední dlouhodobě podporovanou verzi programu (LTS) a spustíme instalaci. Po dokončení instalace můžeme dostupnost programů Node.js a npm zjistit v příkazové řádce dotazem na verzi:

- node -v
- npm -v

Pokud byla instalace úspěšná, měli bychom v příkazovém řádku vidět verze příslušných programů, očekávaný výstup lze vidět v příloze (Příloha 5). V tomto kroku jsme připraveni nainstalovat Angular CLI.

4.6.2 INSTALACE ANGULAR CLI

Nástroj Angular CLI lze nainstalovat za pomocí programu npm použitím příkazu **`npm install -g @angular/cli`**.[54] Volba **`-g`** značí globální instalaci, tedy program nebude nainstalován do aktuálního adresáře, ale bude dostupný v celém systému. Úspěšnou instalaci je možné zkontrolovat příkazem **`ng version`**.

4.6.3 STAŽENÍ ZDROJOVÉHO KÓDU

Zdrojový kód je veřejně dostupný na portálu GitHub. [55] Kód může být získán např. stažením celého projektu v souboru ZIP, ale i pomocí programu GitHub CLI nebo SSH. Rozhraní nabízející různé možnosti stažení lze nalézt v příloze (Příloha 6).

4.6.4 SPUŠTĚNÍ PRO DEV

Po stažení lze aplikaci lokálně spustit, a to za pomocí již nainstalovaného programu Angular CLI. Aplikace by měla v tuto chvíli správně fungovat a měla by se otevřít na stránce pro přihlášení. Pro spuštění v prohlížeči bude v konzoli připraven odkaz nebo stačí zadat URL: http://localhost:4200/.

[54] CLI Overview and Command Reference. Dostupné z: https://angular.io/cli.
[55] Zdrojový kód je dostupný zde: https://github.com/dominikapudilova/dragonseAngularBP

5 NÁZORNÉ UKÁZKY KÓDU V ANGULARU

Hotová hra obsahuje ukázky základních i pokročilých konceptů, se kterými framework Angular pracuje a které můžou být vysvětleny pro jeho lepší pochopení. V následujících kapitolách bude možné nalézt praktické příklady spojující prvky z hry a dříve vysvětlené koncepty frameworku Angular.

Veškeré ukázky kódu lze najít i ve zdrojových souborech aplikace, která je dostupná na portálu GitHub.

5.1 DASHBOARD

Dashboard je první stránka, na kterou se uživatel dostane po přihlášení. Bude to hlavní stránka, kde by měl být přehled všeho, co by uživatele mohlo zajímat. Jelikož je prozatím hra poměrně prázdná, bude Dashboard ukazovat pouze probíhající expedici. V budoucnosti ale může obsahovat i připravená vajíčka, přehled inventáře, vybraná zvířata nebo hráčovy úspěchy. Bude se jednat o jednoduchou komponentu, na které bude vidět vložení služby (DI) a propojení s ostatními komponentami.

dashboard.component.ts

```typescript
import {Component} from '@angular/core';
import {AuthService} from "../services/auth.service";
import {firstValueFrom} from "rxjs";

@Component({
  selector: 'dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent {
  user: any;

  constructor(private authService: AuthService) {
    firstValueFrom(this.authService.user).then((user : User ) => {
      this.user = user;
    });
  }
}
```

Obrázek 10 TypeScriptový kód komponenty Dashboard (zdroj: vlastní)

Na obrázku (Obrázek 10) lze vidět soubor s koncovkou **component.ts** a dekorátor **@Component**, naznačující třídu komponenty. Na ř. 13 probíhá vložení **AuthService** (DI) a následně její použití pro získání dat uživatele. Tato data jsou uložena do vlastnosti třídy **this.user**. V šabloně lze potom k těmto datům přistoupit odkázáním na proměnnou **user**.

dashboard.component.html

```
1  <header [username]="user.name" ></header>
2
3  <div class="container">
4    <div class="row">
5      <expedition-module class="col-6"></expedition-module>
6      <achievement-module class="col-6"></achievement-module>
7    </div>
8  </div>
```

Obrázek 11 HTML kód šablony pro komponentu Dashboard (zdroj: vlastní)

V dekorátoru komponenty bylo možné vidět odkaz na soubor šablony **dashboard.component.html**. Při zavolání komponenty **Dashboard** bude vykreslena tato šablona. V kódu na obrázku (Obrázek 11) lze vidět nové tagy **<header>**, **<expedition-module>** a **<achievement-module>**, jedná se o další komponenty, které si volá sám **Dashboard**. Do komponenty **Header** je pomocí Property binding odesláno dříve získané uživatelské jméno, aby hlavička mohla hráče pozdravit.

5.2 DIREKTIVY

Direktivy **ngIf** a **ngFor** mohou v některých situacích ušetřit mnoho práce, protože umožňují dynamicky transformovat šablonu komponenty. Díky direktivě **ngClass** lze zase měnit styly na základě proměnných a podmínek.

5.2.1 NGFOR

Direktivu **ngFor** lze ve hře použít k výpisu jednotlivých zvířat ve stáji. V komponentě **Stables** můžeme pro každé zvíře v poli vykreslit samotnou komponentu, čehož lze dosáhnout velmi jednoduše.

stables.component.html

```
1  <header></header>
2  <div class="content">
3    <div class="creatures">
4      <creature-card class="creature"
5                     *ngFor="let creature of creatureCollection$ | async"
6                     [creature]="creature">
7      </creature-card>
8    </div>
9  </div>
```

Obrázek 12 Výpis kolekce prvků v šabloně komponenty Stables (zdroj: vlastní)

Na obrázku (Obrázek 12) lze vidět, že celá šablona na vykreslení seznamu zvířat má pouhých 9 řádků. Direktiva **ngFor** zde zakládá dočasnou proměnnou **creature** a pro každou položku z kolekce vytvoří novou komponentu **creature-card**. Proměnná **creatureCollection$** je v tomto případě asynchronní Observable, která je rozbalena využitím async roury. Tato roura umožňuje Observable rozbalit a přečíst rovnou v šabloně bez použití funkce **subscribe()**.

V kódu komponenty (Obrázek 13) lze na ř. 18 vidět, že **creatureCollection$** je Observable obsahující odpověď z dotazu na získání všech zvířat.

stables.component.ts

```
1  import ...
7
8  @Component({
9    selector: 'stables',
10   templateUrl: './stables.component.html',
11   styleUrls: ['./stables.component.css']
12 })
13 export class StablesComponent {
14
15   creatureCollection$: Observable<any>;
16
17   constructor(private creatureService: CreatureService) {
18     this.creatureCollection$ = creatureService.getAllCreatures();
19   }
```

Obrázek 13 Třída komponenty Stables a původ proměnné creatureCollection$ (zdroj: vlastní)

5.2.2 NGIF

Podobně jako **ngFor**, direktiva **ngIf** má také řadu praktických využití. Lze ji využít např. pro skrytí či zobrazení elementu na základě dané podmínky a také ji lze použít pro kontrolu, zdali daná proměnná obsahuje hodnotu, aby nenastala chyba.

home.component.html

```
1   <div class="welcome-img-container"...>
4
5   <div class="welcome-login module module-bg-light">
6     <h2>Log in & play</h2>
7
8     <ng-template #alreadyLogged>
9       <p class="font-p mt-3">
10        You are already logged in as {{ username }}.<br>
11        Continue to <a routerLink="/dashboard">Dashboard</a>
12      </p>
13      <button class="btn btn-dark-info mt-2" (click)="logout()">Log out</button>
14    </ng-template>
15
16    <ng-container *ngIf="!loggedIn; else alreadyLogged">
17      <login></login>
18
19      <p class="text-muted font-p mb-0 mt-3">
20        Don't have an account yet? Register<a href="/register">here</a>.
21      </p>
22    </ng-container>
23  </div>
```

Obrázek 14 Ukázka použití direktivy ngIf, pojmenování šablony a její použití v else větvi. (zdroj: vlastní)

V šabloně komponenty **Home** (Obrázek 14) lze vidět podmínku v direktivě **ngIf** na ř. 16. Ve třídě komponenty existuje vlastnost **loggedIn** držící informaci o tom, zdali se uživatel přihlásil či nikoliv. Direktiva jednoduše vyhodnotí tuto proměnnou a v případě, že výsledek je pravda, bude zobrazen kód, který je obsažen v kontejneru **<ng-container>**.

Za povšimnutí stojí použití výrazu **else** uvnitř podmínky direktivy **ngIf**. Výraz poukazuje na úsek kódu, který má být použit v případě, kdy podmínka bude vyhodnocena jako nepravda. V tento moment nebude níže připravený úsek kódu zobrazen a místo něj bude vykreslena šablona na ř. 8-14. Propojení podmínky a šablony probíhá jejím pojmenováním **#alreadyLogged** a použitím tohoto jména v podmínce direktivy.

5.2.3 NGCLASS

Direktivu **ngClass** lze použít k jednoduchému přepínání CSS tříd na základě podmínky či proměnné.

expedition-module.component.html

```
10  <div class="row row-cols-2">
11    <div class="col-md-auto">
12      <button class="btn btn-dark no-hover dashboard-icon"
13          [ngClass]="closestExpedition ? 'bg-' + closestExpedition['element_name'] : ''">
14        <fa-icon [icon]="faQuestion" size="2x"></fa-icon>
15      </button>
16    </div>
17
18    <div class="col">
19      <span>Expedition site:</span><br>
20      <span *ngIf="closestExpedition else noOngoingExpeditions">
21        <strong>{{ closestExpedition.name }}</strong>
22      </span>
23      <ng-template #noOngoingExpeditions>
24        <span><strong class="text-nowrap">No ongoing expeditions at the moment</strong></span>
25      </ng-template>
26    </div>
27  </div>
```

Obrázek 15 Ukázka použití direktivy ngClass. (zdroj: vlastní)

V šabloně komponenty **ExpeditionModule** (Obrázek 15) lze vidět použití direktivy **ngClass** s ternárním operátorem na ř. 13. Pokud proměnná **closestExpedition** existuje, bude prvku přidělena třída s vhodným názvem. Tato třída potom způsobí podbarvení prvku v odstínu daného elementu. Výsledný vzhled komponenty lze vidět v příloze (Příloha 7).

5.3 ROUTOVÁNÍ

Ve hře chceme, aby bylo možné přecházet mezi různými stránkami a odkazovat na ně, čehož dosáhneme právě použitím routování. Při vytváření nového projektu je dobré si pro ulehčení práce nechat routování připravit přímo nástrojem Angular CLI.

5.3.1 NASTAVENÍ ROUTERMODULE

V hlavním modulu aplikace (Obrázek 16) lze vidět použití cest a jejich mapování na konkrétní komponenty na ř. 64-78. Pořadí route objektů je důležité, proto se jako první připravuje prázdná cesta a jako poslední musí být výraz zahrnující všechny cesty (`'**'`). Porovnávání URL s cestami route objektů probíhá postupně shora dolů a je vhodné je řadit od nejvíce konkrétních po nejméně konkrétní.

app.module.ts

```
1   import ...
35
36  @NgModule({
37    declarations: [...],
63    imports: [
64      RouterModule.forRoot( routes: [
65        {path: '', component: HomeComponent},
66        {path: 'register', component: RegisterComponent},
67
68        {path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard]},
69        {path: 'stables', component: StablesComponent, canActivate: [AuthGuard]},
70        {path: 'hatchery', component: HatcheryComponent, canActivate: [AuthGuard]},
71        {path: 'store', component: NotImplementedComponent, canActivate: [AuthGuard]},
72        {path: 'achievements', component: NotImplementedComponent, canActivate: [AuthGuard]},
73        {path: 'wiki', component: NotImplementedComponent, canActivate: [AuthGuard]},
74
75        {path: 'creature/:id', component: CreatureComponent, canActivate: [AuthGuard]},
76        {path: 'egg/:id', component: EggComponent, canActivate: [AuthGuard]},
77
78        {path: '**', component: NotFoundComponent},
79      ]),
80      BrowserModule,
81      HttpClientModule,
82      FontAwesomeModule,
83      NgbModule,
84      ReactiveFormsModule,
85    ],
86    providers: [...],
93    bootstrap: [AppComponent]
94  })
95  export class AppModule { }
```

Obrázek 16 Hlavní modul aplikace s nastavenými cestami v RouterModule (zdroj: vlastní)

V poslední route lze vidět, že odkazuje na **`NotFound`** komponentu. Jakákoliv jiná cesta, než jsou výše zmíněné, totiž není validní cestou a odkazuje na neexistující stránku.

5.3.2 ZÍSKÁNÍ PARAMETRU Z URL

V cestách **`creature`** a **`egg`** v **`RouterModule`** (Obrázek 16) lze vidět proměnnou ID zapsanou jako **`:id`**. Toto umožní v aplikaci určovat, na jaké zvíře se přesně díváme. Modul routeru přesné ID znát nepotřebuje, ale je třeba na něj myslet při vytváření odkazů a v samotných **`Creature`** a **`Egg`** komponentách se musí ID z URL vyextrahovat.

Objekt **`ActivatedRoute`** umožňuje nahlédnout na aktuální URL a získat její parametry. Vlastnost **`paramMap`** je Observable a umožňuje přístup k parametrům URL po použití funkce **`subscribe()`**.

creature.component.ts

```
44    constructor(
45      private route: ActivatedRoute,
46      private router: Router,
47      private creatureService: CreatureService,
48      private pushService: PushService
49    ) {}
50
51    ngOnInit(): void {
52      this.route.paramMap.subscribe( next: (params : ParamMap ) => {
53        if (params.has( name: 'id')) {
54          this.id = parseInt(<string>params.get('id'));
55        } else {
56          this.id = 0;
57        }
58        if (this.id > 0) {...}
98      });
99    }
```

Obrázek 17 Získání parametru ID z URL přes ActivatedRoute a paramMap (zdroj: vlastní)

Ve třídě **Creature** komponenty (Obrázek 17) je možné na ř. 52-54 vidět způsob, kterým lze získat a uložit parametr ID. Po získání ID je pak možné zavolat na API a dotázat se na údaje pro konkrétní zvíře.

5.3.3 SKLÁDÁNÍ ODKAZŮ

Při odkazování mezi stránkami se směrováním je třeba použít direktivu **RouterLink**. Její hodnotu lze přiřadit jako řetězec, ale v případě více parametrů je třeba použít pole řetězců.

Ve stáji vypisujeme řadu komponent **CreatureCard** pomocí **ngFor**. Tyto komponenty se chovají jako kartička se základními informacemi, ale zároveň jsou i odkazem pro přesměrování na stránku s konkrétním zvířetem.

creature-card.component.html

```
1    <ng-container *ngIf="creature">
2      <a [routerLink]="['/creature', creature.id]">
```

Obrázek 18 Komponenta CreatureCard obsahuje odkaz na zvíře dle ID (zdroj: vlastní)

Šablonu komponenty **CreatureCard** obsahující odkaz i parametr ID v direktivě **RouterLink** lze vidět na obrázku (Obrázek 18). Vzhled několika kartiček **CreatureCard** je možné vidět v příloze (Příloha 8).

5.4 MODÁLNÍ OKNA

Ve hře se modální okna objevují v případech, kdy obsah není dost velký na to, aby pro něj musela existovat vlastní stránka. Vybere-li uživatel akci pro vyslání zvířete na expedici, stačí mu malé okno pro výběr místa a doby trvání, k čemuž postačí modální okno.

Modální okna lze vytvořit a ovládat pomocí knihovny Bootstrap za použití služby **NgbModal**. Její funkce **open()** umožňuje okna otevírat, přičemž jako parametr je třeba dodat již připravenou šablonu. Pro vytvoření šablony je možné použít element **<ng-template>** s pojmenováním ve formátu #jméno, které lze pak použít jako parametr pro funkci **open()**.

actions-left.component.html

```
<ng-template #expeditionModal let-modal>
  <div class="modal-header">
    <h5 class="modal-title">
      <span class="text-capitalize">{{ selectedExpeditionElement.name }}</span>
      element expedition
    </h5>
    <button type="button" class="btn-close" aria-label="Close"
            (click)="modal.dismiss()"></button>
  </div>
  <div class="modal-body"...>
  <div class="modal-footer">
    <div class="container">
      <div class="row row-cols-3">
        <div class="col"></div>
        <div class="col">
          <button type="button" class="btn btn-dark btn-dark-long"
                  (click)="modal.dismiss()">Close</button>
        </div>
        <div class="col">
          <button type="submit" class="btn btn-dark-info btn-dark-long"
                  (click)="modal.close(true)">
            <span>Send off</span>
            <span *ngIf="expeditionEnergyCost > 0">
              {{ expeditionEnergyCost }} <fa-icon [icon]="faBolt"></fa-icon>
            </span>
          </button>
        </div>
      </div>
    </div>
  </div>
</ng-template>
```

Obrázek 19 Šablona modálního okna pro použití Bootstrapem (zdroj: vlastní)

Na obrázku (Obrázek 19) je možné vidět celou šablonu modálního okna pro výběr expedic. Element **<ng-template>** připraví HTML, které do zavolání nebude na stránce zobrazeno. Šablona je na ř. 31 pojmenována **#expeditionModal** a do elementu je přidán atribut **let-modal**.

Dále šablona obsahuje elementy pro rozdělení struktury, které pochází z oficiální dokumentace Bootstrapu, jako např. oddělení záhlaví, těla a zápatí pomocí elementů ** ** a příslušných tříd. Za povšimnutí stojí **onclick** eventy, které jsou použity pro tři tlačítka v okně: křížek v záhlaví a tlačítka Potvrdit a Zrušit v zápatí (ř. 38, 77 a 81). Křížek a tlačítko Zrušit používají funkci **dismiss()**, která okno zavře, zatímco tlačítko pro potvrzení expedice volá funkci **close()**. Otevření modálu pomocí funkce **open()** vytváří Promise, který bude dokončen uzavřením okna s funkcí **close()**. Do ní lze poslat libovolnou hodnotu, která bude při dokončení daného Promise vrácena.

actions-left.component.ts

```
79   openExpeditionModal(modalContent: any, elementName: string) {
80     this.selectedExpeditionElement = null;
81     this.selectedExpeditionElement = this.elements.find(
82       ({ name }) => name === elementName
83     );
84     if (this.selectedExpeditionElement) {
85       this.expeditionService.getExpeditionsByElement(this.selectedExpeditionElement.id).then(
86         (res: any) => {...}, (error) => {...}
94       );
95       this.expeditionService.getExpeditionLengths().then(
96         (res: any) => {...}, (error) => {...}
104      );
105      this.openModal(modalContent).then(
106        (result) => {
107          if (result === true) {...}
125        }, () => { }
126      );
127    }
128  }
129
130  private openModal(modalContent: any) {
131    return this.modalService.open(modalContent,  options: {centered: true, size: "lg"}).result;
132  }
```

Obrázek 20 Funkce, které umožňují otevření a vyhodnocení modálního okna v třídě komponenty (zdroj: vlastní)

Na obrázku (Obrázek 20) lze vidět funkce komponenty, kde se modální okno otevírá. Samotné otevření probíhá na ř. 131, avšak jako první je volána funkce **openExpeditionModal()**, která nejdříve získá informace o expedici a pak na ř. 105

připravuje handler pro Promise. Hodnota **`true`**, kterou posílá tlačítko v šabloně (Obrázek 19) na ř. 81 bude získána a porovnána po dokončení Promise v komponentě na ř. 106 pod názvem **`result`**.

Ve funkci **`open()`** na ř. 131 lze také vidět možnosti modálního okna v druhém parametru. Kromě vycentrování a velikosti lze nastavit i pozadí, animaci, fullscreen či přidávat CSS třídy. Ukázku finálního modálního okna lze najít v příloze (Příloha 9).

5.5 NAVRŽENÁ ŘEŠENÍ VYBRANÝCH SITUACÍ V APLIKACI

V aplikaci je řešeno mnoho specifických problémů, jejichž řešení nemusí být na první pohled vždy jasné. V kapitolách níže jsou popsané konkrétní situace a jedno z jejich možných řešení – nejedná se však o jediné ani zaručeně nejlepší řešení, způsobů je mnoho a některé mohou být lepší než jiné.

5.5.1 PŘEJMENOVÁNÍ ZVÍŘETE

Pro přejmenování zvířete je třeba připravit tlačítko, na které uživatel klikne a zpřístupní formulář, přičemž staré jméno je třeba skrýt a po uložení přepsat na nové. Jak donutit jednu část hlavičky se schovat a druhou ukázat? Jak provést změnu jména tak, aby se zapsalo i do databáze a neprojevilo se jen na front-endu?

Schování jedné části a zobrazení formuláře pro přejmenování lze dosáhnout direktivou **`ngIf`** v kombinaci s boolean proměnnou, která určuje stav procesu přejmenování. Pokud **`renaming = true`**, chceme vykreslit formulář a v opačném případě chceme zobrazit jméno zvířete s tlačítkem pro přejmenování.

V třídě komponenty připravíme funkci **`rename()`**, která zadané jméno zvaliduje a odešle na API. Nakonec stačí změnit stav **`renaming = false`** a zobrazit uživateli nové jméno. Příklad vyobrazení jména a formuláře lze vidět v příloze (Příloha 10).

creature.component.html

```
23  <ng-container *ngIf="renaming == false else renameInput">
24    <span id="creature-name">{{ creature.name }}</span>
25    <button id="detail-rename-button" class="btn btn-dark-transparent"
26            type="button" (click)="renaming = true">
27      <fa-icon [icon]="faPen"></fa-icon>
28    </button>
29  </ng-container>
30
31  <ng-template #renameInput>
32    <div class="row align-items-center justify-content-center">
33      <div class="col-3"></div>
34      <div class="col-4">
35        <div class="input-group">
36          <input #nameInput type="text" class="form-control" [value]="creature.name">
37        </div>
38      </div>
39      <div class="col-2">
40        <button class="btn btn-dark-info float-start"
41                (click)="renaming = false; rename(nameInput.value)">
42          <fa-icon [icon]="faPen"></fa-icon>
43        </button>
44      </div>
45      <div class="col-3"></div>
46    </div>
47  </ng-template>
```

Obrázek 21 Část šablony pro přejmenování zvířete (zdroj: vlastní)

Na obrázku (Obrázek 21) je možné vidět část šablony zabývající se přejmenováním zvířete v **`Creature`** komponentě. Na ř. 23 se nachází použití proměnné **`renaming`** v direktivě **`ngIf`** a na ř. 31-47 lze vidět šablonu, která bude zobrazena v případě, kdy uživatel začne proces přejmenovávání. Za povšimnutí stojí pojmenování elementu **`input`** na ř. 36, jehož hodnota je poté rovnou odeslána do funkce **`rename()`** na ř. 41.

5.5.2 ČASOVAČ

Trénování, expedice i odpočívání jsou akce, které ve hře trvají nějakou dobu a uživatel musí počkat, než proběhnou do konce a zvíře bude opět dostupné pro další aktivity.

Jelikož aplikace často operuje s odpočítáváním času, je vhodné připravit službu, která se jím bude zabývat. Tato služba může převádět čas z řetězce na objekt, dopočítávat zbývající čas do nějakého momentu a použít Observable pro odpočítávání zbývajícího času. Jak takový Timer připravit a jak ho ve zbytku aplikace použít?

TimerService je služba, jejíž funkce **start()** vrací Observable obsahující zbývající čas do dokončení určité akce ve hře. Používá funkci **timer()**, která pracuje s nastavením intervalu a vrací Observable. Timer bude nastaven tak, aby každou sekundu vracel počet zbývajících sekund.

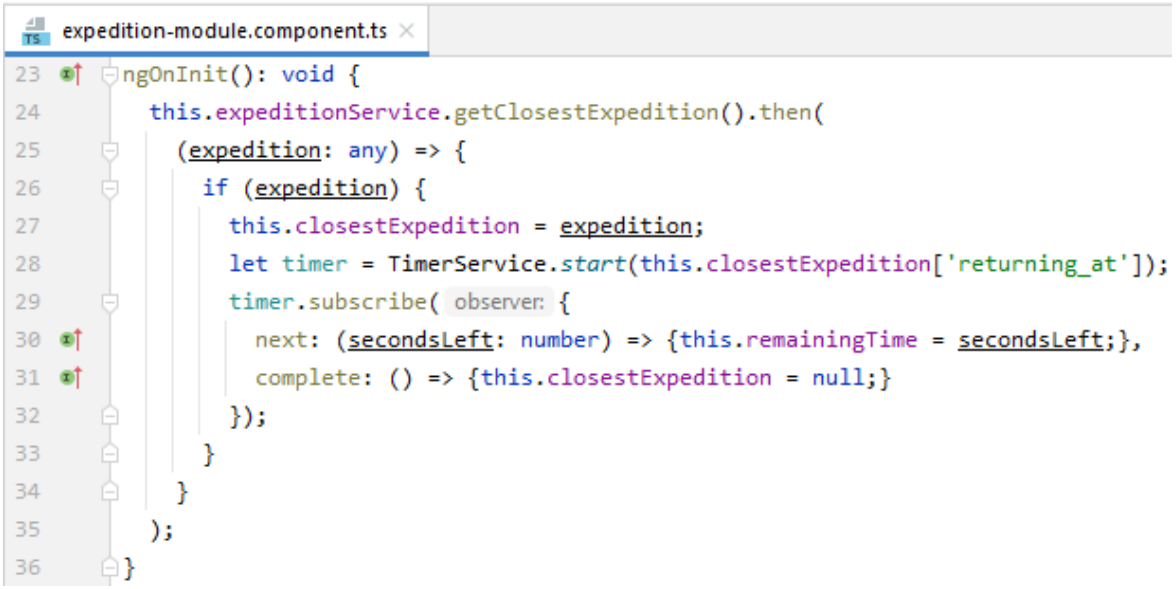
timer.service.ts

```
11  public static start(availableAt: string) : Observable<number> {
12    let availableAtSeconds = new Date(availableAt).getTime();
13    let nowSeconds = new Date().getTime();
14
15    let remainingSeconds = Math.round( x: (availableAtSeconds - nowSeconds) / 1000);
16    if (remainingSeconds <= 0) { return new Observable<number>; }
17
18    return timer( startDue: 1000, intervalDuration: 1000)
19      .pipe(
20        takeWhile( predicate: () => remainingSeconds > 0),
21        tap( next: () => remainingSeconds = remainingSeconds - 1),
22        map( project: () => remainingSeconds)
23      );
24  }
```

Obrázek 22 Funkce TimerService, která vrací Observable s dopočítaným zbývajícím počtem sekund na základě přijatého data v budoucnosti (zdroj: vlastní)

Na obrázku (Obrázek 22) lze vidět funkci **start()**, která dostává časový údaj v budoucnosti a dopočítává k němu zbývající počet sekund. Na ř. 19-22 lze vidět transformaci Observable pomocí RxJS operátorů, které v tomto případě odečítají po jedné z původního počtu zbývajících sekund, dokud se nedostanou k nule.

Komponenta, která odpočítávání vyžaduje (např. detail zvířete), bude začínat i ukončovat subscription k Observable z **TimerService** na základě dění ve hře. Spustí-li uživatel např. trénování, bude funkce **start()** zavolána a komponenta bude vykreslovat zbývající sekundy, které z ní získá. Zbývající čas je vhodné vypisovat ve formátu HH:MM:SS, což si ale komponenta už zpracuje sama.



Obrázek 23 Komponenta ExpeditionModule spouštějící odpočítávání z TimerService (zdroj: vlastní)

V komponentě **ExpeditionModule** (Obrázek 23) lze vidět jeden ze způsobů, jak spustit časovač v **TimerService** a použít jeho výstup. Komponenta **ExpeditionModule** je součástí Dashboardu a při jejím vytvoření je odeslán dotaz na API pro zjištění, zdali aktuálně neprobíhá expedice a pokud ano, jaký je termín návratu. Na základě získaných informací může být spuštěn Timer, což lze vidět na ř. 28-29. Následující řádky s klíčovými slovy **next** a **complete** definují, co se má stát při získání další hodnoty z Observable a co má být provedeno po jejím dokončení. Proměnná **remainingTime** je použita v šabloně pro výpis zbývajícího času. Ukázku komponenty **ExpeditionModule** s výpisem času lze vidět v příloze (Příloha 7).

5.5.3 PUSHSERVICE

Některé funkcionality hry vyžadují aktualizaci jedné komponenty, i když akce probíhá v jiné. Příkladem je přičtení peněz ve chvíli, kdy uživatel odklikne odměny z expedic nebo při prodeji zvířete. Akce se odehrává v detailu zvířete, ale my potřebujeme aktualizovat množství zlata v hlavičce stránky. Jakým způsobem umožnit komunikaci mezi komponentami, které jsou na sobě nezávislé? Jak poslat data mezi komponentami, které spolu nejsou ve vztahu rodič-potomek?

Pro dosažení komunikace mezi na sobě nezávislými komponentami je možné vytvořit službu, která tuto komunikaci bude zprostředkovávat. Komponenta, která bude očekávat signál zvenku, bude odebírat Observable vlastnost **pushCurrency** nové služby

PushService. Komponenta, která na druhou stranu bude chtít signál vyslat, zavolá vhodnou funkci v **PushService**, čímž způsobí vyslání signálu pro obnovení hodnoty všem ostatním komponentám, které budou vlastnost **pushCurrency** poslouchat. Vlastnost **pushCurrency** je typu **BehaviorSubject**, což je typ Observable, a bude vysílat boolean hodnoty. Pokaždé, co tato Observable vyšle novou hodnotu, všechny odebírající komponenty provedou svou akci, např. provedou dotaz pro získání nového množství zlata.

push.service.ts

```
1  import { Injectable } from '@angular/core';
2  import { BehaviorSubject } from "rxjs";
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class PushService {
8    private _pushCurrency: BehaviorSubject<boolean> = new BehaviorSubject( _value: true);
9
10   pushCurrency() {
11     this._pushCurrency.next( value: true);
12   }
13   balance() {
14     return this._pushCurrency;
15   }
16 }
```

Obrázek 24 Služba PushService, její vlastnost pushCurrency a funkce pro vyslání i přijetí signálu (zdroj: vlastní)

Na obrázku (Obrázek 24) je možné vidět službu **PushService**, která přes funkci **pushCurrency()** umožňuje vyslat signál všem komponentám odebírajícím její Observable. K té lze přistoupit přes getter **balance()**.

BehaviorSubject může obsahovat i jiné hodnoty než boolean. Může např. pracovat s typem **number**, který bude označovat ID materiálu, jehož data je třeba obnovit. Služba pro takový systém signalizování musí být Injectable, jelikož chceme napříč aplikací pouze jednu instanci jejích Observable vlastností. Komponenty ji potom získají pomocí Dependency Injection.

5.5.4 CHYBOVÉ HLÁŠKY

V aplikaci se lze setkat s některými situacemi, kdy rodičovská komponenta bude reagovat na události z jejího potomka, např. na stránce detailu zvířete. Aktivity jako expedice či

trénování se nachází ve vlastní komponentě, ale způsobují změnu stavu, kterou je třeba projevit v jejím rodiči, **Creature** komponentě. Takovou změnu stavu může způsobit spuštění trénování, které povýší vybranou schopnost zvířete, ale třeba i chybové hlášky či varování, které je třeba zobrazit jednotně v nadřazené komponentě. Jak signalizovat rodičovské komponentě, že má aktualizovat své hodnoty? Jak zprostředkovat komunikaci z potomka na rodiče?

Odpověď jsou události. Rodiče mohou poslouchat dětské události pomocí Event binding, které probíhá v HTML. Na událost potomka poté může rodičovská komponenta reagovat provoláním vlastní funkce s přijatými daty z události.

Pro jednotné vypisování chybových hlášek je třeba v rodičovské komponentě připravit proměnnou typu řetězec nebo pole řetězců. Dále bude rodič obsahovat funkci **addMessage()**, která do proměnné uloží danou chybovou hlášku přejatou z potomka.

V potomku je třeba použít dekorátor **@Output** a typ **EventEmitter** u vlastnosti **errorMessage**, která bude vysílat případnou chybovou hlášku. V případě, že nastane chyba, zavolá komponenta funkci **emit()** nad vlastností **errorMessage**, přičemž v parametru bude obsah chybové hlášky typu **string**. Funkce **emit()** způsobí vyslání chybové hlášky k rodiči. Nakonec stačí provázat událost potomka s připravenou handler funkcí **addMessage()** rodiče v jeho HTML.

creature.component.html

```
59  <div class="col-md-2">
60    <detail-actions-left
61      [creatureId]="creature.id"
62      (errorMessage)="addMessage($event)"
63      (availableAt)="setAvailableAt($event)"
64      (reloadStats)="prepareCreatureInformation()">
65    </detail-actions-left>
66  </div>
```

Obrázek 25 Šablona rodičovské komponenty, která používá mimo jiné event binding pro přijímání dat z potomka (zdroj: vlastní)

V šabloně **Creature** komponenty (Obrázek 25) je možné vidět na ř. 62 propojení eventu **errorMessage** z potomka na funkci **addMessage()**. Klíčové slovo **$event** umožní předání

hodnoty z události rovnou do funkce, přičemž hodnotou je v tomto případě samotná chybová hláška.

Kromě chybových hlášek lze obdobným způsobem také signalizovat změnu vlastností při trénování či nedostupnost zvířete v případě, že je posláno na expedici.

ZÁVĚR

Výsledkem této bakalářské práce je ukázka využití frameworku Angular na konkrétním příkladu v podobě webové online hry. Práce se nejprve věnovala popisu frameworku Angular a jeho základních prvků. Následně byla navržena a implementována hra, která tyto prvky využívá.

Výsledná online hra umožňuje registraci a přihlášení uživatelů a obsahuje několik různých herních prvků. Ve hře je možné pečovat o virtuální mazlíčky, kteří mají vrozené vlastnosti jako druh, rasu, pohlaví a genetické vlastnosti. Tyto vlastnosti je možno trénovat a postupně šlechtit produkováním nových potomků. Mazlíčci se rodí z vajec, která je třeba nejdříve inkubovat, a to za pomocí vzácných kamenů, přičemž tyto kameny lze získat z výprav. Na výběr je z různých typů expedic, které umožňují získat různé materiály a mají různé délky trvání. Hráč musí také nakládat s mazlíčkovou energií a v případě potřeby nechat zvíře odpočinout. Cíl hry není dán, avšak je nabádáno k získávání lepších genetických vlastností.

V praktické části práce byly demonstrovány schopnosti a výhody použití frameworku Angular v rámci vývoje webové aplikace. Na kódu hry je ukázáno několik částí frameworku, např. využití komponent, direktiv i správné nastavení směrování. Praktická část obsahuje také řadu vybraných situací, které při vývoji aplikace nastaly, a návrh jejich řešení. Výsledný program je dostupný jako součást výstupu praktické části práce.

Výsledkem této práce je ucelená ukázka toho, jak lze framework Angular využít pro vývoj moderních webových aplikací, včetně her.

RESUMÉ

Tato bakalářská práce se zabývá platformou Angular a jejím využitím pro tvorbu webových aplikací. První část se zaměřuje na frameworky obecně a popisuje i ostatní dostupné webové frameworky. Dále se práce zaměřuje na framework Angular a na práci s konzolovým nástrojem Angular CLI. Nakonec jsou v teoretické části popsány základní prvky frameworku, jejich význam a použití.

Pro praktickou část byla vytvořena webová online hra, na které je možné základní prvky frameworku ukázat. Práce popisuje, jakou mají některé části frameworku funkci a vysvětluje, jak je použít s ukázkami na kódu hotové aplikace. Kromě základů je vysvětleno i několik praktických situací, které při vývoji aplikace nastaly, včetně jejich řešení a ukázek kódu. Zdrojový kód hry je součástí práce a je dostupný ke stažení na portálu GitHub.

ABSTRACT

This thesis deals with the Angular platform and its use in web application development. The first part focuses on frameworks in general and describes other available web frameworks as well. Furthermore, the thesis focuses on the Angular framework and working with the Angular CLI tool. Finally, the theoretical part describes the basic elements of the framework, their significance and usage.

For the practical part, an online browser game that demonstrates the basic elements of the framework was created. The thesis explains the functionality and usage of some parts of the framework through the code of the finished application. In addition to the basics, several practical situations that occurred during the development are explained, including their solutions and code samples. The source code of the game is part of the thesis and is available for download on GitHub.

SEZNAM LITERATURY

2021 Developer urvey. In: *Stack Overflow* [online]. New York: Stack Exchange Inc. [cit. 2023-04-05]. Dostupné z: https://insights.stackoverflow.com/survey/2021

Angular Routing. In: *Angular.io* [online]. Mountain View, California: Google [cit. 2023-04-18]. Dostupné z: https://angular.io/guide/routing-overview

Angular versioning and releases. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2022-11-09]. Dostupné z: https://angular.io/guide/releases

ARIA. In: *MDN Web Docs* [online]. [cit. 2023-04-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

Attribute directives. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2023-03-24]. Dostupné z: https://angular.io/guide/attribute-directives

Beginning our Angular todo list app. In: *MDN Web Docs* [online]. [cit. 2023-04-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_todo_list_beginning

BHUSAL, Subash. Are frameworks always open-source? And If no, what are some examples of non open-source frameworks?. In: *Quora* [online]. 2023 [cit. 2023-04-12]. Dostupné z: https://www.quora.com/Are-frameworks-always-open-source-And-If-no-what-are-some-examples-of-non-open-source-frameworks/answer/Subash-Bhusal-2

CLI Overview and Command Reference. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-03-27]. Dostupné z: https://angular.io/cli

Communicating with backend services using HTTP. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-04-16]. Dostupné z: https://angular.io/guide/http

Component. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-04-12]. Dostupné z: https://angular.io/api/core/Component

DEED, Ian. Pros and Cons of Web Development Frameworks. In: *Pangea* [online]. Pangea.ai [cit. 2023-04-05]. Dostupné z: https://www.pangea.ai/dev-web-development-resources/best-practices/

Displaying values with interpolation. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2023-04-08]. Dostupné z: https://angular.io/guide/interpolation

Event binding. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-04-12]. Dostupné z: https://angular.io/guide/event-binding

Getting started with Angular. In: *MDN Web Docs* [online]. [cit. 2023-04-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_getting_started

Grid system. In: *Bootstrap* [online]. [cit. 2023-04-18]. Dostupné z: https://getbootstrap.com/docs/5.3/layout/grid/

HAMEDANI, Mosh. AngularJS vs Angular 2 vs Angular 4 | Mosh. In: *Youtube* [online]. Mountain View, California: Google [cit. 2023-04-12]. Dostupné z: https://youtu.be/9AaRJ8COXdM

JO FOLEY, Mary. Microsoft takes the wraps off TypeScript, a superset of JavaScript. In: *ZDNet* [online]. Red Ventures, 2023 [cit. 2023-04-12]. Dostupné z:

https://www.zdnet.com/article/microsoft-takes-the-wraps-off-typescript-a-superset-of-javascript/

MANGER, Christian, Tomasz TREJDEROWSKI a Jarosław PADUCH. Advantages and disadvantages of framework programming with reference to Yii php framework, Gideon .net framework and other modern frameworks. *Studia Informatica*. 2010, **31**(4), 119-137. ISSN 1642-0489.

More on Functions. In: *TypeScript* [online]. Redmond, Washington: Microsoft, c2012-2023 [cit. 2023-04-12]. Dostupné z: https://www.typescriptlang.org/docs/handbook/2/functions.html

Observables compared to other techniques. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2023-04-10]. Dostupné z: https://angular.io/guide/comparing-observables

Observables in Angular. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2023-04-08]. Dostupné z: https://angular.io/guide/observables-in-angular

Package.json: Specifics of npm's package.json handling. In: *Npm Docs* [online]. 2022 [cit. 2023-03-04]. Dostupné z: https://docs.npmjs.com/cli/v9/configuring-npm/package-json

Promise. In: *MDN Web Docs* [online]. [cit. 2023-04-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Property binding. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2023-04-08]. Dostupné z: https://angular.io/guide/property-binding

Pros and cons of Frameworks in web development. In: *NerdVision* [online]. [cit. 2023-04-12]. Dostupné z: https://www.nerd.vision/post/pros-and-cons-of-frameworks-in-web-development

RIEHLE, Dirk. *Framework Design: A Role Modeling Approach*. Zürich, Switzerland, 2000. Ph.D. Thesis. Universität Hamburg.

Router tutorial: tour of heroes. In: *Angular.io* [online]. Mountain View, California: Google [cit. 2023-03-29]. Dostupné z: https://angular.io/guide/router-tutorial-toh

SAKS, Elar. *JavaScript frameworks: Angular vs React vs Vue*. Helsinki, Finsko, 2019. Bachelor's Thesis. Haaga-Helia University of Applied Sciences.

SPA (Single-page application). In: *MDN Web Docs* [online]. [cit. 2023-04-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/SPA

THEROX, Orta, Dinanjanan RAVINDRAN, Michael ESTEBAN, Lucas COSTA, Jack BATES a Iván OVEJERO. Decorators. In: *TypeScript* [online]. Redmond, Washington: Microsoft, 2023 [cit. 2023-03-08]. Dostupné z: https://www.typescriptlang.org/docs/handbook/decorators.html

THOMPSON, Mark. Discontinued Long Term Support for AngularJS. In: *Angular Blog* [online]. [cit. 2022-11-09]. Dostupné z: https://blog.angular.io/discontinued-long-term-support-for-angularjs-cc066b82e65a

Two-way binding. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-03-27]. Dostupné z: https://angular.io/guide/two-way-binding

TypeScript for JavaScript Programmers. In: *TypeScript* [online]. Redmond, Washington: Microsoft, 2023 [cit. 2023-02-26]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Understanding dependency injection. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-04-12]. Dostupné z: https://angular.io/guide/dependency-injection

View encapsulation. In: *Angular.io* [online]. Mountain View, California: Google, c2010-2023 [cit. 2023-04-12]. Dostupné z: https://angular.io/guide/view-encapsulation

What are Progressive Web Apps?. In: *Web.dev* [online]. Mountain View, California: Google [cit. 2023-04-06]. Dostupné z: https://web.dev/what-are-pwas/

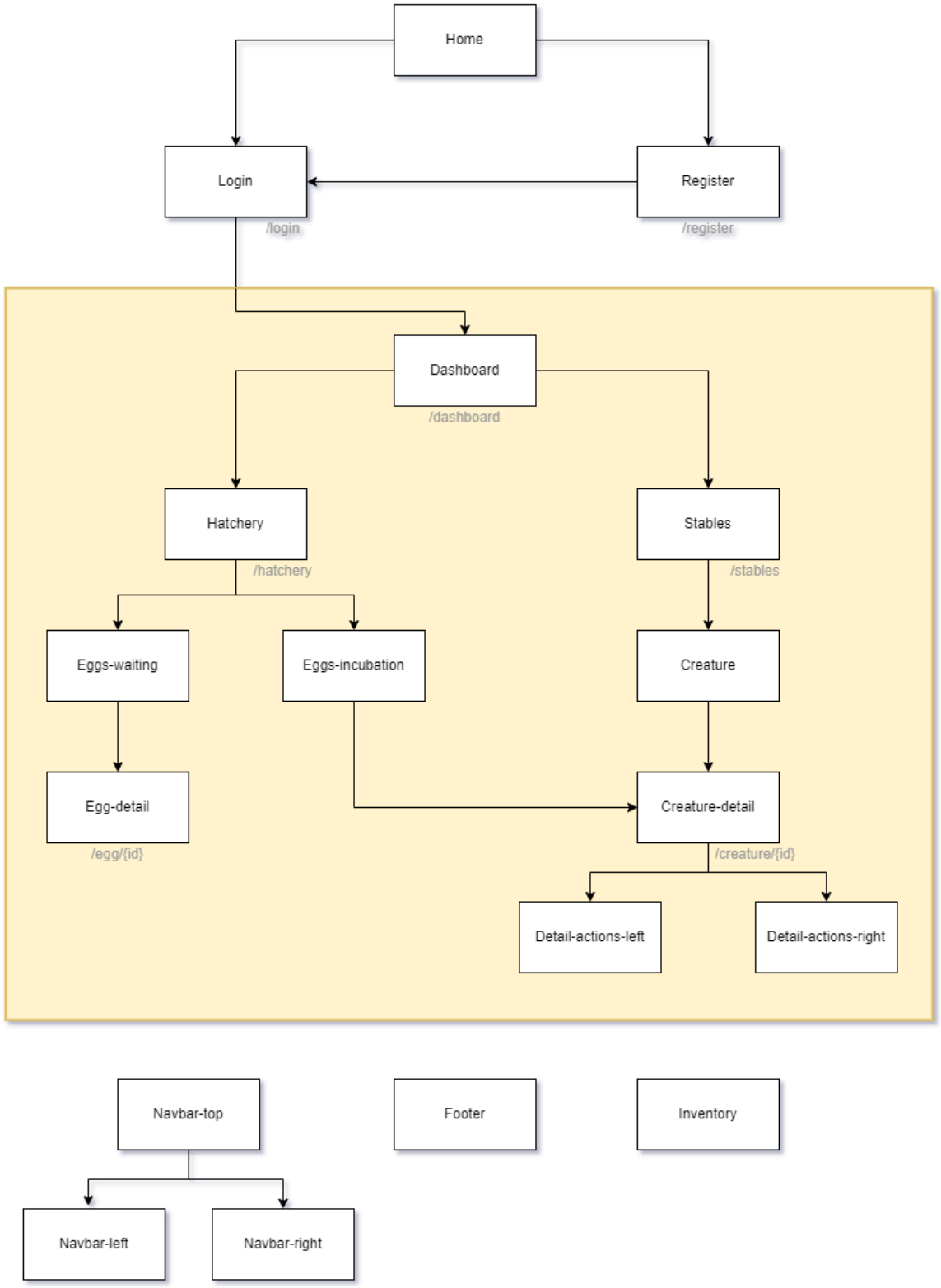
What Is a Framework? (Definition and Types of Frameworks). In: *Indeed* [online]. 2023 [cit. 2023-04-12]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-a-framework

What is Angular?. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2022-11-09]. Dostupné z: https://angular.io/guide/what-is-angular

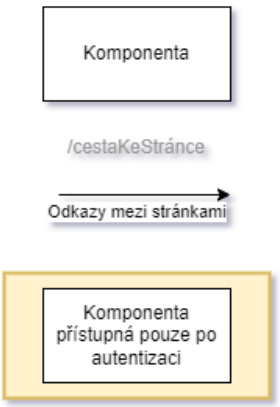
Workspace npm dependencies. In: *Angular.io* [online]. Mountain View, California: Google, 2022 [cit. 2023-03-04]. Dostupné z: https://angular.io/guide/npm-packages

SEZNAM OBRÁZKŮ, TABULEK, GRAFŮ A DIAGRAMŮ

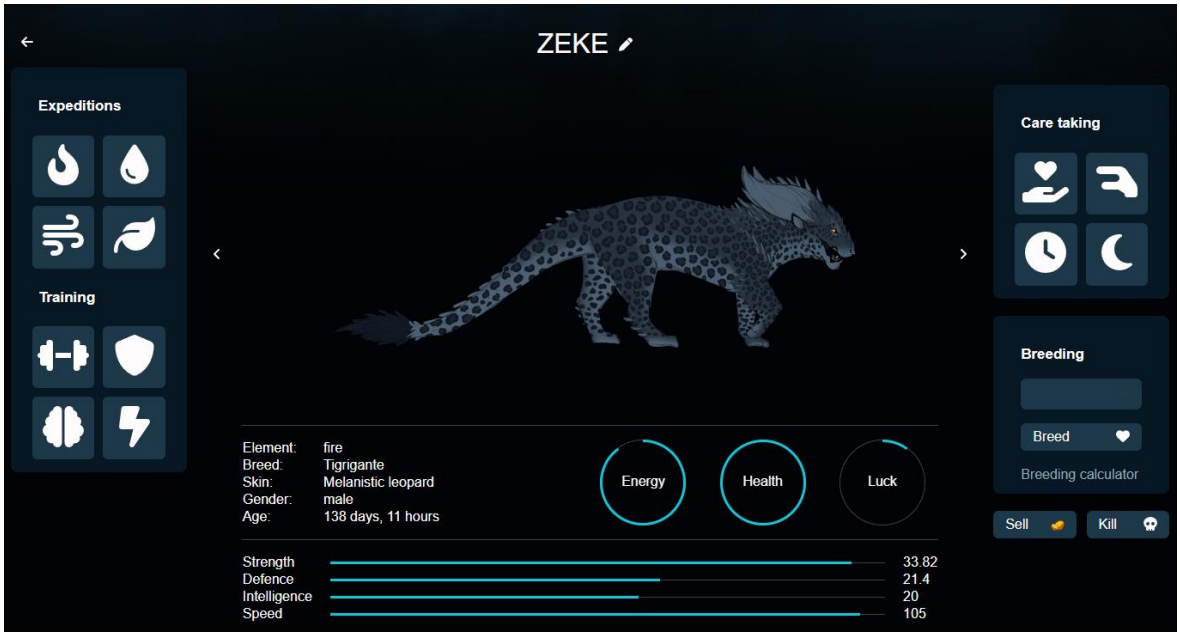
PŘÍLOHY



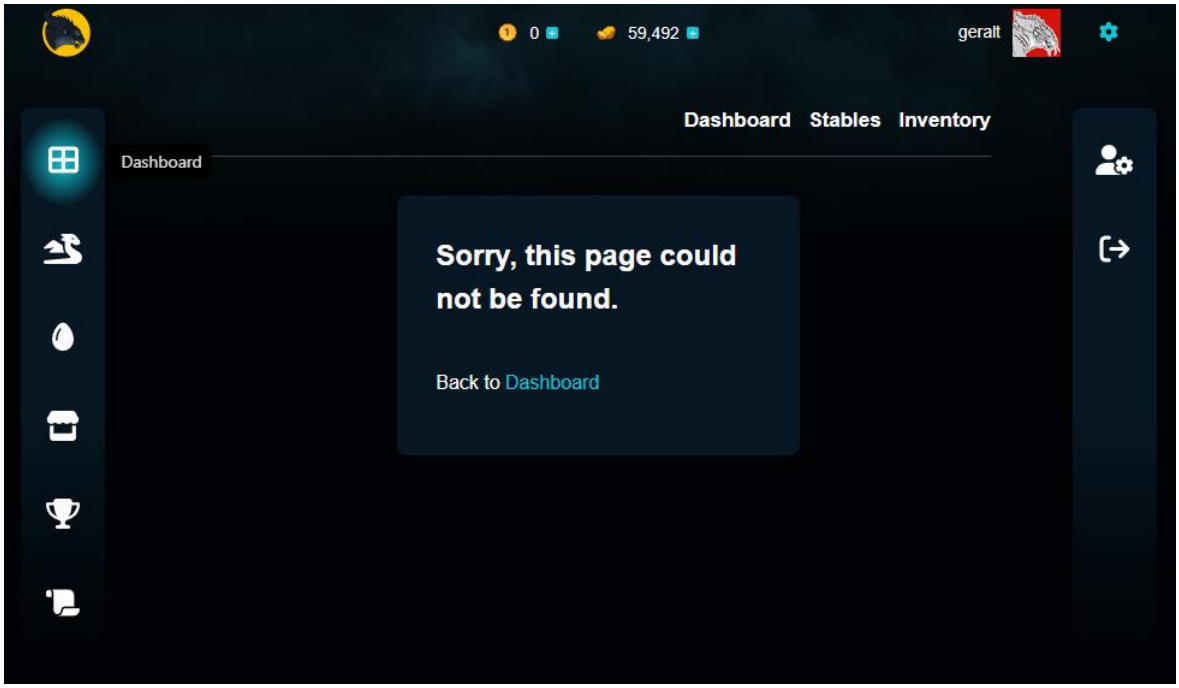
Příloha 1 Diagram rozdělení aplikace na komponenty a stránky (zdroj: vlastní)



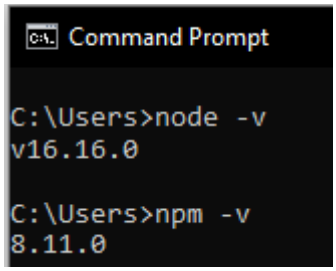
Příloha 2 Legenda k diagramu v Příloze č. 1 (zdroj: vlastní)



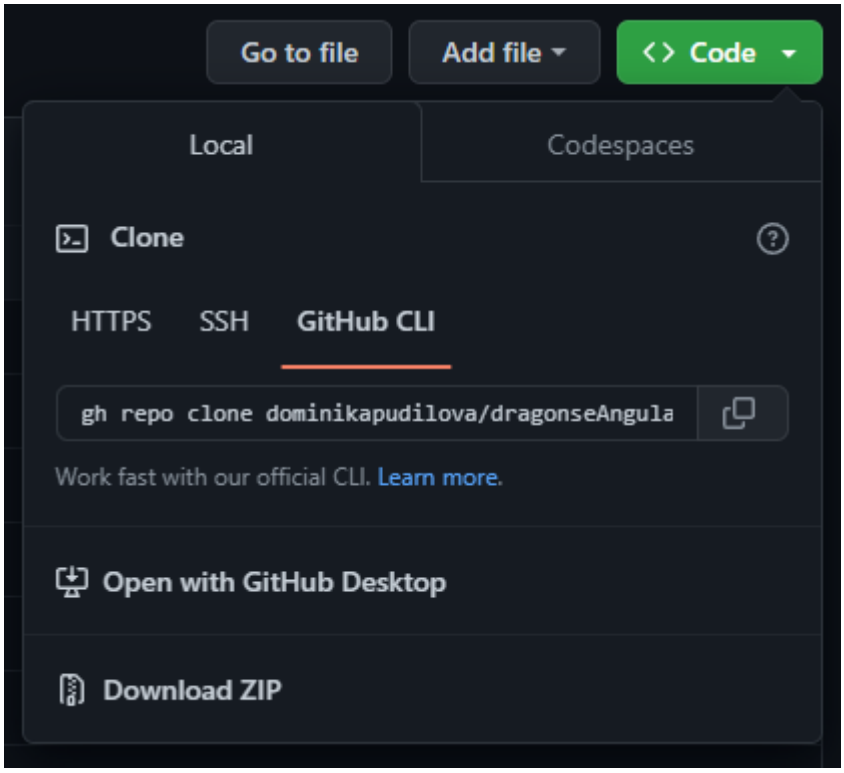
Příloha 3 Creature komponenta v běžící aplikaci (zdroj: vlastní)



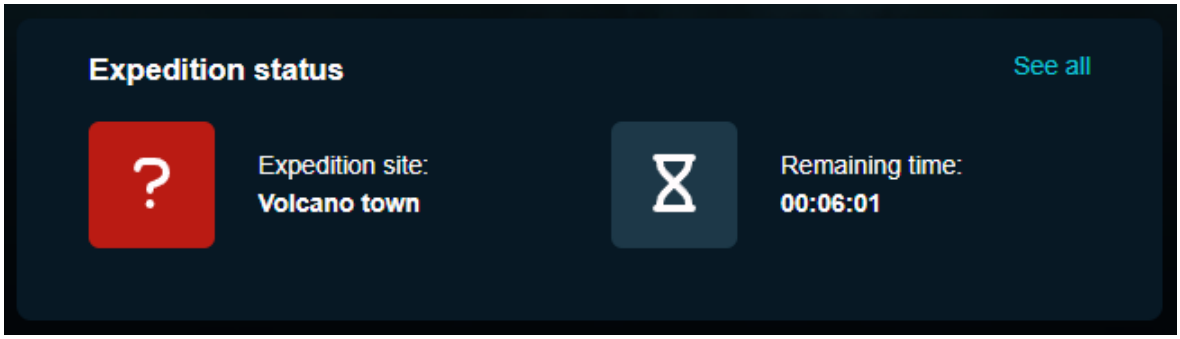
Příloha 4 Rozložení navigačních lišt na stránce. Na první položku Dashboard je najeto myší (zdroj: vlastní)



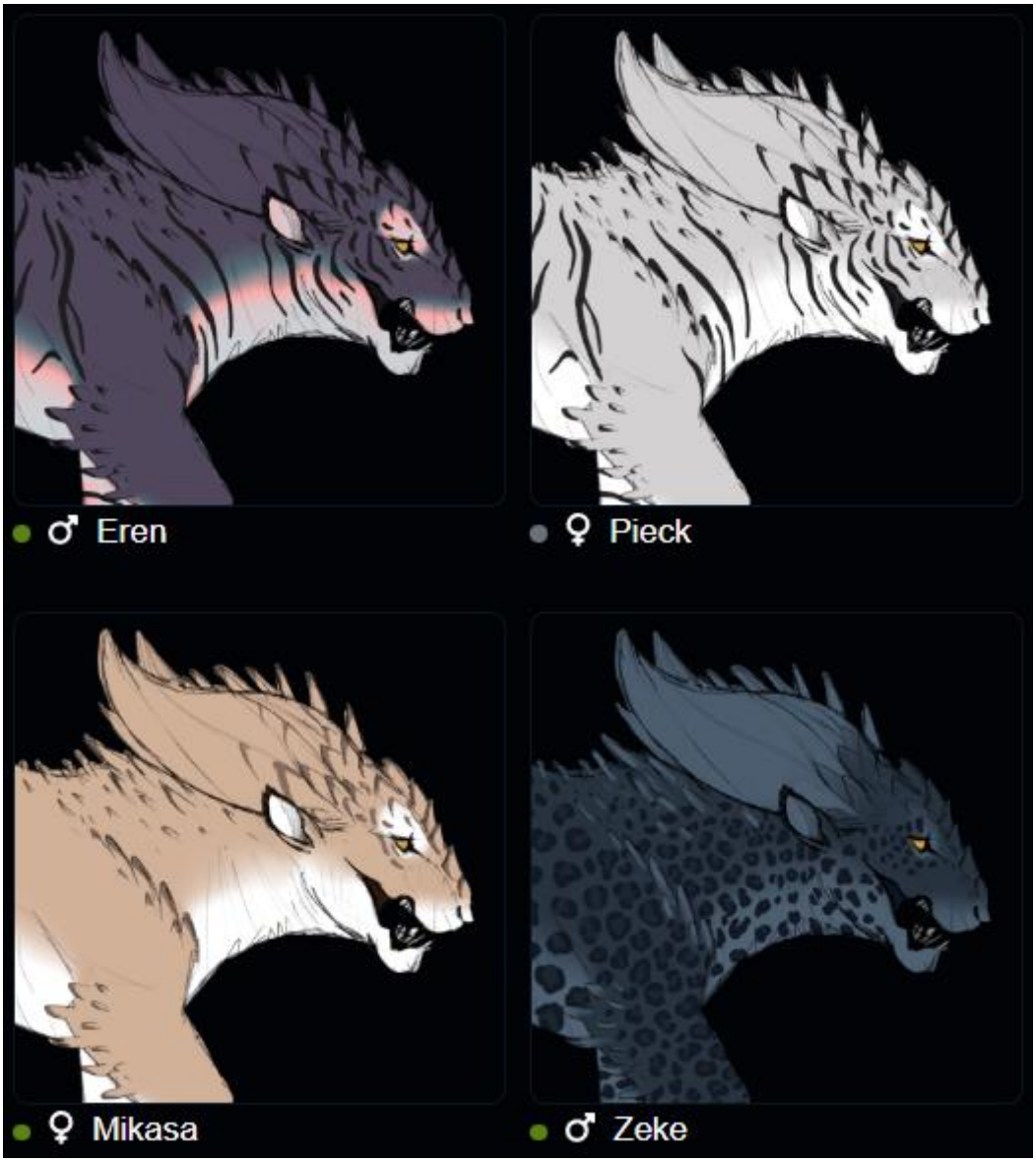
Příloha 5 Dotazy na verze programů Node.js a npm v příkazovém řádku (zdroj: vlastní)



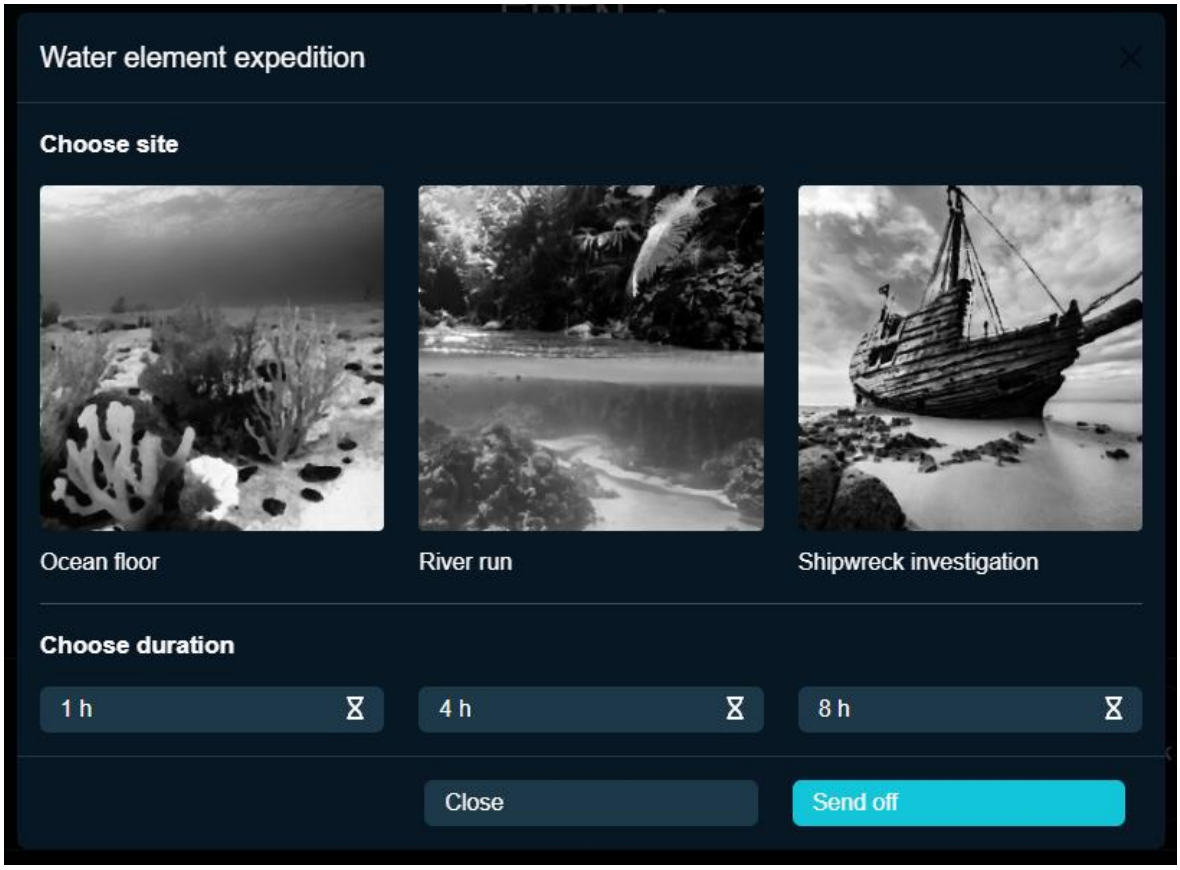
Příloha 6 Možnosti stažení zdrojového kódu aplikace na portálu GitHub (zdroj: vlastní)



Příloha 7 Komponenta ExpeditionModule v běžící aplikaci (zdroj: vlastní)



Příloha 8 Několik komponent CreatureCard v běžící aplikaci (zdroj: vlastní)



Příloha 9 Ukázka hotového modálního okna připraveného pro výběr expedice (zdroj: vlastní)

Příloha 10 Ukázka jednoho ze způsobů, jak může vypadat vyobrazení jména a formuláře pro přejmenování (zdroj: vlastní)