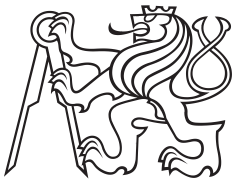
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Extreme Startup

Radim Sückr

Vedoucí: Ing. Petr Aubrecht, Ph.D.
Obor: Softwarové inženýrství a technologie
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sückr** Jméno: **Radim** Osobní číslo: **474609**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Extreme Startup

Název bakalářské práce anglicky:

Extreme Startup

Pokyny pro vypracování:

Soutěž v programování je velmi zábavná činnost, která dovoluje programátorovi otestovat svoje schopnosti a eventuelně poměřit síly mezi juniorními a seniorními členy týmu. Problémem je systém, který výsledky vyhodnocuje a počítá body. Jedním z těchto systémů je tzv. Extreme startup, jehož myšlenka je zadání úlohy a její co nejrychlejší vyřešení. Systém rozesílá dotazy přes HTTP a počítá body na základě správnosti. Řešení je možné vytvářet v jakémkoliv jazyku. Systém má obvykle více úrovní se zvyšující se obtížností úloh.
Výsledný systém má umět nejprve v pomalém tempu posílat úlohy, dokud nezačne řešitel odpovídat správně. Pak by se měla frekvence zvýšit a v případně samých správných odpovědí by se mělo tempo zvýšit maximálně. Pokud jsou některé odpovědi špatně, tempo se sníží, v opačném případě se automaticky přepne na vyšší úroveň, na složitější dotazy.
Příklady typů dotazů jsou: součet dvou čísel, faktoriál, minimální číslo ze seznamu, součet seznamu čísel, vyhodnocení číselného výrazu ap.
1. Zanalyzujte a popište aktuální stav software pro pořádání programátorských soutěží. Dvě možnosti jsou v doporučené literatuře.
2. Vyberte si formu, která Vám připadá nejzábavnější a navrhněte architekturu, která dovolí: a) Snadnou rozšiřitelnost do budoucna, b) Snadné zadávání nových úkolů, c) Přehlednou reprezentaci výsledků, ideálně stránku s rychlou odezvou a např. grafem získaných bodů jednotlivých týmů.
3. Implementaci vyzkoušejte.
4. Řešení implementujte v technologii JakartaEE.
5. Implementaci vyzkoušejte.

Seznam doporučené literatury:

[1] The Jakarta EE 8 Tutorial: https://eclipse-ee4j.github.io/jakartaee-tutorial/toc.html
[2] Jakarta EE Cookbook - Second Edition, https://www.packtpub.com/programming/jakarta-ee-cookbook-second-edition
[3] Patterns of Enterprise Application Architecture — Martin Fowler
[4] Existující řešení Extreme startup: https://github.com/rchatley/extreme_startup
[5] Existující řešení Extreme carpaccio: https://github.com/dlresende/extreme-carpaccio

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Aubrecht, Ph.D. katedra počítačů FEL

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

Datum zadání bakalářské práce: **07.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

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

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji své přítelkyni, rodině a přátelům za podporu a pochopení nejem během této práce, ale během celého mého dosavadního studia. Zároveň děkuji vedoucímu mé práce Ing. Petru Aubrechtovi, Ph.D. za pečlivý dohled nad mou prací a za věcné a cenné konzultace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

Abstrakt

Cílem tohoto projektu je vytvoření aplikace pro snadnou organizaci soutěží v programování. Aplikace jako taková poskytuje rozhraní pro komunikaci s dodatečným serverem, který po definovaném rozhraní poskytuje hernímu serveru otázky. Herní server následně tyto otázky rozesílá účastníkům soutěže, kteří za jejich zodpovězení dostávají buď kladné nebo záporné body.

Hlavní motivací pro vývoj této aplikace je nedostatečná flexibilita a rozmanitost již existujících řešení, jejichž rozšiřování závisí na specifických znalostech technologií, pomocí kterých jsou vytvořené, a tím na uživatele značně zvyšují nároky pro jejich nasazení a rozšíření.

Klíčová slova: soutěž v programování, Java, Jakarta EE

Vedoucí: Ing. Petr Aubrecht, Ph.D.

Abstract

The aim of this project is to create an application for easy organization of programming competitions. As such, the application provides an interface for communication with an add-on server, which provides questions to the game server over a defined interface. The game server then sends these questions to the contest participants, who receive either positive or negative points for answering them.

The main motivation for the development of this application is the lack of flexibility and diversity of existing solutions, the extension of which depends on specific knowledge of the technologies they are built with, and thus greatly increases the demands on the user for their deployment and extension.

Keywords: programming competition, Java, Jakarta EE

Obsah

1 Analýza **1**
1.1 Úvod 1
1.2 Existující řešení 1
1.2.1 Extreme Startup 2
1.2.2 Extreme Carpaccio 2
1.2.3 Slabiny existujících řešení 3
1.3 Požadavky 3
1.3.1 Snadné použití 3
1.3.2 Rozšiřitelnost 4
1.4 Uživatelé aplikace 4

2 Návrh **7**
2.1 Rozšíření nad rámec zadání 7
2.1.1 Otázkové servery 7
2.1.2 Komunikace pomocí SSE 8
2.2 Použité technologie 8
2.2.1 Jakarta EE 9
2.2.2 Payara Micro 9
2.2.3 Docker 10
2.2.4 PostgreSQL 11

3 Implementace **13**
3.1 Základní popis 13
3.2 Datový model 13
3.3 API aplikace 14
3.3.1 Otázkové API 15
3.3.2 Přehled HTTP endpointů otázkového API 17
3.4 Komunikační protokol 17
3.4.1 HTTP API 18
3.4.2 SSE API 18
3.4.3 Příjem odpovědí účastníků . . 20
3.5 Herní smyčka 21
3.5.1 Ovládání herní smyčky 21
3.5.2 Průběh herní smyčky 23
3.6 Struktura balíčků aplikace 23
3.7 UI aplikace 24
3.8 Proces nasazení 27
3.8.1 Nasazení pomocí Dockeru a Docker Compose 27
3.8.2 Nasazení mimo Docker 29

4 Testování **31**
4.1 Strategie unit testování 31
4.1.1 Metrika code coverage 31
4.1.2 GitLab CI 32
4.2 Manuální testování 32
4.2.1 Registrační formulář 32
4.2.2 Integrace otázkového serveru 33
4.3 Ukázkové implementace otázkového serveru a klientů 33
4.3.1 Otázkový server 33
4.3.2 HTTP klient 34
4.3.3 SSE klient 34

5 Závěr **35**
5.1 Výstupy projektu 35
5.2 Shrnutí 35
5.3 Možnosti pro další funkce 36
5.4 Zdrojové kódy 36

Literatura **37**

Slovník **39**

Obrázky

1.1 Use case diagram aktérů 5

2.1 Náhled komponent aplikace 7
2.2 Diagram nasazení Payara Micro aplikace pomocí Dockeru 10

3.1 Diagram tříd datového modelu . 14
3.2 Diagram API aplikace 15
3.3 Sekvenční diagram SSE komunikace 20
3.4 Diagram aktivit herní smyčky . . 21
3.5 Sekvenční diagram znázorňující komunikaci serveru s klienty 22
3.6 Stromová struktura balíčků 24
3.7 Registrační formulář 24
3.8 Administrativní stránka 25
3.9 Hlavní stránka s přehledem 26
3.10 Diagram nasazení 28

Ukázky kódu

3.1 Metadata o účastníkovi . . . 15
3.2 Otázka se statickými odpověďmi 16
3.3 Otázka s dynamickou odpovědí 16
3.4 Tělo přeposílané odpovědi . 17
3.5 Ukázka SSE eventů 19
3.6 Spuštění aplikace přes Docker Compose 27

Kapitola 1
Analýza

1.1 Úvod

Programátoři často rádi řeší složité problémy a velmi rádi mezi sebou soutěží, což dokládá existence populárních soutěží jako *Google's Coding Competition*[1] nebo *Facebook Hacker Cup*[2] a specializovaných knih na toto téma, jako je například *Competitive Programmer's Handbook* [12]. Není překvapením, že aplikace pro pořádaní vlastních takových soutěží, i v menším měřítku jednotek lidí nebo menších týmů, je žádaným a zajímavým projektem. Poptávka po snadno použitelné a rozšiřitelné aplikaci přesně pro tyto účely je motivací mé bakalářské práce, která implementuje hru Extreme Startup [2].

Účast v takové soutěži je dobrá nejen pro zábavu a procvičení technických dovedností a znalostí účastníků, nebo může také velmi dobře posloužit jako menší teambuldingová akce pro týmy ve společnostech a navázat nebo prohloubit vztahy mezi členy týmu. Navíc je možné pro různé účely zvolit různé typy takových her a soutěží. Hra typu Extreme Startup je zaměřená na jednotlivce nebo velmi malé týmy vývojařů, kteří se snaží co nejrychleji dodat řešení na jim předkládané problémy. Extreme Startup maximálně odměňuje správná řešení dodaná co nejrychleji. Naopak hra Extreme Carpaccio [13] cílí a svou podstatou motivuje k utvoření týmů namísto účasti jednotlivců, jelikož v této hře je kladen větší důraz na pochopení předloženého problému.

1.2 Existující řešení

Z existujících řešení známe dvě opensource implementace her Extreme Startup a Extreme Carpaccio, kterými se v této práci zabývám. Soutěže *Google's Coding Competition* a *Facebook Hacker Cup* se zaměřují na jiný druh hry a navíc neposkytují svůj kód veřejně. Ačkoliv jsou to tedy také soutěže v programování, v této práci se jimi ani dalšími podobnými soutěžemi nezabývám.

[1] `https://codingcompetitions.withgoogle.com/`
[2] `https://www.facebook.com/codingcompetitions/hacker-cup/`

1.2.1 Extreme Startup

Účastníkům hry obvykle zůstává celkové zadání utajené a nedílnou součástí jejich práce je také přijít na to, co přesně je problém, který po nich zadavatel úloh chce vyřešit. Obsahem otázek může být cokoliv a každý účastník obdrží své otázky nezávisle na ostatních. Jak již název Extreme Startup napovídá, tato hra se zaměřuje čistě na co nejrychlejší vývoj aplikace.

V původním provedení cílí hra zejména na vývojáře pracující přímo s kódem, jelikož pro plánování postupu nenechává přílíš prostoru kvůli častým periodickým úkolům. Vzhledem k tomu, že vítězí ten, kdo při ukončení soutěže získá nejvíce bodů, záleží na pojetí hry jednotlivými účastníky a jejich přístupu k výslednému kódu.

Rychlé iterace nad změnami lze vzít jako dobrou příležitost k procvičení správné architektury aplikace a čisté organizaci kódu od samého začátku a je ideální pro vyzkoušení praktik jako je TDD, párové programování nebo dokonce XP. Zároveň ale mohou účastníci zvolit taktiku co nejrychlejších ale efektivních řešení a maximalizovat tak své bodové zisky i za cenu méně kvalitní aplikace.

Před začátkem hry je na organizátorovi, aby zvolil obsah (otázky) hry a pravidla. Například maximální povolenou velikost týmů (zpravidla 1 až 3), bodové zisky, penalizace atd. Zároveň by měl organizátor samozřejmě předem i připravit síť, na které bude hra probíhat, aby neměli účastníci problémy s připojením k serveru a registrací. Příprava sítě je mimo rozsah problematiky řešené hrou.

1.2.2 Extreme Carpaccio

Obsahem zadání jsou data o objednávce v nějakém fiktivním obchodě. Účastníci musí přijít na správný způsob, jak spočítat celkovou cenu objednávky a odpověď opět odeslat zpět. Za správnou odpověď dostanou body v celkové ceně objednávky a za každou špatnou odpověď jsou jim strženy body v polovině ceny objednávky.

Extreme Carpaccio je zaměřené nejen na stránku implementace projektu, ale zapojuje i část produktového vývoje a plánování. Největší rozdíl oproti hře Extreme Startup tkví v tom, že zadání původní hry Extreme Carpaccio je jedno a je dopředu známé (nebo alespoň jeho zásadní část v případě úpravy organizátorem) od začátku hry. Zadání je schválně tak velké, aby se nedalo stihnout celé, a tím motivovalo účastníky k zamyšlení se nad problematikou a prioritou jednotlivých úkolů před začátkem samotné implementace.

Na účastnících hry je potom rozdělit zadání na menší podproblémy a z podproblémů vytvořit a prioritizovat úkoly, jejichž postupným řešením se nejsnáze dojde k cíli a tedy k hotové aplikaci. Proto je v této hře významné i plánování a projektový manažer nebo analytik mohou být při hře cennými členy týmu.

Zadání jednotlivých úkolů opět rozesílá server provozovaný organizátorem hry. Účastník zadání rozesílá pomocí HTTP.

1.2.3 Slabiny existujících řešení

Největší slabiny existujících řešení tkví v možnosti jejich úprav a rozšíření o nové úlohy a funkce. Extreme Startup je napsaný v jazyku Ruby[3], což znamená, že přímo vyžaduje znalost Ruby pro své úpravy či prostou přípravu soutěžních úloh. Nepatrnou výhodou je fakt, že Extreme Startup podporuje Docker[4] a poskytuje připravený Dockerfile[5]. To značně usnadňuje sestavení a spuštění aplikace bez nutnosti instalovat Ruby do systému.

Naproti tomu existuje hra Extreme Carpaccio, která je naopak napsaná v JavaScriptu[6] na platformě Node.js[7]. Aplikace navíc nepodporuje Docker a tedy ani neposkytuje Dockerfile, který by zjednodušoval a unifikoval proces sestavení a nasazení. Kromě technologických rozdílů implementují obě řešení zcela jiný druh hry.

Společný mají obě hry průběh komunikace s účastníky. Komunikace probíhá přes HTTP, pomocí kterého server rozesílá periodicky všem účastníkům úlohy, které mají jejich aplikace vyřešit a opět pomocí HTTP odeslat zpět odpověď. Za správné odpovědi získavají účastníci kladné body, za špatné odpovědi získávají body záporné. Výhercem soutěže se stává účastník s nejvyšším počtem bodů v době ukončení hry.

1.3 Požadavky

Dva nejdůležitější a konkrétní požadavky pro mou aplikaci cílí zejména na organizátory her a přímo vychází ze slabin existujících řešení. Snadné použití, nasazení a rozšiřitelnost jsou pro organizátory velmi důležité, aby se příprava hry co nejvíce omezila na časovou organizaci a vytvoření otázek namísto seznamování se s konkrétními technologiemi pro spuštění projektu.

1.3.1 Snadné použití

Původní a zároveň největší motivací tohoto projektu je vytvořit aplikaci, která bude snadno a rychle umožňovat pořádání programovacích soutěží. Hlavní způsob, kterým toho tento projekt dosahuje, je nezávislost na konkrétní technologii díky minimalistickému komunikačnímu protokolu přes HTTP. Tímto způsobem může být soutěž připravena v jakékoliv technologii, která podporuje komunikaci pomocí HTTP a JSON, ačkoliv jádro aplikace je vytvořené pomocí Jakarta EE [5] ve verzi 8 v programovacím jazyce Java [18] ve verzi 11.

Ke snadné přípravě a spuštění hry dále přispívá příprava aplikace na provoz na kontejnerizační platformě Docker.

[3] `https://www.ruby-lang.org/en/`
[4] `https://www.docker.com/`
[5] `https://docs.docker.com/engine/reference/builder/`
[6] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`
[7] `https://nodejs.org/en/`

1.3.2 Rozšiřitelnost

Dalším cílem projektu je udržovat kód jádra aplikace snadno čitelný a ve vysoké kvalitě, aby případní noví přispěvatelé neměli problémy s pochopením stávajícího kódu a mohli snadno a rychle přispět svými vylepšeními nebo opravami.

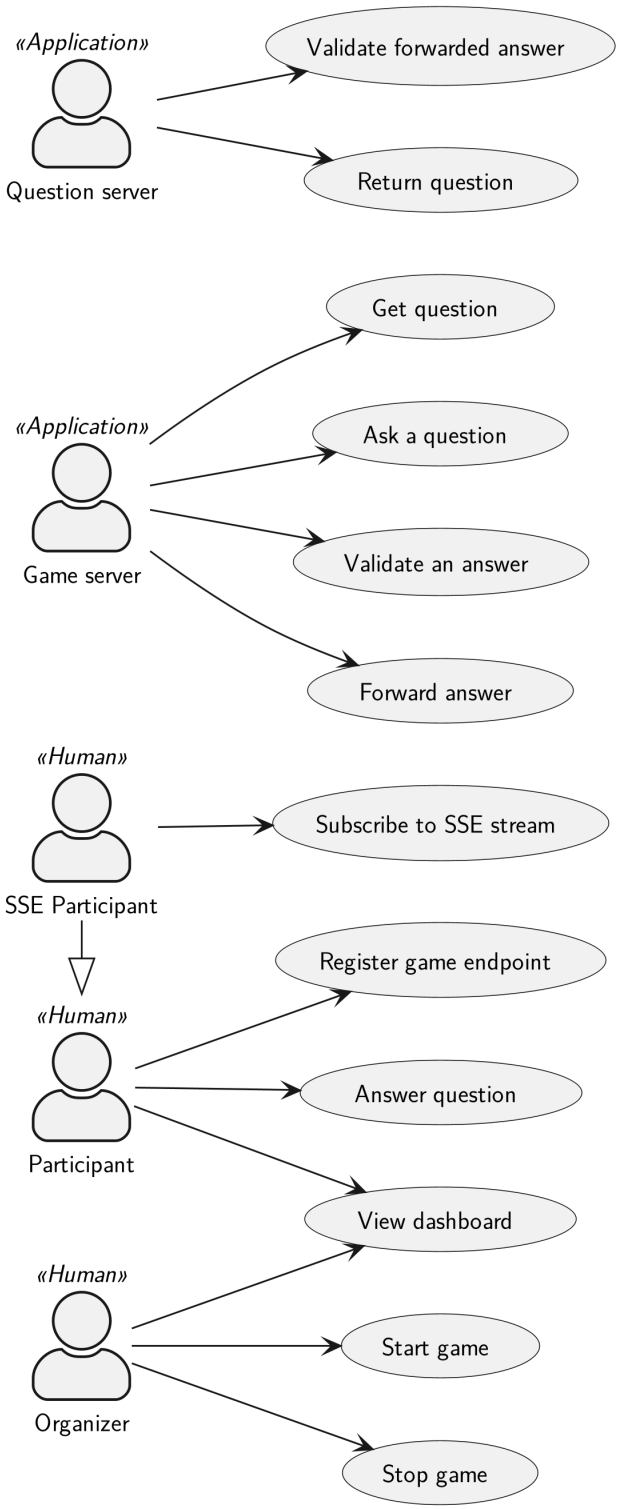
Rozšiřitelnost podporují unit testy kritických částí aplikace a OpenAPI[8] specifikace.

1.4 Uživatelé aplikace

V aplikaci se vyskytuje pět druhů aktérů. Účastník hry (*Participant*), účasník hry využívající SSE [21] (*SSE Participant*), organizátor hry (*Organizer*), herní server (*Game server*) a otázkový server (*Question server*). Uživatel *SSE Participant* rozšiřuje aktéra *Participant*. Pro aktéra *Participant* je stěžejní zodpovídání otázek, které obdrží. Nejzásadnějšími aktivitami *Organizer* jsou spouštění a pozastavení herní smyčky.

Konkrétní případy užití jednotlivých aktérů jsou znázorněné use case diagramem na obrázku 1.1.

[8] `https://spec.openapis.org/oas/v3.0.3`



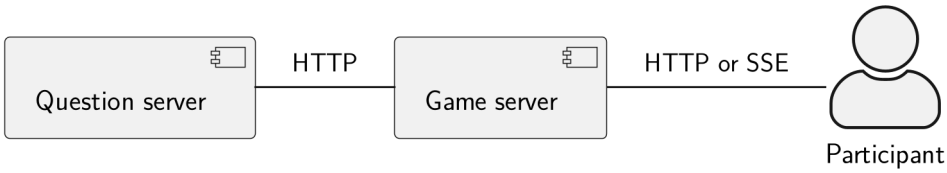
Obrázek 1.1: Use case diagram aktérů

Kapitola 2
Návrh

2.1 Rozšíření nad rámec zadání

Během prvotní analýzy vznikla myšlenka nejít ve stopách podobných implementací a tedy vytvořit vendor lock-in [3] na programovací jazyk Java a framework Jakarta EE tím, že by byla aplikace postavena čistě nad JVM [20] a otázky by se do ní přidávaly jako rozšiřující knihovny JAR [19].

Namísto toho byl zvolen přístup komunikace přes HTTP s další aplikací, takzvaným *otázkovým serverem*, který je využíván jako zdroj otázek pro hru.



Obrázek 2.1: Náhled komponent aplikace

2.1.1 Otázkové servery

Implementace otázkových serverů přináší obrovskou flexibilitu pro organizátory hry při přípravě otázek. Odstraňuje nutnost znalosti programovacího jazyka Java nebo frameworku Jakarta EE a umožňuje autorovi otázek využít jemu nejznámější či nejpříjemnější technologii a poskytnout mu tak co nejvyšší efektivitu.

Jediné technologické nároky na implementaci otázkového serveru jsou podpora komunikace pomocí HTTP s využitím formátu JSON. Autor otázkového serveru pak už jen musí naimplementovat tři jednoduché HTTP endpointy podle OpenAPI specifikace.

Repozitář projektu obsahuje v adresáři `example/python` ukázkové implementace otázkového serveru a klientů jak pro klasickou request-response komunikaci, tak pro SSE. Jak server, tak klienti jsou pro demonstraci rozšiřitelnosti naprogramovány v jazyce Python[1] pomocí frameworku Flask[2].

[1] `https://www.python.org/`
[2] `https://flask.palletsprojects.com/en/2.1.x/`

2.1.2 Komunikace pomocí SSE

Doba COVID-19 pandemie přinesla zcela nové požadavky na online komunikaci a zásadně omezila možnosti osobního setkávání. Pro tento projekt to přináší další výzvu v podobě podpory pořádání soutěže vzdáleně mimo LAN prostředí.

Bez podpory SSE je totiž aplikace připravena pro běh na LAN, což ale právě kvůli COVID-19 či jiným restriktivním okolnostem nemusí být v daném momentu možné uskutečnit. I před pandemií by ale bylo obtížné uspořádat tuto hru například pro dislokované globální týmy. Organizátor by musel zajistit VPN, aby mohla komunikace probíhat obousměrně, čemuž by jistě někde po cestě internetem mohly bránit mimo jiné například NAT gateways.

Navíc zajištění VPN infrastruktury s sebou samo přináší vlastní problémy, které je nutné vyřešit. Ať už se jedná o zvolení vhodné VPN technologie, zajištění bezproblémové funkčnosti připojení pro všechny účastníky napříč různými operačními systémy, implementacemi klientských software či jen různými verzemi stejné implementace, které nemusí být kompatibilní.

SSE bylo zvoleno namísto WebSockets [14] zejména kvůli velmi jednoduchému komunikačnímu protokolu, který funguje v čistě textové podobě nad obyčejným HTTP protokolem. WebSockets jsou oproti tomu zcela nový vlastní TCP protokol.

Komunikace pomocí WebSockets navíc přináší potenciálně další svá vlastní úskalí jako může být například blokování nešifrované komunikace na různých proxy serverech [8]. Zajištění šifrované komunikace je vedle VPN infrastruktury další velká překážka, kterou by musel organizátor soutěže vyřešit. Ať už se jedná o obstarání domény a pro ni platného veřejného certifikátu od uznávané autority, či o vygenerování vlastního certifikátu a jeho následnou distribuci účastníkům.

2.2 Použité technologie

Dle požadavků v zadání bakalářské práce je aplikace implementovaná v programovacím jazyce Java s pomocí balíku knihoven Jakarta EE. Konkrétní použitá implementace je Payara Server. Pro ukládání dat používá aplikace technologii JPA a databázový server PostgreSQL.

K co největšímu usnadnění implementace UI se využívá knihovna PrimeFaces[3], jelikož UI není stěžejním předmětem této aplikace a má ryze informativní činnost.

Pro sestavení aplikace je použit systém Maven[4]. Nasazení a spuštění aplikace zjednodušuje použití Dockeru, pro který má v repozitáři projektu připravený Dockerfile a YAML konfiguraci pro Docker Compose[5].

[3] `https://www.primefaces.org/`
[4] `https://maven.apache.org/`
[5] `https://docs.docker.com/compose/`

2.2.1 Jakarta EE

Jakarta EE je sada specifikací, které rozšiřují Java SE [17] a jež vznikla přejmenováním specifikace Java EE 8. To znamená, že Java EE 8 a Jakarta EE 8 jsou vzájemně kompatibilní až na přejmenování kořenových balíčků z `javax` na `jakarta`. Zároveň s přejmenováním přešla tato technologie od firmy Oracle pod správu Eclipse Foundation, která nyní obstarává další vývoj této specifikace. Populární, avšak nekompatibilní alternativou Jakarta EE, je Spring Framework[6].

Jakarta EE sama o sobě obsahuje pouze definice Java rozhraní a potřebuje tedy implementaci těchto rozhraní. Takové implementace se nazývají aplikační servery a existuje jich celá řada. Například klasické aplikační servery GlassFish, WildFly, Open Liberty a IBM WebSphere nebo mladší Apache TomEE a Payara[7].

Sada specifikací, které Jakarta EE obsahuje, je obrovská a pokrývá nejzákladnější nástroje pro vývoj webových aplikací jako například Jakarta Servlet nebo JSF, tak i pokročilé nástroje JPA, JTA, EJB, CDI nebo JMS a mnoho dalších. Dále existuje specifikace MicroProfile [1], která přináší podporu pro microservice architekturu pro Jakarta EE aplikace.

Jakarta EE byla pro tuto bakalářskou práci zvolena již v zadání. Důvod použití je dlouhá historie a zejména stabilní rozhraní, která svou signaturu dodržují dlouhá léta. Udržování takové aplikace je tedy značně usnadněno. Aplikace používá ještě specifikaci Java EE 8, jelikož během vývoje aplikace ještě Payara Server nepodporoval specifikaci Jakarta EE 8.

2.2.2 Payara Micro

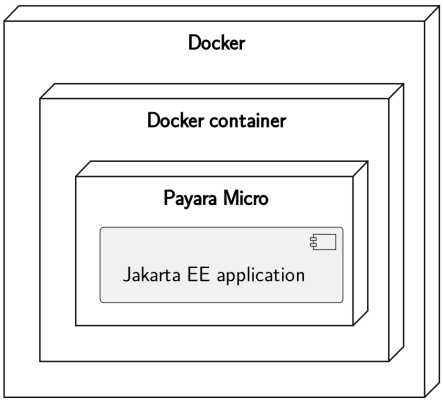
Aplikační server Payara vychází ve dvou hlavních variantách. Jedna je klasický Payara Server, druhá je Payara Micro, která je uzpůsobená pro provoz v kontejnerech a je obdobou Spring Boot.

Verzi Payara Micro jsem vybral, protože primární a doporučovaný způsob spouštění této bakalářské práce je pomocí Dockeru a Docker Compose. Payara Micro je ideální kandidát díky tomu, že například neobsahuje webové rozhraní pro administraci aplikačního serveru, velmi snadno se dá předkonfigurovat pomocí konfiguračních souborů nebo přepínačů příkazové řádky a kromě snadné kontejnerizace lze Payara Micro provozovat i jako samostatnou JAR aplikaci, jelikož výsledný JAR je takzvaně self-contained a nepotřebuje spuštěnou instanci klasického Payara Server, do kterého by se musela teprve nasadit a tudíž ji lze spustit přímo bez dalších instalačních kroků.

Nasazení Payara Micro aplikace v Docker kontejneru je znázorněno na obrázku 2.2.

Payara Micro tedy přináší spoustu výhod pro provoz v Docker kontejnerech a zároveň zachovává možnost standardního spuštění JAR pomocí hostitelské JVM.

[6] `https://spring.io/projects/spring-framework`
[7] `https://payara.fish`



Obrázek 2.2: Diagram nasazení Payara Micro aplikace pomocí Dockeru

2.2.3 Docker

Docker je platforma pro kontejnerizaci procesů, která podporuje různé operační systémy. Mezi hlavní podporované systémy patří Linux, macOS a Windows. Kontejnery se z jistého pohledu chovají jako virtuální stroje provozované například přes LXC[8], VirtualBox[9] nebo VMware[10]. Docker avšak využívají nativní funkce operačního systému Linux[11] cgroups[12] a kernel namespaces[13]. Podobně jako virtuální stroje slouží k izolaci procesů, ale narozdíl od virtuálních strojů mají kontejnery mnohem menší nároky na systémové prostředky a jejich spouštění je velmi rychlé.

To, že Docker využívá funkce Linuxu, přináší otázku, jak funguje na ostatních operačních systémech. Odpověď je, že na Windows i macOS spouští Docker Linuxový virtuální stroj a ten používá pro své fungování a z pohledu spouštěného procesu vše vypadá jako na nativním Linuxovém systému. Na systémech mimo Linux tedy Docker používá i virtualizaci. Docker podporuje i spouštění kontejnerů založených na operačním systému Windows[14], ale ty mají své specifické použití a pro tuto práci nejsou důležité.

Hlavním přínosem Dockeru pro tuto práci je to, že aplikace jde v Dockeru i sestavit a při dnešním rozšíření Dockeru přináší velkému množství uživatelů možnost aplikaci sestavit a spustit bez nutnosti mít v operačním systému nainstalovanou Javu. Na druhou stranu uživatelé, kteří mají Javu a nemají Docker, mohou použít klasický způsob a sestavit si JAR pomocí Payara Micro popsané v sekci 2.2.2.

[8] https://linuxcontainers.org/
[9] https://www.virtualbox.org/
[10] https://www.vmware.com/cz/products/workstation-player.html
[11] https://www.kernel.org/
[12] https://man7.org/linux/man-pages/man7/cgroups.7.html
[13] https://man7.org/linux/man-pages/man7/namespaces.7.html
[14] https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-base-images

2.2.4 PostgreSQL

Při volbě databázového systému byly brány v potaz dva zásadní faktory: zkušenosti s daným RDBMS a jistá univerzálnost použití. Přemýšlel jsem i nad jiným úložištěm než relační databází, ale rozhodl jsem se, že využiji stability a širokých možností JPA a EclipseLink[15], se kterými mám zkušenosti a poskytují velmi dobré a dostatečně univerzální úložiště pro případné rozšiřování.

Volba PostgreSQL[16] vychází rovněž z faktu, že s tímto RDBMS mám největší zkušenosti, jeho konfigurace i používání jsou přímočaré a zároveň je rovněž více než dostačující pro další funkce aplikace nebo větší nasazení. V úvahu připadalo i použití SQLite[17], ale s větším použitím nemám zkušenosti a případné ladění technických nebo výkonostních problémů by byly zbytečným zdržením při práci na aplikaci, jejíž stěžejní práce nespočívá v databázových dotazech.

[15] https://www.eclipse.org/eclipselink/
[16] https://www.postgresql.org/
[17] https://sqlite.org/index.html

Kapitola 3
Implementace

3.1 Základní popis

Vzhledem k tomu, že jedním z největších omezení existujících řešení je závislost na konkrétní technologii i pro prosté přidání nových úloh do aplikace, zvolil jsem pro mou bakalářskou práci jiný přístup.

Rozšiřitelnost samotného jádra aplikace, které obstarává hlavní herní smyčku, je sice napsané v Jakarta EE, nicméně otázky nejsou zadávány pomocí JSON souborů s fixním schématem, ani není třeba programovat rozšiřující pluginy v Javě.

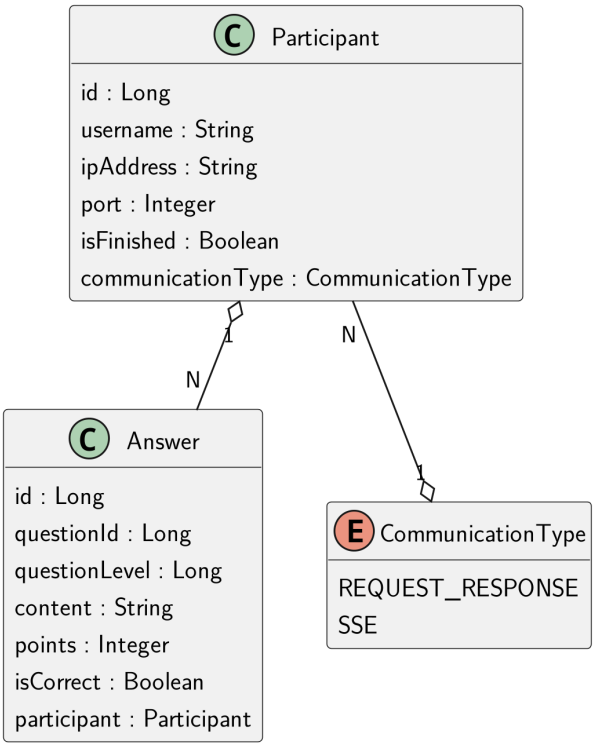
Namísto toho je u projektu definované malé HTTP API pro takzvané otázkové servery. To jsou servery, které implementují API dle OpenAPI specifikace aplikace. Od těchto otázkových serverů pak herní server pomocí HTTP dotazů získává otázky pro účastníky.

Výsledkem tohoto přístupu je, že pro přidání nových úloh nemusí mít organizátor zkušenosti s programováním v Javě, ale může použít nástroje, které mu jsou dobře známy a pouze implementovat pár naprosto jednoduchých HTTP endpointů.

V kořenovém adresáři projektu je adresář `src`, který obsahuje jádro systému v Javě. Zároveň s ním tam lze ale nalézt i adresář `example/python`, kde jsou implementace ukázkového otázkového serveru, který jednak umí fungovat v režimu servírování předdefinovaných statických otázek ze souboru, nebo funguje jako hlavní plnohodnotný otázkový server s vlastní logikou, a k němu jsou dvě korespondující klientské implementace jak pro SSE, tak pro klasické HTTP. Otázkový server i klienti jsou z důvodů jednoduchosti, mých zkušeností a demonstrace multiplatformnosti aplikace naprogramovány v Pythonu s pomocí knihovny Flask.

3.2 Datový model

Aplikace má doslova triviální datový model, který se skládá pouze ze dvou entit a pro aktuální rozsah a potřeby aplikace naprosto dostačuje. Diagram datového modelu je na obrázku 3.1. Díky použití RDBMS PostgreSQL a JPA je případné další rozšíření datového modelu velmi snadné, pokud by to



Obrázek 3.1: Diagram tříd datového modelu

v budoucnu přidávaná funkcionalita vyžadovala.

V současném stavu si aplikace drží pouze informace o účastnících registrovaných do hry, o kterých zná pouze jejich IP adresu a port, na kterém poslouchá klient týmu, jméno účastníka (týmu), druh komunikace s klientem a zda-li klient účastníka úspěšně prošel všechny levely a tím dokončil hru.

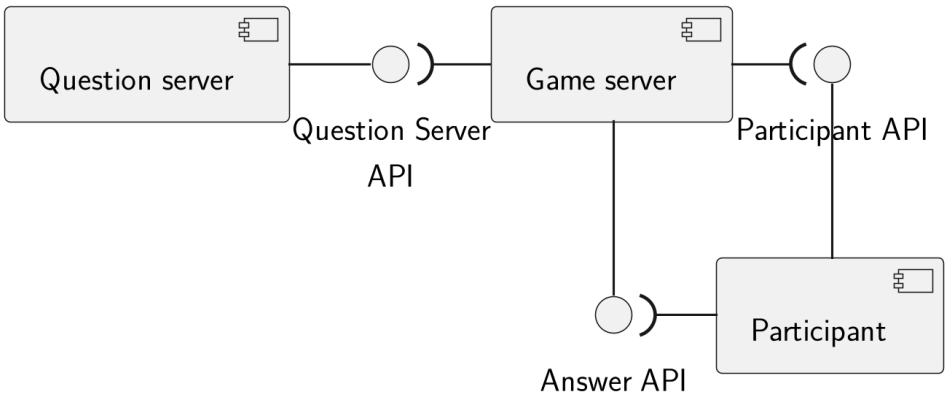
Aplikace si neukládá otázky, jelikož se na ně pomocí ID otázky může kdykoliv doptat otázkového serveru. Namísto toho aplikace udržuje přehled všech odpovědí klientů. Součástí odpovědi je ID otázky, které je neměnné a přiřazuje jej otázkový server. Dále obsahuje zadání otázky, level, do kterého otázka spadá, body, které účastník za odpověď obdržel a zda-li byla odpověď správná. Bodový zisk účastníka je vypočítán jako suma bodových zisků ze všech jeho odpovědí.

Díky metadatům otázky, které odpověď obsahuje, lze zpětně zjistit, zda-li je otázkový server naimplementován korektně (pro stejné ID otázky vrací vždy stejné zadání), nebo zda všechny účastníky hodnotí stejně.

3.3 API aplikace

Aplikace pracuje se třemi různými API. API pro otázkové servery nebo také *otázkové API*, API pro účastníky, které je aktuálně velmi strohé a omezuje se pouze na endpoint pro zasílání odpovědí, a jistou formu API na straně účastníků sloužící pro příjem otázek od herního serveru.

Vztahy mezi jednotlivými API jsou znázorněné na obrázku 3.2.



Obrázek 3.2: Diagram API aplikace

3.3.1 Otázkové API

Otázkové API podporuje dva druhy otázek. S fixní a s dynamickou odpovědí. Otázka s fixní odpovědí může být například kolik je 2+2. Otázka s dynamickou odpovědí může být *co je dnes za den?* Každá otázka má také své *unikátní* ID v poli `id` a pole `level` určuje, do jaké úrovně otázka náleží.

Při dotazu na novou otázku obdrží otázkový server metadata o účastníkovi, na jejichž základě se může rozhodovat, jakou otázku položí a zda-li nenastal vhodný čas posunout účastníka do další úrovně. Průběh hry si tedy může organizátor naprosto přesně naplánovat. Příklad metadat zasílaných otázkovému serveru je v ukázce kódu 3.1.

```
{
    "name": "The Best Team",
    "level": 7,
    "correctAnswersCount": 178,
    "answersCount": 234,
    "correctAnswersInRowCount": 84
}
```

Kód 3.1: Metadata o účastníkovi

Odpovědi na otázky s fixní odpovědí kontroluje už herní server, protože přesný tvar očekávané odpovědi zná spolu s otázkou. Odpovědi na dynamické otázky naopak přeposílá pomocí POST metody dále na endpoint `/answer` otázkového serveru, který jako jejich tazatel zároveň zná správný postup k získání a ověření odpovědi. Na základě ověření správnosti výsledku odpovídá otázkový server hernímu serveru kladně či záporně a ten podle výsledku otázkového serveru přidělí příslušný počet bodů.

Otázky se statickou a dynamickou odpovědí se liší zejména hodnotou atributu `shouldForwardAnswer`, která se v případě statických odpovědí neodesílá

vůbec a v případě dynamických odpovědí má nastavenou hodnotu na `true`. Otázky se statickými odpověďmi pak namísto tohoto pole obsahují pole `expectedAnswers`, což je JSON pole, které obsahuje serializované objekty tak, jak jsou očekávány od účastníků.

Příklad obou možností otázek ve formátu JSON jsou uvedené na ukázkách kódu 3.2 a 3.3.

```
{
    "id": 0,
    "summary": "Is this the best game you've ever played?",
    "expectedAnswers": [
        true
    ],
    "level": 0
}
```

Kód 3.2: Otázka se statickými odpověďmi

```
{
    "id": 1,
    "summary": "What's the sum of today's and yesterday's day
    number?",
    "shouldForwardAnswer": true,
    "level": 1
}
```

Kód 3.3: Otázka s dynamickou odpovědí

Jak naznačuje sekvenční diagram hlavní smyčky na obrázku 3.5, aplikace se postupně ptá na otázky otázkového serveru. Na otázkovém serveru je, aby pro účastníky vracel příslušné otázky.

3.3.2 Přehled HTTP endpointů otázkového API

Zjednodušený popis otázkového API. Detailní OpenAPI specifikace je k dispozici v souboru `docs/g2q/openapi.yaml`.

- `POST /question`
  - Tímto endpointem žádá herní server nové otázky. Jako tělo požadavku jsou přiložena metadata o účastníkovi jako na ukázce kódu 3.1. Otázky vrací s HTTP 200, dokončení hry oznamuje HTTP 204 a HTTP 404 označuje stav, kdy nejsou dostupné žádné otázky.
- `GET /question?questionId=<id>`
  - S HTTP 200 vrací otázku s ID `<id>`. HTTP 404 značí, že otázka s daným ID neexistuje.
- `POST /answer`
  - Slouží k přeposílání odpovědí na otázky s dynamickými odpověďmi. Tělo požadavku je v JSON formátu na ukázce kódu 3.4. HTTP 200 značí správnou odpověď, HTTP 406 značí špatnou odpověď a HTTP 400 značí, že odpověď na tuto otázku není dynamická a nemá se přeposílat.
- `GET /healthcheck`
  - Jednoduchý endpoint, který má vrátit HTTP 200, pokud je server připravený a HTTP 503 pokud ne. Jiné stavové kódy jsou považovány za chybu a hrubé selhání serveru.

Pole `payload` na ukázce kódu 3.4 obsahuje odpověď od účastníka již serializovanou do JSON. Pro její přečtení je tedy nutné deserializovat pole `payload` jako JSON.

```
1 {
2     "questionId": 1337,
3     "payload": "[7, 5, 1998]"
4 }
```

Kód 3.4: Tělo přeposílané odpovědi

3.4 Komunikační protokol

Stejně, jako již existující implementace Extreme Startup a Extreme Carpaccio, i má bakalářská práce používá ke komunikaci protokol HTTP a formát JSON, jelikož spolu s jednoduchým REST-like/RPC API se jedná o zřejmě nejsnažší a nejznámější způsob komunikace v prostředí webových aplikací a služeb.

Všechny komponenty celého systému kolem aplikace mají svá rozhraní, která poskytují, a zároveň klienty pro rozhraní, která konzumují.

Navíc je v repozitáři projektu v adresáři `docs` k celému API dostupná OpenAPI specifikace.

Jelikož aplikace používá jako formát pro výměnu dat JSON, mají veškeré HTTP požadavky, jak ze strany účastníků vůči hernímu serveru, tak ze strany herního serveru vůči otázkovému serveru či účastníkům, hlavičku `Content-Type: application/json`.

3.4.1 HTTP API

Nejzákladnější a výchozí formou komunikace jsou prosté HTTP dotazy. Tento klasický textově založený komunikační protokol má velmi dobrou, až perfektní, podporu pro naprostou většinu dnes používaných programovacích jazyků a technologií.

Čisté HTTP API je použito pro komunikaci mezi herním a otázkovým serverem. Cílem pro toto rozhraní byla co nejsnažší tvorba otázkových serverů pro organizátory. Navíc SSE API by zde neřešilo hlavní problémy, kvůli kterým bylo do aplikace v první řadě přidáno, jelikož herní a otázkový server jsou oba spuštěny na stejném stroji a protože tato aplikace není žádný složitý distribuovaný systém mikroservis.

Tento přístup ale přináší svá jistá úskalí nastíněná v sekci 2.1.2. Je to zejména komunikace se všemi účastníky po WAN. Dá se předpokládat, že pokud budou někteří účastníci soutěžit vzdáleně, nebudou mít zdaleka všichni k dispozici veřejnou IP adresu, ať už verze 4 nebo 6. Pokud by aplikace podporovala pouze čistou HTTP komunikaci, čelili by účastníci a organizátor složité situaci, jak zařídit obousměrnou komunikaci. Bylo by potřeba zařídit nějakou formu VPN, ať už pomocí OpenVPN[1], Wireguard[2] nebo třeba SSH[3] tunelování. V každém případě je to zbytečná a ne zcela triviální překážka v organizování takové soutěže.

3.4.2 SSE API

Abychom se elegantně vyhnuli problémům pořádání hry přes WAN, nabízí se použít některé z modernějších přístupů obousměrné a jednosměrné komunikace, které přinesly aktualizace HTTP specifikace. Konkrétně jsou to technologie WebSockets a o něco novější SSE.

Díky faktu, že herní server nepotřebuje vytvářet nová spojení na herní klienty účastníků, nýbrž otázky zasílá do streamu, který vytváří svým dotazem na herní server účastník, zcela odpadá požadavek na účastníky vlastnit veřejnou IP adresu. Naprosto postačuje, že veřejnou adresu poskytuje libovolnou cestou organizátor hernímu serveru. Tomu se bohužel nevyhneme. Pokud ale organizátor nedisponuje veřejnou adresou například pro své domácí internetové připojení, může si velmi levně pronajmout v libovolném veřejném

[1] `https://openvpn.net/`
[2] `https://www.wireguard.com/`
[3] `https://www.openssh.com/`

cloudu po dobu hry virtuální privátní server a na něm hru bezproblémově spustit.

Další z velkých výhod SSE je, že se narozdíl od WebSockets jedná o čistě textový protokol a jeho použití je tudíž velmi snadné. Navázání spojení probíhá tak, že klient pošle serveru klasický HTTP dotaz a server jednoduše odpoví s hlavičkou `Content-Type: text/event-stream` a *neuzavře* spojení. Tím je navázána jednosměrná komunikace pomocí SSE. Od této chvíle může server bezstarostně streamem posílat klientovi data, aniž by musel mít klient veřejnou IP adresu. Ukázka eventů přijatých od serveru je v kódu 3.5.

Jedna SSE událost může obsahovat více `data` polí, což se využívá například pro víceřádkový JSON dokument. Pokud je třeba přenést binární data, z povahy textového protokolu to nelze provést snadno přímo, ale data je předem třeba zakódovat pomocí Base64 [11].

```
:retry 5000

: Comments begin with colon
event: question
id: 1337
data: What's your name?

: Comments still begin with colon
event: question
id: 42
data: Does pineapple belong on pizza?
```

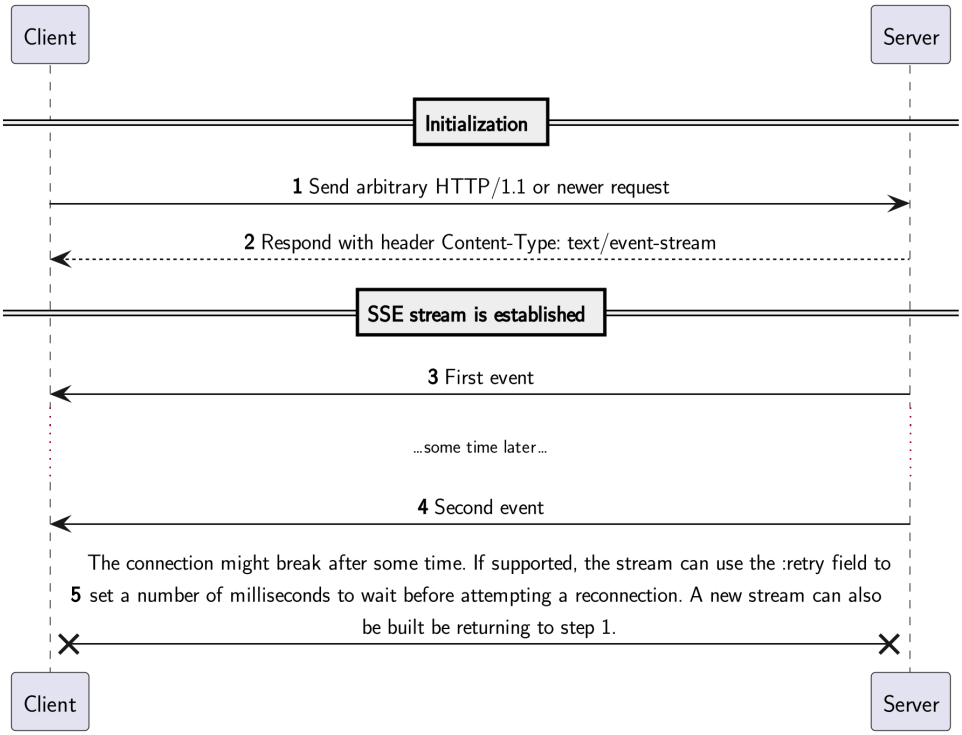
Kód 3.5: Ukázka SSE eventů

Samozřejmě může nastat také situace, že se SSE stream z nejrůznějších důvodu rozpadne. V takovém případě lze využít speciálního pole `retry`, které určuje počet milisekund, jež se má počkat před pokusem o obnovení spojení. Pokud není pole `retry` podporováno, nic nebrání klientovi v tom, aby znovu poslal na server HTTP dotaz, který otevře nový stream. Průběh navázání spojení a komunikace pomocí SSE streamu je znázorněn na obrázku 3.3.

Největší technická omezení SSE mohou být problémová, pokud je klientem webový prohlížeč. V takovém případě mohou vznikat chyby při vytváření SSE streamů, nebo se mohou naopak neobvykle rychle spojení rozpadat. Problém totiž spočívá v tom, jak si prohlížeč interně spravuje TCP spojení pro jednotlivé taby webových stránek a sdílí je. Problémová může být webová stránka, která používá více SSE streamů najednou a uživatel ji má otevřenou ve více tabech najednou, jelikož většina moderních prohlížečů implementující specifikaci HTTP/1.1 umožňuje maximálně 6 spojení pro jednu doménu. Typicky je taková aplikace například nějaký nástroj pro monitorování stavu aplikace jako Grafana[4].

Tento případ se naštěstí této aplikace netýká, jelikož účastníci soutěže

[4] `https://grafana.com/grafana/`



Obrázek 3.3: Sekvenční diagram SSE komunikace

budou vyvíjet své klienty velmi pravděpodobně s jinou technologií, než je webový prohlížeč, ačkoliv to možné je. Navíc je problém se sdílením spojení markantní jen pro HTTP/1.1, od verze HTTP/2 a výše obsahuje protokol HTTP podporu pro multiplexing [16].

3.4.3 Příjem odpovědí účastníků

Účastníci odesílají svá řešení otázek jako HTTP POST dotaz na jednoduchý endpoint `/rest/answer`. Metoda POST se používá kvůli potenciální velikosti těl dotazů, která účastníci zasílají. Endpoint využívá technologie JAX-RS [4].

Technicky je možné, aby i dotazy pomocí metody GET obsahovaly tělo, avšak specifikace HTTP/1.1 z RFC 2616 [7] v sekcích 4.3 a 9.3 udává, že metoda GET neobsahuje tělo. Bylo by to tedy nestandardní chování, které může vyvolat neočekávané vedlejší účinky nebo chybové stavy. RFC 2616 bylo později nahrazeno RFC 7230 až 7235, kde konkrétně RFC 7231 [6] v sekci 4.3.1 nově specifikuje, že metoda GET jednoduše nemá definované chování, pokud obsahuje tělo. Důsledek ale zůstává v principu stejný a to ten, že některé implementace takový požadavek mohou odmítnout.

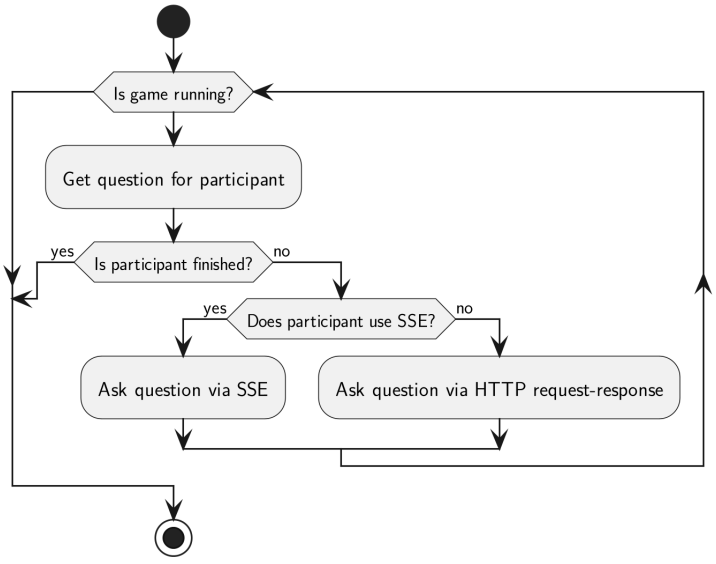
V prvních verzích aplikace endpoint `/rest/answer` neexistoval a účastníci odpovídali na otázky přímo odpovědí na příchozí dotazy od herního serveru. Při implementaci SSE jsem ale narazil na problém jednotného systému pro odesílání odpovědí na otázky, jelikož kvůli jednosměrné komunikaci přes SSE není možné narozdíl od WebSockets možné odpovídat stejným kanálem.

Z toho důvodu vznikl endpoint pro odesílaní odpovědí, který sjednocuje chování a logiku zpracovávání odpovědí účastníků.

Z tohoto řešení tedy benefituje jak herní server, který má pro odpovědi pouze jeden vstupní bod a nemusí tak podobnou logiku implementovat na více místech, tak účastníci. Pro ty díky tomuto rozhodnutí nezáleží na tom, jakou formu komunikace si vyberou, jelikož odpovídání na otázky je pro obě formy identické. Do hry to také vnáší rovné a férové podmínky pro obě možnosti, čemuž by tak v případě různých cest pro přímé HTTP dotazy a WebSockets nebylo a technické rozdíly by zasahovaly do tohoto procesu.

3.5 Herní smyčka

Kroky, které aplikace periodicky vykonává v hlavní herní smyčce, jsou znázorněné v sekvenčním diagramu na obrázku 3.5. Herní smyčka má za úkol získávat od otázkového serveru otázky pro účastníky a rozesílat je účastníkům jimi zvoleným komunikačním kanálem. Samotné rozesílání otázek pak zařizuje jednoduché rozesílání zpráv pomocí návrhového vzoru observer [9] a příslušných tříd připravených v balíčku `javax.enterprise.event`. Pohled na herní smyčku poskytuje diagram aktivit na obrázku 3.4.

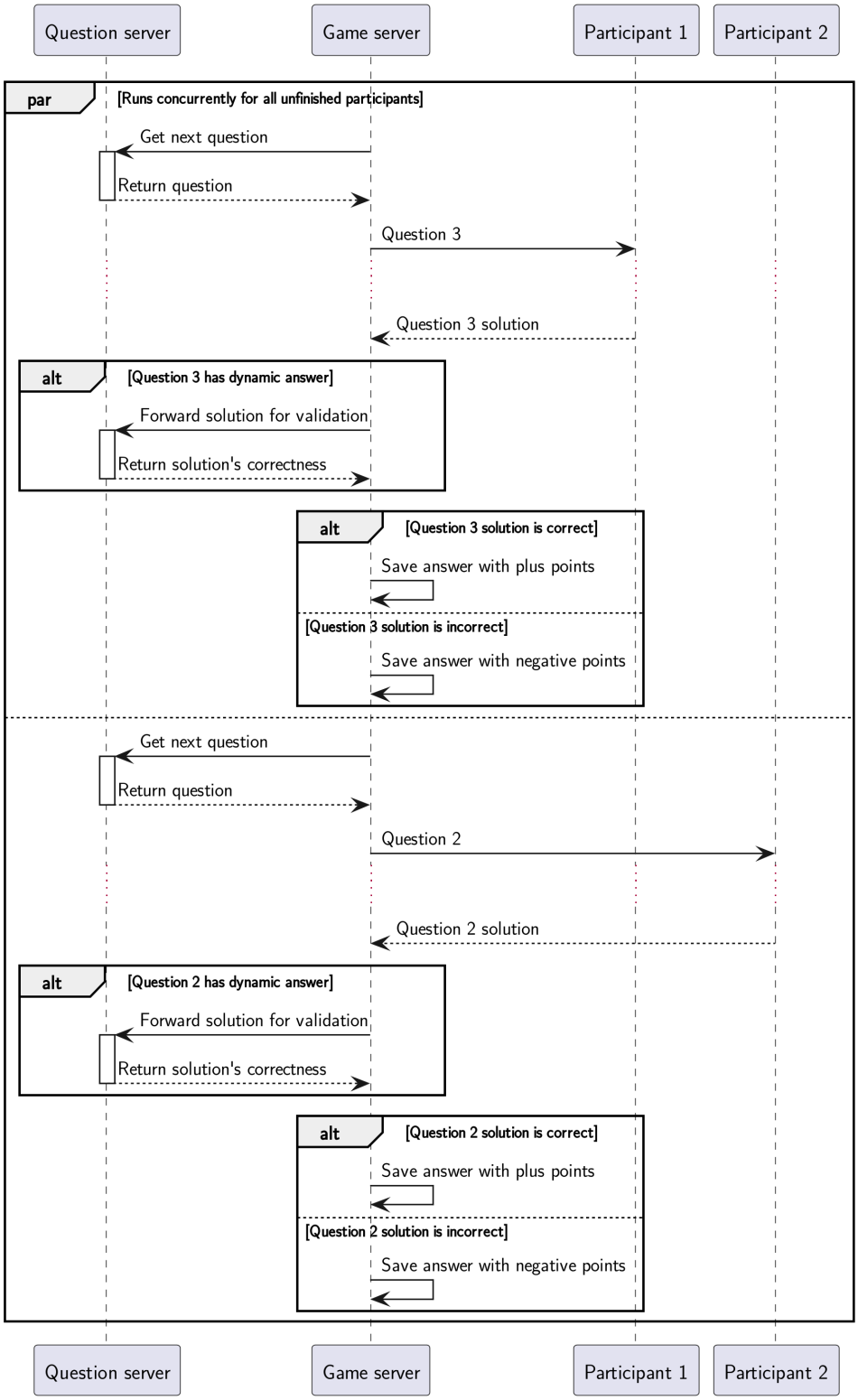


Obrázek 3.4: Diagram aktivit herní smyčky

3.5.1 Ovládání herní smyčky

Herní smyčka probíhá ve třídě `GameLoopController`, která má privátní subtřídu `LoopTask` implementující rozhraní `Runnable` a která instrumentaci tříd zajišťující samotnou logiku herní smyčky.

Třída `GameLoopController` využívá návrhového vzoru singleton [9] a slouží pro spouštění a pozastavování herní smyčky. `GameLoopController` spouští



Obrázek 3.5: Sekvenční diagram znázorňující komunikaci serveru s klienty

třídu `LoopTask` na pozadí v samostatném vlákně vytvořeného pomocí třídy `ManagedThreadFactory` z balíčku `javax.enterprise.concurrent`.

Samotná herní smyčka se může nacházet pouze ve dvou stavech. Spuštěno a pozastaveno. Mezi těmito stavy smyčku přepíná organizátor hry pomocí webového rozhraní.

3.5.2 Průběh herní smyčky

Samotná herní logika, která vykonává získávání otázek a jejich rozesílání, je implementována v třídách `QuestionPollBean` a `QuestionPollTask` implementující rozhraní `Runnable`.

Třída `QuestionPollBean` každý herní tik získá z databáze seznam účastníků, kteří ještě nedokončili hru, a pomocí třídy `ManagedExecutorService` z balíčku `javax.enterprise.concurrent` pro každého vybraného účastníka vytvoří ve vlastním vlákně objekt třídy `QuestionPollTask`. Ten si pak na základě odeslaných metadat o účastníkovi od otázkového serveru vyžádá další otázku, kterou pak vybraným kanálem pomocí Jakarta EE observeru odešle.

Posluchači pro události Jakarta EE observables jsou implementováni ve třídě `GameController`, která poskytuje metody pro oba druhy komunikace. Zároveň poskytuje HTTP endpoint pro asynchronní příjem odpovědí účastníků.

Aplikace obsahuje dva posluchače pro tyto události, kteří zajišťují rozesílání otázek účastníkům. Jeden z nich má na starost odesílání otázek pomocí HTTP a druhý pomocí SSE. Přijímané události jsou `HttpQuestionEvent` a `SseQuestionEvent`. Obě události pouze přejímací pole a metody jejich společného abstraktního předka `QuestionEvent` a jsou rozdělené zejména pro snadné rozlišení přijímané události již v argumentu metody posluchače.

3.6 Struktura balíčků aplikace

Zvolil jsem základní a přímočarou strukturu aplikace vzhledem k požadavkům na srozumitelnost kódu aplikace a snadnou úpravu. Zvažoval jsem použití i takzvané techniky *package by feature* [23], jež sdružuje kód, který souvisí s jednou doménou či funkcionalitou aplikace, pod společný balíček. To je velký rozdíl oproti klasickému dělení balíčků technikou *package by layer*, která typicky kód rozděluje podle vrstvy kódu, kterou implementuje. Ať už to jsou DAO, JPA entity nebo servisní vrstva.

Kvůli malému rozsahu aplikace jsem ale dospěl k závěru, že zvolená přímočará a klasická organizace balíčků technikou *package by layer* je dobře známá většině vývojářů a i s ohledem na rozsah kódu bude lepší volbou.

Balíček `concurrent` obsahuje třídy obsluhující asynchronní události v aplikaci. Je to hlavní herní smyčka `GameLoopController`, která se periodicky dotazuje na nové otázky pro účastníky a rozesílá je. Dále to jsou také třídy `HttpQuestionEvent` a `SseQuestionEvent` s jejich společným předkem `QuestionEvent` reprezentující události pro využití nativní Jakarta EE implementace návrhového vzoru observer.

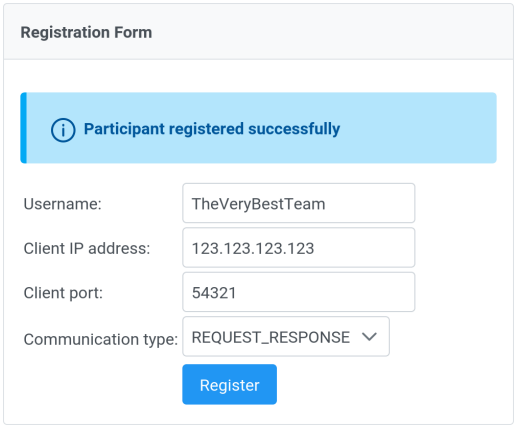
```
cz.radimsuckr.extremestartup
├── concurrent
│   └── event
├── dao
├── dto
├── exception
│   ├── dto
│   └── mapper
├── jsf
├── model
├── rest
└── service
```

Obrázek 3.6: Stromová struktura balíčků

V balíčku `rest` jsou k nalezení klienti pro REST API otázkového serveru a k rozesílání otázek účastníkům. Zároveň obsahuje i implementaci HTTP endpointu pro přijímání odpovědí účastníků.

3.7 UI aplikace

Protože UI je v této aplikaci ryze informativního charakteru a pro účastníky, kteří s ním během soutěže přijdou do styku naprosto minimálně, nemá, vyjma registračního formuláře a hlavní stránky s přehledem, žádný další význam. Tomu odpovídá i grafická úprava. Hlavním cílem je podat rychle a přehledně informace o aktuálním stavu aplikace a průběhu hry.



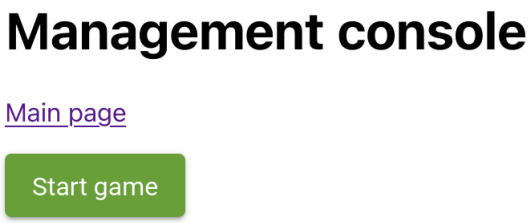
Obrázek 3.7: Registrační formulář

UI se skládá ze tří obrazovek. První obrazovka na obrázku 3.9 slouží jako hlavní přehledová stránka s aktuálními výsledky registrovaných účastníků a jejich průběžným pořadím. Výsledky a pořadí znázorňuje jak jednoduchý graf vytvořený pomocí JSF a knihovny PrimeFaces, tak prostý seznam účastníků seřazený dle průběžných součtů získaných bodů. Hlavní stránka se

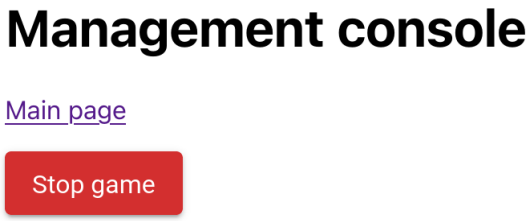
automaticky obnovuje každých 5 vteřin. Automatické obnovování zajišťuje prohlížeč a je nastavené HTML tagem `<meta>` s atributy `http-equiv="refresh"` a `content="5"`.

Druhá obrazovka na obrázku 3.7 je registrační formulář, kde účastníci vyplní své jméno (či název týmu), IP adresu jejich zařízení a port, na kterém jejich implementace poslouchá. Díky knihovně PrimeFaces poskytuje formulář okamžitou odezvu na případné chyby ve formuláři jako je zadání již existujícího jména, páru IP adresy a portu, nebo zadání neplatné IP adresy či portu mimo rozsah 1024 až 65535.

Porty menší než 1024 nejsou dovolené jako prevence potenciálních kolizí, protože tyto porty mohou být často obsazeny různými systémovými službami a aplikacemi na straně uživatele.



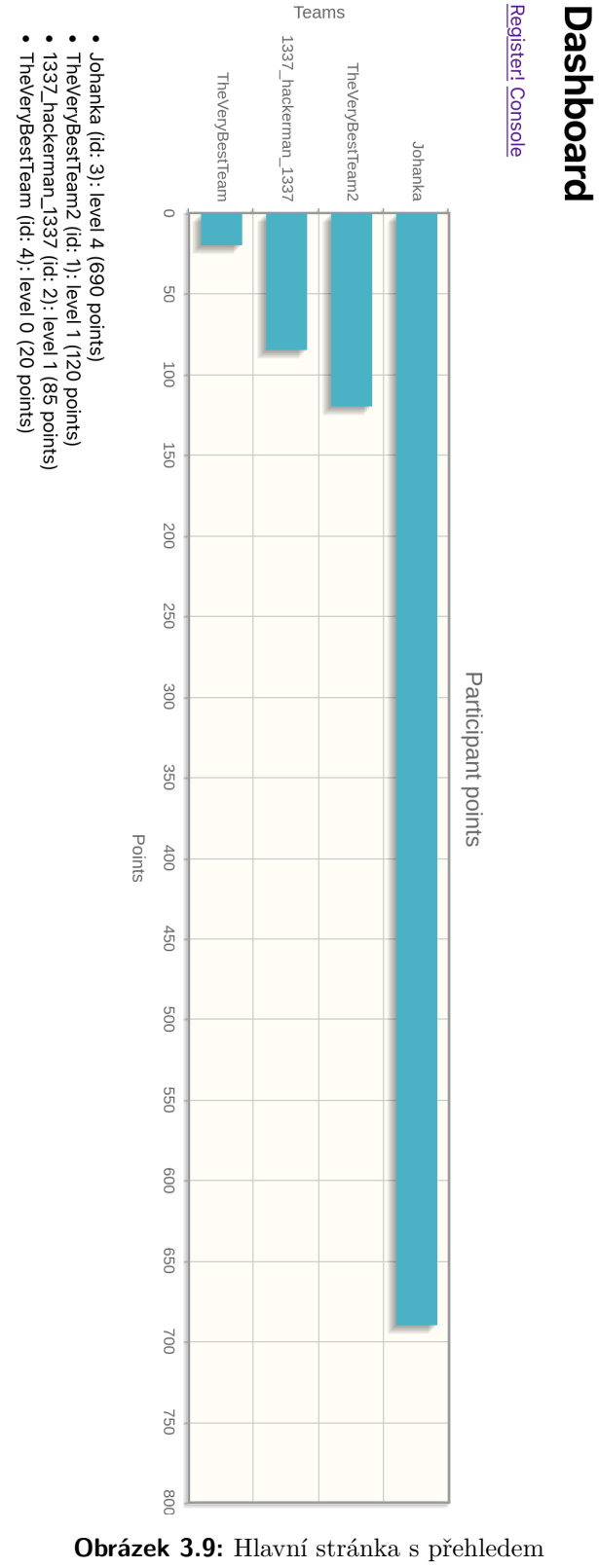
(a) : Herní smyčka je pozastavená



(b) : Herní smyčka je spuštěná

Obrázek 3.8: Administrativní stránka

Třetí obrazovka na obrázku 3.8 je jednoduchá administrativní stránka, která aktuálně obsahuje pouze tlačítko pro spuštění a zastavení hlavní herní smyčky.



Obrázek 3.9: Hlavní stránka s přehledem

3.8 Proces nasazení

Preferovaný způsob nasazení aplikace je pomocí Dockeru. Aplikace nabízí Dockerfile pro snadné sestavení a konfigurační soubor pro Docker Compose pro spuštění aplikace včetně PostgreSQL databáze. Otázkové servery by měly být rovněž distribuovány jako Docker image pro co nejsnažší zapojení. Samozřejmě je lze spustit i jiným způsobem, takové zapojení už ovšem záleží na schopnostech organizátora a není oficiálně doporučené.

Účastníci provozují své herní klienty na svých vlastních počítačích na IP adrese a portu, kterou zadali při registraci, a spolu s aplikací, kterou provozuje organizátor hry, mají dostupný obousměrný přístup na internet. Aplikace u organizátora běží na platformě Docker a pomocí API pro komunikaci s účastníky rozesílá soutěžní otázky.

Na obrázku 3.10 je diagram nasazení, který schéma nasazení všech komponent zachycuje.

3.8.1 Nasazení pomocí Dockeru a Docker Compose

Návod na sestavení a spuštění aplikace včetně databáze je velmi jednoduchý. Obsahuje totiž pouze jeden příkaz uvedený na ukázce kódu 3.6. Uvedený příkaz zajistí sestavení samotné aplikace, její spuštění na portu 8080 a zprovoznění PostgreSQL databáze včetně perzistentního úložiště pomocí Docker volumes[5]. Výchozí konfigurace neobsahuje žádný otázkový server a je na organizátorovi jej do konfiguračního souboru přidat či jeho provoz zařídit jiným způsobem.

```
1 docker-compose up
```

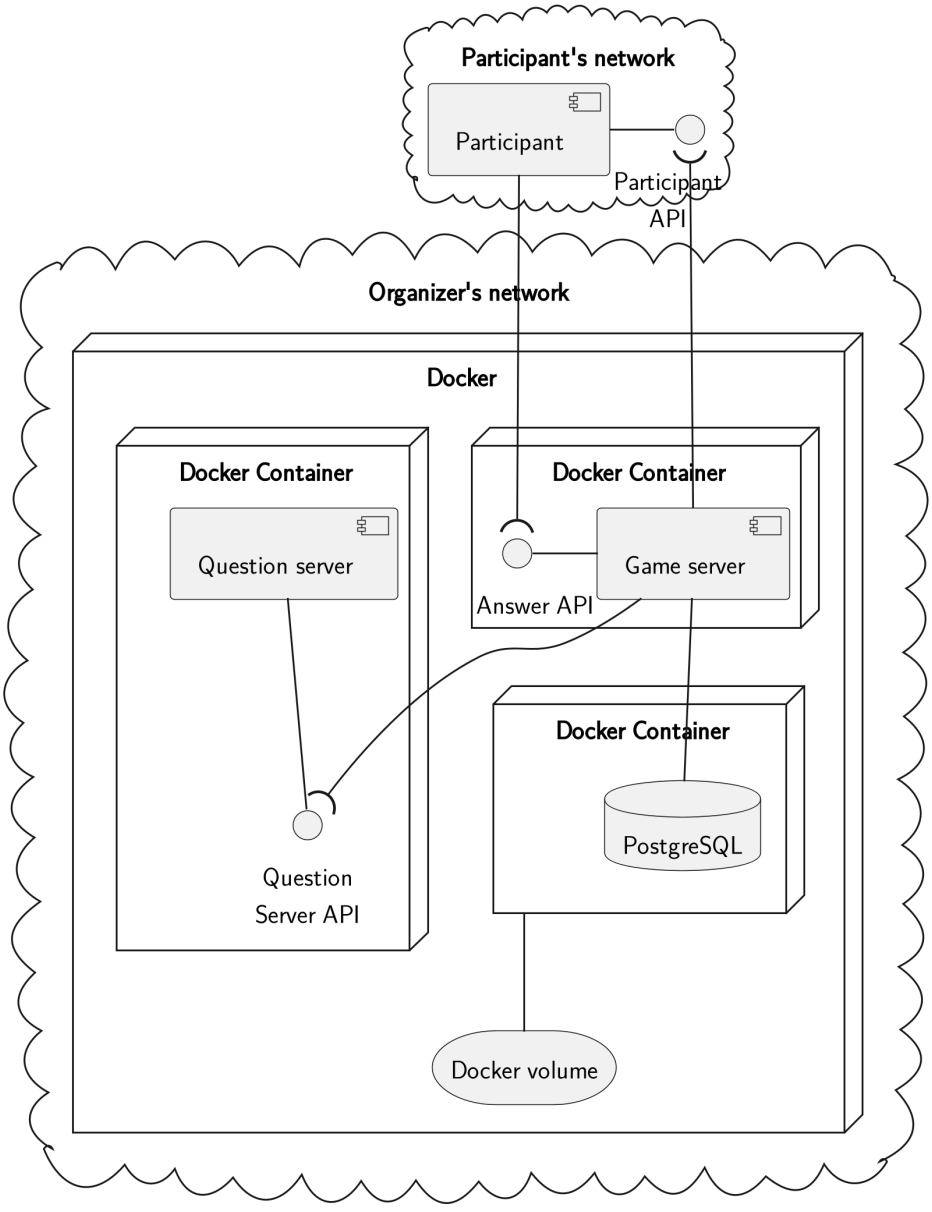
Kód 3.6: Spuštění aplikace přes Docker Compose

Příkaz pro spuštění lze rozšířit přidáním přepínače `--help` na konec. Tento přepínač zajistí, že Docker image se sestaví znovu, i pokud nejsou provedené žádné změny. Bez tohoto přepínače proběhne sestavení automaticky pouze pro první spuštění.

Docker je moderní a velmi snadný způsob provozu aplikací. Jednou z jeho předních výhod je izolace spouštěného procesu včetně jeho závislostí. Před spuštěním kontejneru tedy není nutné instalovat žádné další nástroje či balíčky, ale stačí stáhnout Docker image, který všechny závislosti již obsahuje. Připravený Docker image si buď můžeme již sestavený stáhnout z Docker repozitáře, nebo si jej můžeme sami sestavit, pokud máme k dispozici Dockerfile.

Docker Compose je nástroj naprogramovaný v jazyce Python, který přináší různé zkratky a pokročilejší funkce pro práci s Dockerem. Bez Docker Compose bychom si museli připravit prostředí sami pomocí jednotlivých příkazů binárky `docker`. S Docker Compose stačí připravit konfigurační soubor v jazyce YAML a spustit `docker-compose up`. V případě, že se odkazujeme na Dockerfile, ke kterému ještě nebyl sestaven Docker image, při prvním spuštění

[5] `https://docs.docker.com/storage/volumes/`



Obrázek 3.10: Diagram nasazení

se automaticky sestaví. Pokud bychom Docker image potřebovali sestavit znovu, můžeme tak učinit pomocí příkazu `docker-compose build` nebo před spuštěním příkazem `docker-compose up --build`. Docker Compose samozřejmě poskytuje mnohem více funkcí a příkazem `docker-compose --help` se o nich můžeme dozvědět více.

Použitá databáze PostgreSQL poskytuje svůj Docker image přes repozitář Docker Hub. Pří spuštění se tedy automaticky stáhne. Naopak naše aplikace žádný veřejný Docker image nemá a místo toho se před prvním spuštění sestaví z poskytuného Dockerfile.

Konfigurační soubor pro Docker Compose `docker-compose.yml` v kořenovém adresáři projektu obsahuje ještě definici takzvaného *volume*. Volumes slouží pro perzistentní úložiště na disku, které můžeme připojit k Docker kontejnerům. Docker kontejnery ve výchozím nastavení nemají žádné perzistentní úložiště. Výchozí úložiště se nazývá *ephemeral* a při zastavení kontejneru se odstraní. Databáze má k sobě volume připojený, aby mohla aplikace udržovat stav i mezi případnými restarty.

3.8.2 Nasazení mimo Docker

Aplikaci lze provozovat i bez Dockeru jako klasickou webovou aplikaci nasaditelnou jako WAR nebo self-contained JAR spustitelný přímo příkazem `java -jar`. Tento způsob spuštění vyžaduje Javu 11 a v případě nasazení pomocí WAR také zprovozněný aplikační server s podporou pro Jakarta EE.

Díky použití Payara Micro lze sestavit WAR pomocí příkazu `mvn package` a self-contained JAR příkazem `mvn package payara-micro:bundle`.

Výhodou WAR je menší velikost, jelikož obsahuje pouze zdrojový kód přeložený do JVM bytokódu a přiložené JAR soubory knihoven. Nevýhodou je nutnost nasazení aplikace do klasického aplikačního serveru provozovaného samostatně.

Naopak výhodou self-contained JAR je fakt, že obsahuje kromě bytekódu aplikace a knihoven také zabudovaný aplikační server, k jehož spuštění stačí pouze instalace Java 11. Spolu s Docker image vycházejícím z Payara Micro Docker image sdílí výhody externí konfigurace. To z Payara Micro aplikací činí velmi dobře použitelnou technologii pro takzvané *twelve-factor* [22] nebo *cloud-native* [10] [15] aplikace. Nevýhodou je oproti WAR větší velikost souboru.

Při nasazení aplikace mimo Docker musí organizátor také nějakým způsobem zajistit PostgreSQL databázi. Ta se opět dá provozovat v Dockeru nebo například jako nativní systémová služba.

Rovněž je potřeba zajistit spuštění otázkového serveru. Jeho provoz závisí na konkrétní implementaci, ať už organizátor používá již existující server, nebo si vytváří vlastní. Doporučený způsob je opět poskytnout Docker image nebo zdrojový kód, ze kterého lze Docker image sestavit. V opačném případě je nutné při použití specifické technologie nainstalovat všechny potřebné nástroje do hostitelského systému, což by jinak obstaral Docker build.

Ze zmíněných důvodů velmi doporučuji použít Docker, jelikož do jisté míry značně unifikuje práci s různými technologiemi.

Kapitola 4
Testování

4.1 Strategie unit testování

Cílem je udržet testovací strategii minimalistickou a efektivní. Vzhledem k bohaté historii implementací standardu Jakarta EE není nutné psát takové unit testy, které by především testovaly Payara Server, nebo implementaci JPA EclipseLink a jiné použité knihovny namísto kódu aplikace. Jinými slovy není cílem mít 100 % code coverage, ale mít takové testy, které opravdu udržují a kontrolují stěžejní funkcionalitu aplikace.

Unit testy jsou vytvořené pomocí knihovny JUnit 5[1]. Testy pokrývají zejména kód obsluhující hlavní herní smyčku a funkcionalitu. Jelikož se jedná o unit testy, jejich průběh netestuje žádné komplexní události jako například odesílání registračního formuláře účastníků, obsluhu asynchronních událostí, orchestraci vláken a další, které se obtížně automatizují. Tyto procesy byly testovány manuálně takzvaně *end-to-end* podle scénařů v sekci 4.2.

Vzhledem k rozsahu aplikace by komplexní automatizace testování zabrala velmi mnoho cenného času pro práci na aplikaci, který byl díky tomu věnován návrhu aplikace.

4.1.1 Metrika code coverage

Metrika *code coverage* jednoduše určuje, jaký podíl řádek zdrojového kódu aplikace je pokryt unit testy. Dále už se však nezabývá množstvím testů, kvalitou testů, nebo zda-li testy prochází všechny větve rozhodovací logiky pro dané řádky. Interpretovat tedy 100% code coverage jako fakt, že aplikace neobsahuje bugy, je chybné. Tato metrika je spíše užitečná jako upozornění na kód, který není téměř otestován, jelikož jeho pokrytí je nízké. Interpretace nízkého pokrytí ale velmi záleží na potřebách projektu a je individuální.

Zajistit 100% code coverage se dá docílit naprosto jednoduše tak, že napíšeme takové testy, které provolají 100 % řádek zdrojového kódu aplikace, ale testy jako takové mohou jednoduše obsahovat řádku `assertTrue(true)` a podobné další výrazy bez vypovídající hodnoty o testované logice. Testy mohou být navíc naprosto prázdné a stejně bychom docílili 100% pokrytí.

[1] `https://junit.org/junit5/`

Při návrhu testů pro aplikaci jsem použil zejména formu code coverage známou jako branch coverage, která určuje podíl otestovaných rozhodovacích větví. Testy tedy pokrývají důležitou logiku jádra aplikace, která obsluhuje získávání nových otázek pro účastníky, jejich distribuci a validaci a vyhodnocení odpovědí.

4.1.2 GitLab CI

Pro vývoj aplikace jsem použil platformu GitLab. Automatizace unit testů tedy probíhá ve službě GitLab CI. Unit testy se spouští pro každou větev verzovacího systému Git.

Jednou z výhod GitLab CI je také to, že dokáže interpretovat reporty testovacích frameworků o code coverage. Díky tomu dokáže při začleňování kódu barevně označit zeleně řádky, které jsou pokryty testy a červeně ty řádky, které testy pokryty nejsou.

4.2 Manuální testování

Pro manuální end to end testování je třeba aplikaci spustit včetně otázkového serveru a kompatibilních klientů. Lze použít připravený otázkový server a klienty v adresáři `example/python`. Scénař pro test integrace otázkového serveru lze použít i pro testování vlastních implementací.

4.2.1 Registrační formulář

Test začíná na hlavní stránce aplikace.

1. Kliknout na *Register!*
2. Vyplnit uživatelské jméno *team1*
3. Vyplnit IP adresu *123.123.123.123*
4. Vyplnit port *54321*
5. Kliknout na *Register*

Uživateli by se v tuto chvíli měla zobrazit hláška o úspěšném vytvoření účtu. Při zachování stejné databáze pro navazující běhy se otestují validační podmínky kombinaci vstupů formuláře. Například vytvoření stejnojmenného účastníka, zadání již existující kombinace IP adresy a portu, nebo zadání nevalidní IP adresy či portu mimo rozsah 1024 až 65535. Ve všech těchto případech se uživateli zobrazí chybové hlášky s konkrétní chybou a účastník nebude vytvořen.

4.2.2 Integrace otázkového serveru

Před začátkem testu je třeba zaregistrovat požadovaný počet klientů pro test. Test začíná kliknutím na zelené tlačítko *Start game* na stránce *Management console*. Výstupy testu pro kontrolu výsledků jsou zejména logy aplikace.

Při použití ukázkových implementací v Pythonu nejprve otázkový server zaznamená do logu informace o příchozím dotazu na novou otázku pro účastníka. Následně příslušný klient vytvoří log o nové příchozí otázce z herního serveru. Jelikož jsou otázky snadné a klienti je správně rozeznávají, odešle klient správnou odpověď hernímu serveru. Herní server opět zaloguje příchozí požadavek, který je tentokrát odpověď od účastníka.

Na základě správné odpovědi přičte herní server odpovídající počet bodů účastníkovi a započtené body jsou viditelné na hlavní stránce. Takto proběhnou první tři otázky se statickými odpověďmi. Čtvrtá otázka testuje přeposílání odpovědí na otázky s dynamickými odpověďmi. Herní server tedy obdrženou odpověď přepošle na otázkový server, který vyhodnotí její správnost. Na základě odpovědi otázkového serveru přičte herní server odpovídající počet bodů. Jelikož je čtvrtá otázka finální otázkou, odpoví otázkový server na další dotaz na novou otázku HTTP stavovým kódem 204 a tím dokončil účastník hru.

Tento scénář se dá využít pro implementaci vlastního otázkového serveru, jelikož je při něm snadné sledovat aktuální stav hry a zda-li otázkový server vybírá správné otázky při dotazech, nebo jestli funguje vyhodnocování přeposílaných odpovědí. Samozřejmě lze otázkový server doplnit o unit testy a tak přispět k celkové kvalitě implementace.

4.3 Ukázkové implementace otázkového serveru a klientů

Návod pro zprovoznění ukázkových implementací je v souboru `example/python/readme.md`.

4.3.1 Otázkový server

Přiložený otázkový server ze souboru `question_server.py` vybírá náhodně otázky z aktuálního levelu účastníka. Do nového levelu účastníky posouvá pouze za podmínky, že účastník zodpověděl alespoň 5 otázek a podíl správných odpovědí je minimálně 75 %. Výjimkou je poslední level hry, kde je potřeba minimálně 10 odpovědí a z toho minimálně 80 % správně. Po splnění podmínky posledního levelu server vrací HTTP 204 a účastník úspěšně dokončil hru.

Pokud je před spuštěním serveru nastavená proměnná prostředí `QUESTIONS_FROM_FILE`, ignoruje server přednastavené otázky a namísto toho načte otázky z externího JSON souboru. Avšak při použití externího souboru podporuje *pouze* otázky se statickými odpověďmi.

4.3.2 HTTP klient

Klient, který přijímá otázky pomocí klasické HTTP komunikace, se nachází v souboru `client.py`. ID registrovaného účastníka, přebírá v proměnné prostředí `PARTICIPANT_ID`. Samotná implementace je naprosto triviální a odpovídá na předem známé otázky.

4.3.3 SSE klient

SSE klient přijímá otázky pomocí SSE a nachází se v souboru `sse_client.py`. Kromě rozdílného komunikačního protokolu dále funguje stejně jako HTTP klient ze sekce 4.3.2. Také přebírá ID registrovaného účastníka v proměnné prostředí `PARTICIPANT_ID` a implementace odpovědí je identická.

Kapitola 5
Závěr

5.1 Výstupy projektu

Výstupem mé práce je funkční verze aplikace pro organizování soutěží pro programátory dle pravidel hry Extreme Startup, jejíž výhodou oproti existující implementaci jsou možnost přidání otázek do hry pomocí HTTP komunikace i bez znalosti programovacího jazyka Java, snadné zprovoznění a podpora SSE, která zjednodušuje pořádaní her přes internet.

Výsledná aplikace slouží jako dobrý základ pro pořádání soutěží ve hře Extreme Startup a poskytuje mnoho prostoru pro další vylepšení. Největším přínosem oproti existujícím řešením je zavedení otázkových serverů a možnost pořádat soutěže přes internet pomocí SSE komunikace.

5.2 Shrnutí

Jelikož opensource programátorských her nebo soutěží není příliš velké množství, vytvářím v této práci novou implementaci pravidel hry Extreme Startup. Při analýze původní implementace vyplynula jako největší překážka závislost na konkrétní technologii při rozšiřování soutěže o vlastní otázky, což komplikuje práci organizátorům soutěže.

Na základě analýzy existujících řešení jsem se rozhodl rozšířit architekturu aplikace o další server nazývaný jako *otázkový server*, který má na starost poskytování soutěžních otázek. Otázkový server implementuje organizátor soutěže dle specifikovaného API. Technologie použité pro vývoj aplikace vychází jednak ze zadání práce a jednak z mých osobních zkušeností. Stěžejní použité technologie jsou Jakarta EE, Payara Micro, PostgreSQL a Docker.

S úmyslem zachovat návrh i kód aplikace co nejjednodušší jsem navrhl i datový model. Ten sestává ze dvou základních a jednoduchých JPA entit *Participant*, která ukládá informace o účastnících soutěže, a *Asnwer*, která zaznamenává odpovědi účastníků na jim kladené jednotlivé otázky. Herní server poskytuje účastníkům hry takzvané *answer API*, pomocí kterého přijímá odpovědi na otázky. Herní server naopak od účastníků předpokládá velmi jednoduchý způsob zasílání otázek, který se nazývá *participant API*. Rozesílání otázek je možné buď klasicky protokolem HTTP, nebo i protokolem

SSE, což přináší obrovské výhody při pořádání her přes internet. Aplikaci lze spustit klasicky přímo pomocí JVM nebo velmi snadno pomocí Dockeru, který aplikace dobře podporuje a jehož použití doporučuji. V kapitole 3 ještě popisuji konkrétně logiku herní smyčky, strukturu a architekturu kódu a UI aplikace.

Pro testování aplikace jsem zvolil kombinovanou strategii. Unit testy aplikace testují pouze stěžejní rozhodovací logiku aplikace, protože další části kódu jsou natolik přímočaré, že jejich testy by prakticky testovaly zejména implementaci aplikačního serveru Payara Micro nebo knihovny EclipseLink, za kterými stojí velké komunity dobrovolníků i komerčních vývojařů. Unit testy jsou napsané v JUnit 5. Další částí testovací strategie jsou manuální testy. Manuální přístup jsem zvolil zejména kvůli časově poměrně náročné instrumentaci end-to-end testování, která by se vzhledem k rozsahu projektu nevyplatila. Manuální testování podporují ukázkové implementace otázkového serveru a k němu odpovídajících klientů pro HTTP i SSE. Otázkový server i klienti jsou naprogramováni v jazyce Python a slouží zároveň jako inspirace pro organizátory a účastníky soutěže.

5.3 Možnosti pro další funkce

Jednou z možností pro další funkce je implementace mechanismů, které by při získávání otázek kontrolovaly za běhu jeho férovost. Například zda-li se nesnaží otázkový server hodnotit stejnou otázku různě pro různé účastníky nebo jestli přijímá na stejnou otázku stejnou odpověď od různých účastníků.

Další zajímavou funkcí je možnost načasovat start a konec hry tak, aby probíhala automaticky. Organizátor by tak stejně jako nyní připravil prostředí pro běh hry, ale zároveň nastaví hře čas, kdy má automaticky spustit herní smyčku a čas, kdy se má hra zastavit. Jelikož je průběh a výsledný stav hry uložen v databázi, lze výsledky vyhodnotit až dodatečně.

Jelikož UI a celkový design aplikace nebyl pro tuto práci prioritou, nabízí se UI jako další část pro spoustu vylepšení. Například sjednotit design prvků aplikace, lépe využít barvy pro reprezentaci herního stavu a výsledků. Rozšířit UI o případné další ukazatele, které zvýší čitelnost herního stavu, nebo rozdělit sloupce grafu na hlavní stránce na různě barevné segmenty podle levelu, ve kterém byly body získány.

Rozšíření aplikace mohou být dobrým námětem na navazující práce.

5.4 Zdrojové kódy

Veškeré zdrojové kódy, které tvoří herní server a ukázkové implementace, jsou dostupné jako Git repozitář na adrese `https://gitlab.com/radimsuckr/extreme-startup`.

Literatura

[1] Rudy De Busscher. Relationship between Payara Platform, MicroProfile and Java EE/Jakarta EE, Říjen 2019. URL: https://blog.payara.fish/relation-between-payara-platform-microprofile-and-java-ee/jakarta-ee.

[2] Robert Chatley. Extreme Startup, 2022. URL: https://github.com/rchatley/extreme_startup.

[3] Cloudflare Inc. What is vendor lock-in?, 2022. URL: https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/.

[4] Eclipse Foundation. Jakarta RESTful Web Services 2.1, Září 2019. URL: https://jakarta.ee/specifications/restful-ws/2.1/.

[5] Eclipse Foundation. Jakarta EE, 2022. URL: https://jakarta.ee/.

[6] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor, Červen 2014. http://www.rfc-editor.org/rfc/rfc7231.txt. URL: http://www.rfc-editor.org/rfc/rfc7231.txt.

[7] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, Červen 1999. http://www.rfc-editor.org/rfc/rfc2616.txt. URL: http://www.rfc-editor.org/rfc/rfc2616.txt.

[8] Germano Gabbianelli. Server-Sent Events: the alternative to WebSockets you should be using, Únor 2022. URL: https://germano.dev/sse-websockets/.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson Education, 1994.

[10] Emily Jiang, Andrew McCright, John Alcorn, David Chan, and Alasdair Nottingham. *Practical Cloud-Native Java Development with MicroProfile: Develop and deploy scalable, resilient, and reactive cloud-native applications using MicroProfile 4.1.* Packt Publishing, Září 2021.

[11] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, RFC Editor, Říjen 2006. `http://www.rfc-editor.org/rfc/rfc4648.txt`. URL: `http://www.rfc-editor.org/rfc/rfc4648.txt`.

[12] Antti Laaksonen. *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests.* Springer, Switzerland, 2017.

[13] Diego Lemos. Extreme Carpaccio, 2022. URL: `https://github.com/dlresende/extreme-carpaccio`.

[14] Alexey Melnikov and Ian Fette. The WebSocket Protocol. RFC 6455, Prosinec 2011. URL: `https://www.rfc-editor.org/info/rfc6455`, `doi:10.17487/RFC6455`.

[15] Microsoft. What is Cloud Native?, Duben 2022. URL: `https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition`.

[16] Kaitlin Moreno. Thoughts on Server-Sent Events, HTTP/2, and Envoy, Leden 2020. URL: `https://medium.com/blogging-greymatter-io/server-sent-events-http-2-and-envoy-6927c70368bb`.

[17] Oracle. Differences between Java EE and Java SE, 2012. URL: `https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html`.

[18] Oracle. Java programming language, 2021. URL: `https://dev.java/`.

[19] Oracle America Inc. JAR File Specification, 2017. URL: `https://docs.oracle.com/en/java/javase/11/docs/specs/jar/jar.html`.

[20] Oracle America Inc. The Java® Virtual Machine Specification, Srpen 2018. URL: `https://docs.oracle.com/javase/specs/jvms/se11/html/`.

[21] WHATWG. Server-sent events, Květen 2022. URL: `https://html.spec.whatwg.org/multipage/server-sent-events.html`.

[22] Adam Wiggins. The Twelve-Factor App, 2017. URL: `https://12factor.net/`.

[23] Grzegorz Ziemoński. Package by Feature Is Demanded, Březen 2017. URL: `https://dzone.com/articles/package-by-feature-is-demanded`.

Slovník

API Application Programming Interface. 13–15, 17, 18, 24, 27, 35

CDI Jakarta Contexts and Dependency Injection. 9

CI Continuous Integration. 32

DAO Data Access Object. 23

EJB Jakarta Enterprise Beans. 9

HTML HyperText Markup Language. 25

HTTP HyperText Transfer Protocol. 2, 3, 7, 8, 13, 17–21, 23, 24, 33–36

ID Identifier. 14, 15, 17, 34

IP Internet Protocol. 14, 18, 19, 25, 27, 32

JAR Java Archive. 7, 9, 10, 29

JMS Jakarta Messaging. 9

JPA Jakarta Persistence. 8, 9, 11, 13, 23, 31, 35

JSF Jakarta Server Faces. 9, 24

JSON JavaScript Object Notation. 3, 7, 13, 16–19, 33

JTA Jakarta Transactions. 9

JVM Java Virtual Machine. 7, 9, 29, 36

LAN Local Area Network. 8

LXC Linux Containers. 10

NAT Network Address Translation. 8

RDBMS Relational Database Management System. 11, 13

REST Representational state transfer. 17, 24

RFC Request For Comments. 20

RPC Remote Procedure Call. 17

SSE Server-sent Events. 4, 7, 8, 13, 18–20, 23, 34–36

SSH Secure Shell. 18

TCP Transmission Control Protocol. 8, 19

TDD Test Driven Development. 2

UI User Interface. 8, 24, 36

VPN Virtual Private Network. 8, 18

WAN Wide Area Network. 18

WAR Web Application Archive. 29

XP Extreme Programming. 2

YAML YAML Ain't Markup Language. 8, 27, 40