VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYNCHRONIZACE WEBOVÝCH APLIKACÍ NA PLATFORMĚ JAVA EE

JAVA EE APPLICATION SYNCHRONIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAN POUL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19421/2016/xpoulj01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Poul Jan, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Synchronizace webových aplikací na platformě Java EE**
Java EE Application Synchronization
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s platformou Java EE se zaměřením na podporu technologií pro oboustrannou komunikaci klient-server (např. web sockets).
2. Prostudujte dostupné nástroje pro tvorbu klientských webových aplikací s grafickým výstupem pomocí technologií HTML 5.
3. Navrhněte architekturu aplikace umožňující synchronizovat stav aplikace a grafický výstup pro větší množství klientů.
4. Po dohodě s vedoucím zvolte vhodnou demonstrační aplikaci pro navrženou architekturu a implementujte ji pomocí vhodných technologií.
5. Proveďte testování vytvořené aplikace.
6. Zhodnoťte dosažené výsledky

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá využitím technologie Java EE a oboustrannou komunikací typu klient-server. Řeší návrh architektury aplikace umožňující synchronizovat stav aplikace a grafického výstupu pro větší množství klientů. Výsledné řešení obsahuje ukázkovou aplikaci, která poskytuje synchronizaci připojených klientů.

Abstract

This work deals with the Java EE platform and the bidirectional client-server communication. It presents a design of an application architecture that allows to synchronize the state of applications and their graphical output for a larger number of clients. The final solution contains a sample application that demonstrates the synchronization of connected clients.

Klíčová slova

Java EE, HTML 5, Synchronizace, WebSocket

Keywords

Java EE, HTML 5, Synchronization, WebSocket

Citace

POUL, Jan. *Synchronizace webových aplikací na platformě Java EE*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Synchronizace webových aplikací na platformě Java EE

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Poul
22. května 2017

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce Ing. Radku Burgetovi, Ph.D. a Ing. Václavu Topinkovi za odbornou pomoc a cenné rady při konzultacích.

Obsah

Kapitola 1

Úvod

Tento dokument obsahuje technickou zprávu diplomové práce na téma „Synchronizace webových aplikací na platformě Java EE.“ Nejprve budou popsány jednotlivé technologie se kterými bylo nutné se seznámit v rámci studia problematiky serverové části aplikace. Následně budou rozebrány technologie klientské části aplikace se zaměřením na HTML 5 WebSocket API a protokol WebSocket, které jsou velice důloežitou součástí této práce. Dále zde budou popsány jednotlivé problémy se kterými jsem se setkal při studiu těchto technologií a navržena jednoduchá aplikace, která bude umožňovat synchronizaci grafického výstupu pro větší množství klientů.

Kapitola 2

Analýza technologie Java EE

Serverová část aplikace s podporou technologií pro oboustrannou komunikaci klient-server umožnňující synchronizaci většího množství klientů bude napsána platformě Java EE. Velké množství informací v této kapitole, ale i celém tomto dokumentu, je čerpáno z knihy Java EE 7 Tutorial[5].

Java Platform, Enterprise Edition (dále Java EE), je součást platformy Java, která je určena pro vývoj a provoz informačních systémů a podnikových aplikací. Základem pro platformu Java EE je platforma Java Standard Edition (dále Java SE), ve které jsou definovány jednotlivé nadstavby, ze kterých se skládá Java EE. Pokud tedy chceme provozovat Java EE aplikaci je nutné mít nainstalováno také standardní Java SE API, které je technologií Java EE využíváno. Java EE obsahuje referenční implementaci aplikačního serveru kompletně podporující poslední vydanou specifikaci platformy Java EE. Tato referenční implementace slouží především jako ukázka implementace nových rysů v poslední specifikaci. V případě Java EE 7 se jedná o aplikační server Glassfish 4.1.

Java EE poskytuje oproti Java SE navíc desítky rozhraní, která usnadňují vývoj v určitých oblastech. Tato rozhraní předepisuje Java EE API a některé z více než dvou desítek těchto rozhraní jsou uvedeny v následujícím seznamu:

- **Java Persistence API** - standardní rozhraní pro objektově-relační mapování, pomocí tohoto rozhraní komunikujeme s databází.
- **Java Transaction API (JTA)**, - usnadňuje správu transakcí.
- **JavaMail** - odesílání emailů.
- **Java API for XML Web Services** - poskytuje podporu pro webové služby.
- **Java Server Faces** - knihovna, která je zaměřena na podporu tvorby webových aplikací, především jejich uživatelského rozhraní.
- **EJB (Enterprise Java Beans)** - komponenty obchodní logiky.
- **Servlet** - obsluha protokolu HTTP na straně serveru.
- **WebSocket** - novinka v Java EE 7, poskytuje podporu pro vytváření WebSocket aplikací.
- a mnoho dalších služeb zjednodušujících vývoj i údržbu podnikových aplikací.

Ve své prvotní formě jsou tato rozhraní pouhou specifikací, která nastavuje minimální požadavky na funkcionalitu Java EE platformy. Nejedná se tedy o konkrétní implementaci. Jednotlivé implementace rozhraní mohou být, a většinou jsou, vyvíjeny různými společnostmi, které mohou nad rámec minimálních funkčních požadavků přidat také něco navíc. Existuje však i implementace od firmy Oracle (dříve firma SUN), která implementuje pouze základní specifikaci rozhraní a tuto implementaci označuje jako referenční. Programátor má možnost výběru, jaké implementaci konkrétního rozhraní dá přednost.

2.1 Aplikační server

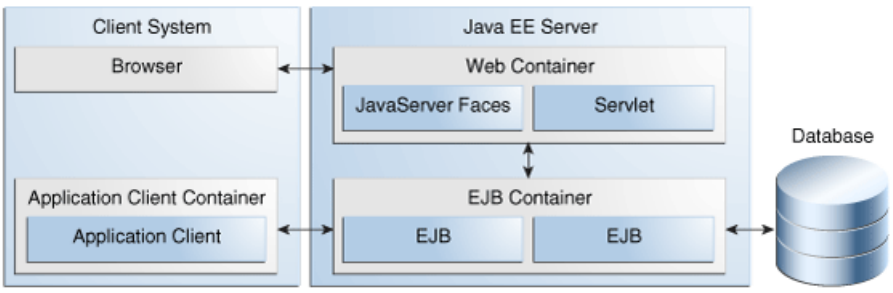
Pojmem Aplikační server je označována serverová aplikace zastřešující všechny knihovny, které dle specifikací Java EE platformy zajišťují požadovanou funkcionalitu. Tyto knihovny implementují veškerá rozhraní obsažená ve specifikaci Java EE. Navíc aplikační server poskytuje služby, jako např. administrátorskou konzoli nebo možnost logování.

Existuje více druhů aplikačních serverů. Je možnost vybrat si server s otevřenými zdrojovými kódy nebo některou z komerčních verzí. Ne všechny aplikační servery implementují kompletní specifikaci Java EE, ale pouze její část. Při výběru aplikačního serveru je třeba vzít tuto vlastnost v úvahu. V rámci analýzy bylo studováno několik aplikačních serverů. V rámci studia proběhla analýza několika aplikačních serverů, která je popsána v části „Analýza dostupných aplikačních serveru“ 2.1.2.

Na aplikační server se nasazují takzvané komponenty. Jedná se o základní jednotky, ze kterých je výsledná aplikace složena. Předtím než je komponenta spuštěna, musí být přiřazena do takzvaného kontejneru.

2.1.1 Kontejnery

Kontejnery jsou části aplikačního serveru, dle specifikace funkcionality technologie Java EE, které poskytují prostředí pro běh komponent. Na obrázku 2.1 vidíme, které kontejnery Java EE specifikuje.



Obrázek 2.1: Uspořádání kontejnerů, ve specifikaci Java EE. Převzato z [5]

Kontejnery dle specifikace Java EE, které budou použity v rámci návrhu aplikace.

- **Webový kontejner** - stará se o běh webové vrstvy a má na starosti správu webových komponent.
- **EJB kontejner** - má na starosti správu EJB komponent, poskytuje řízení transakcí, vytváří a spravuje instance.

- **Java EE kontejner** - jedná se o webový kontejner + EJB kontejner.

2.1.2 Analýza dostupných aplikačních serverů

Java EE je velice rozšířená platforma pro vývoj podnikových aplikací, proto existuje velké množství dostupných aplikačních serverů. Jedná se o komerční i nekomerční řešení. Komerční řešení se snaží poskytovat maximální výkon, stabilitu a podporu. K některým řešením s otevřeným zdrojovým kódem je možné dokoupit komerční podporu.

Některé aplikační servery implementují kompletní specifikaci technologie Java EE, jiné implementují pouze její část, například pouze webový kontejner. Servery podporující pouze webový kontejner jsou dostačující pro menší projekty, kde není nutná obchodní vrstva. V následujícím přehledu budou představeny některé z aplikačních serverů, které jsou v době psaní této práce aktivní projekty.

	Implementace specifikace Java EE	Komerční podpora
Apache Tomcat	částečná	ANO
TomEE	kompletní	ANO
WildFly	kompletní	ANO
Glassfish	kompletní	NE
Payara	kompletní	ANO

Tabulka 2.1: Některé z dostupných aplikačních serverů

Apache Tomcat - Tento aplikační server je možné stáhnot na stránkách projektu Apache Tomcat [13]. Jedná se pouze o Webový kontejner, který je vytvářen nadací Apache Software Foundation. Je distribuován pod licencí „Apache License“. Nejedná se o plnohodnotný Java EE server, ale na velkou většinu aplikací je plně postačující. Chybějící specifikace je možné doplnit. V případě potřeby kompletního Java EE kontejneru je možnost zvolit server TomEE, který kompletně imeplementuje specifikaci Java EE.

WildFly - Aplikační server WildFly (před rokem 2013 JBoss Application Server) lze stáhnout na stránkách projektu Wildfly [18]. Jedná se o kompletní specifikaci Java EE. Server je distribuován s otevřenými zdrojovými kódy pod licencí „GNU Lesser General Public License.“ Jedná se o plnohodnotnou implementaci specifikace Java EE, kterou vyvíjí společnost Red Hat.

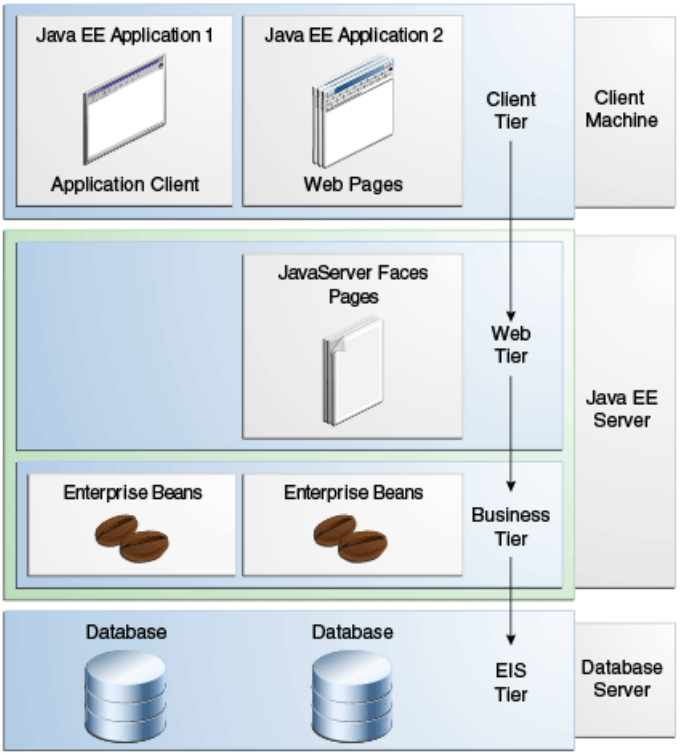
Glassfish a Payara - Projekt Glassfish lze stáhnout na stránkách projektu Glassfish [14]. V roce 2011 byl společností Oracle vydán Glassfish3 jako referenční implementace pro platformu Java EE 6. Oracle následně oznámil ukončení komerční podpory serveru Glassfish. Jako referenční implementace Java EE 7, byl vydán Glassfish 4.1. I přes absenci komerční podpory od firmy Oracle.

Práce na tomto projektu se stále zpomalovala a nahlášené problémy nebyly řešeny vůbec, nebo s velkou časovou prodlevou, proto se komunita vývojářů okolo tohoto projektu rozhodla vytvořit vlastní odnož aplikačního serveru Glassfish, kterou pojmenovala Payara fish. Aplikační server Payara opět poskytuje komerční podporu vývojářům a opravuje nahlášené problémy. Projekt Payara fish je dostupný na stránce projektu Payara [17].

Zvolený aplikační server - Pro tento projekt bude použit aplikační server Payara fish, který vychází z referenční implementace platformy Java EE 7, projektu Glassfish 4.1.

2.2 Vícevrstvá architektura

Vícevrstvá architektura je základním stavebním kamenem aplikací v Java EE. Komponenty jsou rozděleny podle svého typu do oblastí zvané vrstvy. Tyto vrstvy mohou být instalované na jednom nebo na různých počítačích. Na následujícím obrázku můžeme vidět schéma vrstev pro platformu Java EE, která používá čtyřvrstvou architekturu. Platforma Java EE se zaměřuje na vývoj pro business a webovou vrstvu. Tyto dvě vrstvy obvykle běží na stejném stroji (aplikačním serveru), jak je znázorněno na obrázku 2.2.



Obrázek 2.2: Více vrstvá architektura platoformy Java EE. Převzato z [5]

Jednotlivé komponenty lze nasadit na různé servery a tím optimalizovat zatížení a celkovou škálovatelnost systému. V případě, že se jednotlivé vrstvy prolínají a získávají oboustrannou závislost, se patrně jedná o chybný návrh systému. [1]

2.2.1 Klientská vrstva

Prezentační vrstva slouží k zajištění služeb uživatelského rozhraní. Uživatelské rozhraní je závislé na platformě nebo operačním systému, na němž je aplikace provozována.

Dva základní typy klientů jsou webový prohlížeč (tenký klient) nebo klientská aplikace (těžký klient). Uživatelské rozhraní je tvořeno ovládacími prvky, díky nimž má uživatel možnost aplikaci ovládat. Hlavní úlohou klienta je zasílat požadavky střední vrstvě (Webová a business vrstva), přijímat od ní odpovědi a tyto odpovědi následně prezentovat uživateli.

Tenký klient v podobě webové aplikace je upřednostňován, protože uživatel není nucen instalovat speciální software na pracovní stanice a stačí využít pouze běžný internetový prohlížeč. Aplikace je tak ve většině případů dostupná nepřetržitě. Prohlížeč komunikuje pouze s webovou vrstvou pomocí HTTP(s) protokolu. Následně webová vrstva interpretuje

klientské požadavky business vrstvě, která je obsluhuje a posílá stejnou cestou odpověď zpět do webového prohlížeče.

Těžký klient (desktop aplikace, mobilní aplikace) je využíván zejména v případech, kdy není vhodné využít vlastnosti http protokolu. Jedná se zejména o komplikovanější uživatelské rozhraní vyžadující častou aktualizaci dat nebo reference na další dialogová okna nebo ovládací prvky. Těžký klient může rovněž komunikovat s webovou vrstvou. Na rozdíl od tenkého klienta ji však může také přeskočit a komunikovat přímo s EJB komponentami v business vrstvě.

Zodpovědností klientské vrstvy je poskytování uživatelského rozhraní, zachycování uživatelských událostí, odesílání a přijímání HTTP(s) požadavků. Tato vrstva by nikdy neměla měnit hodnoty dat nebo řídit vykonávání kódu.

2.2.2 Webová vrstva

Na obrázku 2.3 lze pozorovat, jaké komponenty Webová vrstva obsahuje. Jedná se především o takzvané Java Servlety, Java Server Faces komponenty a JavaBeans komponenty. Tyto komponenty zpracovávají klientské požadavky a generují odpovědi. Tyto odpovědi na klientské požadavky jsou zasílány zpět do klientské vrstvy. Většinou se jedná o komunikaci protokolem HTTP(s). Při zpracování klientského požadavku může webová vrstva komunikovat s business vrstvou a získávat od ní určité informace.

Dle obrázky 2.3 lze sledovat, že technologie WebSocket, která bude použita v návrhu aplikace, byla přidána do specifikace platformy Java EE jako novinka ve verzi 7.

Java Server Faces komponenty lze rozšiřovat o celé knihovny značek. Například knihovna značek PrimeFaces (více informací na stránkách projektu PrimeFaces [20], RichFaces [21] a dalších. Je možné si napsat vlastní knihovnu značek se specifickými komponentami, které chceme používat ve vytvářených aplikacích.

2.2.3 Business vrstva

V business vrstvě leží celé jádro aplikace. Implementace business vrstvy je vyvíjena pomocí objektů Enterprise Java Beans (EJB). Ty přijímají požadavky od klientské vrstvy a na jejich základě pracují se zdroji z EIS vrstvy, která bude popsána dále. Následně zasílají odpověď zpět klientské vrstvě. Komponenty jsou nasazeny do EJB kontejneru poskytující běhové prostředí. Kontejner se automaticky stará například o řízení zátěže či transakce.

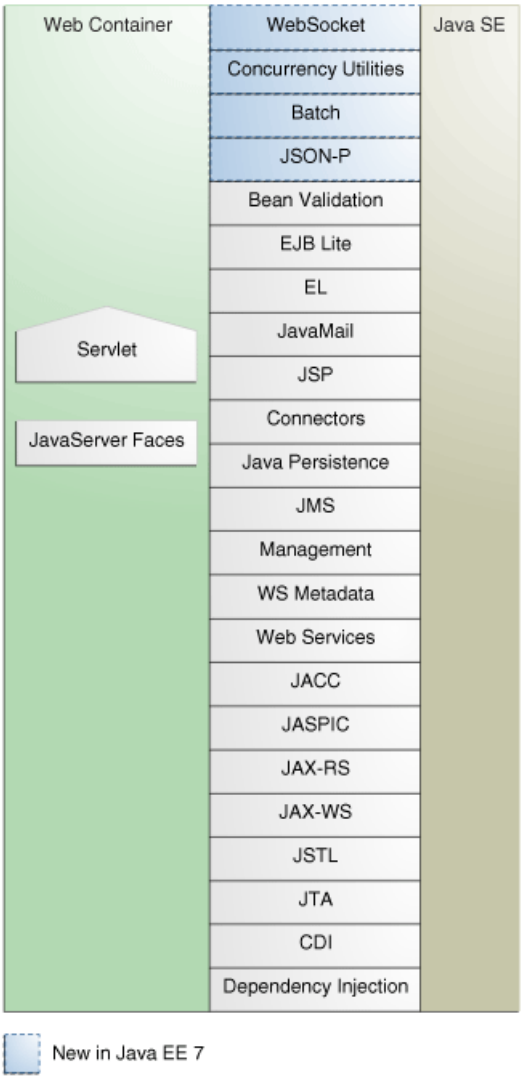
Na obrázku 2.4 můžeme vidět, které komponenty business vrstva dle specifikace platformy Java EE obsahuje.

2.2.4 Enterprise Information System (EIS) vrstva

Tato vrstva představuje veškeré externí systémy, jejichž funkcionalitu či data konkrétní aplikace využívá. Může se jednat například o databázový systém. Komunikaci s Enterprise Information System zpravidla zajišťuje business vrstva.

2.3 Oboustranná komunikace

Cílem této práce je navrhnout architekturu aplikace umožňující synchronizovat stav aplikace a grafický výstup pro větší množství klientů. V této kapitole budou představeny možnosti, kterými by v rámci platformy Java EE bylo možné stav aplikace synchronizovat.



Obrázek 2.3: Webový kontejner platformy Java EE. Převzato z [5]

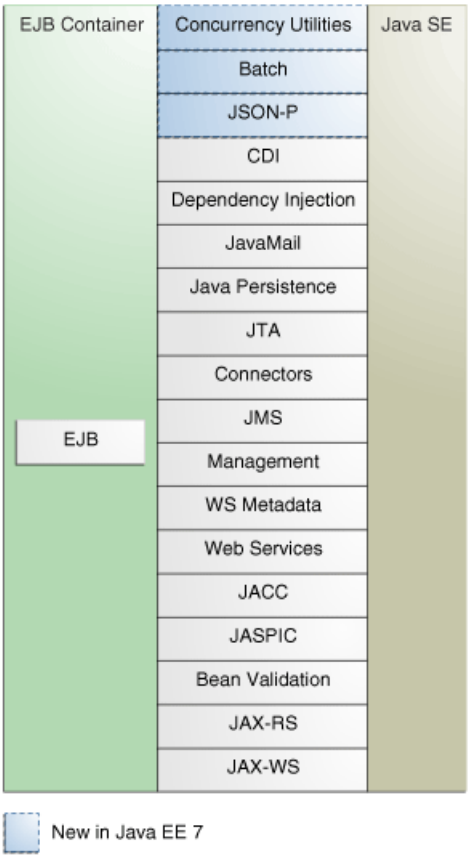
2.3.1 AJAX

AJAX je jeden ze způsobů, jak synchronizovat aplikace. Bohužel proces výměny dat mezi webovou stránkou a serverem probíhá podle HTTP protokolu. Protokol HTTP umožňuje pouze jednosměrnou komunikaci, kdy klient inicializuje spojení se serverem a zašle požadavek, který server následně zpracuje, odešle odpověď a ukončí spojení. V případě, že klientská část aplikace očekává data ze serveru, musí vždy vzniknout nejprve požadavek z její strany. Ten se buď v pravidelných intervalech opakuje (polling), nebo server udržuje spojení do doby, než bude mít požadovaná data k dispozici (long-polling). [10]

Tato technika je velice náročná na zdroje serveru, kdy pro každého klienta je nutné držet systémové prostředky a neustále navazovat HTTP spojení.

2.3.2 Primefaces push

Před specifikací Java EE 7 platforma neobsahovala přímou podporu pro WebSockets. Tuto technologii ale bylo a stále je možné využít s knihovnou značek Primefaces. Tato knihovna



Obrázek 2.4: EJB kontejner platformy Java EE. Převzato z [5]

Java Server Faces značek obsahuje podporu oboustranné komunikace se serverem pomocí rámce Atmosphere. Jedná se o rámec zajišťující komunikaci klient server v reálném čase.

Rámec Atmosphere je podporován většinou prohlížečů a podporuje techniky WebSockets, Server Sent Events (SSE), Long-Polling a HTTP Streaming. Je možná i komerční podpora. [22]

Rámec Atmosphere je v kombinaci s knihovnou značek Primefaces velice jednoduše nasaditelný do aplikace. V rámci analýzy možností pro tuto práci byla ale zjištěna nepříjemná vlastnost tohoto rámce, který je zabudován do knihovny Primefaces. Tento rámec neuvolňuje některé zdroje. To způsobí to, že časem dojde paměť aplikačního serveru. A celá aplikace se stane nefunkční do nutného restartu serveru, který uvolní systémové prostředky.

2.3.3 Websockets jako součást platformy Java EE 7

Platforma Java EE od své verze 7 poskytuje Java API pro WebSocket (JSR 356), které poskytuje podporu pro vytváření WebSocket aplikací. Websocket je aplikační protokol, který poskytuje plně duplexní komunikaci mezi dvěma uživateli přes protokol TCP.

V tradičním modelu požadavek a odpověď používaného v protokolu HTTP klient požaduje nějaké zdroje a server mu je poskytuje. Výměna informací je vždy iniciována klientem. Server nemůže odesílat data, aniž by klient nejprve požádal. Tento model funguje dobře pro webové stránky, kde klient žádá o dokumenty a server mu je poskytuje.

Protokol WebSocket řeší tyto nedostatky tím, že poskytuje plně duplexní komunikační kanál mezi klientem a serverem. V kombinaci s dalšími klientskými technologiemi, jako je JavaScript a HTML 5.

Tato práce bude postavena právě na technologii WebSocket.

Kapitola 3

Dostupné nástroje pro tvorbu klientských webových aplikací

Informace v této kapitole jsou čerpány z online specifikace HTML 5 [15] a knihy [4].

Interaktivní webové aplikace se postupem času vyvinuly ze statických dokumentů a jednoduchých formulářů, které vždy vyžadovaly obnovení celé stránky. Jedním z řešení problémů velkých aplikací bylo používat rámce (frames), které rozdělily okno webového prohlížeče na několik samostatných stránek, které bylo možné načítat odděleně. Později popularitu mezi interaktivním obsahem na webu zaznamenala platforma Adobe Flash, která přinesla vektorovou grafiku, pokročilé animace a nezávislost na načítání statických stránek. Hlavními nevýhodami byla chybějící HTML sémantika potřebná například pro vyhledávání a nutný zásuvný doplněk technologie Flash pro různé prohlížeče. Tyto nevýhody řeší platforma HTML 5, která poskytuje výhodu interaktivity a zároveň nabízí možnost tvorby sémantického webu. [6]

3.1 Adobe Flash Player

Technolige Adobe Flash Player se opouští a bude postupně nahrazena technoligí HTML 5. Například společnost Google oznámila postupné opouštění podpory technologie Flash ve svém prohlížeči Chrome od verze 55.

Google argumentuje tím, že přechod na technologii HTML 5 je pro uživatele výhodný. Je mnohem lehčí a rychlejší, rychlé načítání stránek také pomůže zvýšit výdrž baterie v zařízeních, na kterých webové stránky sledujeme. [11]

Přechod na technoligi HTML 5 je však postupný a Google s jeho prohlížečem Chrome bude ještě nějakou dobu technoligi Flash podporovat. Nebude však Flash ve stránkách spouštět automaticky a bude vyžadovat explicitní povolení od uživatele.

Jako Google se před lety zachovala i společnost Apple. V roce 2010 Steave Jobs napsal dopis [9], ve kterém sepisuje důvody proč zařízení iPad, iPod a iPhone nebudou podporovat technologii flash.

3.2 HTML 5

Technologie HTML 5 je nejnovější verze technologie HTML. V případě, že budeme mluvit o technoligii HTML, mluvíme obecně o značkovacím jazyce, ve kterém se tvoří webové stránky. HTML 5 je nejnovější specifikace tohoto značkovacího jazyka.

HTML 5 přináší celou řadu novinek. Například nové značky pro práci s formuláři nebo značky audio, video, svg, canvas. Také javascriptem dostupná API na geolokaci, drag & drop a další. Informace o HTML 5 byly čerpány z [4] a [15].

Při práci s HTML 5 je nutné rozlišovat dvě důležité vlastnosti. Jedna je to, co je napsané v specifikaci HTML 5. Druhá je to, co skutečně implementují jednotlivé verze prohlížečů. Tyto vlastnosti lze ověřit online pomocí služby CanIUse [19].

3.3 SVG

Jedná se o značkovací jazyk a formát souboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML. HTML 5 umožňuje vložit formát typu svg přímo do stránky [24].

Formát SVG je vektorová reprezentace grafiky, není tedy vhodný pro ukládání fotografií nebo složitějších obrázků. Velkou výhodou také zůstává možnost jednoduchého škálování při různých velikostech obrazovky.

SVG soubor, který je ve formátu XML, je možné jej vytvořit pomocí grafického vektorového editoru jako je například Inkscape [12], nebo přímo ručně v textovém editoru. Základním elementem struktury XML je rodičovský element ,`<svg>` který může přímo obsahovat grafické elementy `<circle>`, `<rect>`, `<polygon>` apod., nebo skupiny objektů `<g>` a `<layer>`.

3.4 Canvas

Prvek canvas sestává z regionu definovaném v HTML kódu šířkou a výškou, na který je možné dynamicky generovat grafiku. To může být využito například k vykreslování grafů, nebo vytváření animací.

K použití prvku `<canvas>` potřebujete základní znalosti HTML a JavaScript. Tento prvek není podporován v některých starších prohlížečích, ale je podporován v posledních verzích všech hlavních prohlížečů. Výchozí velikost plátna je 300 px x 150 pixelů (šířka x výška). [23]

Před možností použití prvku canvas pro kreslení, byl prakticky jediný způsob jak něco na stránce nakreslit, vytváření HTML elementů a jejich vhodné stylování pomocí JavaScriptu.

3.5 Javascript

JavaScript je multiplatformní skriptovací jazyk, který se zpravidla používá pro tvorbu dynamických webových stránek. Obvykle je využíván k ovládání různých prvků (tlačítka, textová políčka) nebo jsou pomocí něj tvořeny animace či celé dynamické aplikace.

V rámci této práce bude javascript využíván prostřednictvím javascriptové knihovny jQuery [7], která poskytuje širokou podporu prohlížečů a řeší za uživatele nekompatibilitu mezi jednotlivými prohlížeči.

3.6 WebSocket

Technologie WebSocket je jednou z nejdůležitějších inovací v HTML 5. Specifikace této technologie se skládá ze dvou standardů. Standard „W3C WebSocket API“ [25] a „IETF RFC 6455 Protocol“ [3]. Definované API umožňuje využít protokol WebSocket, pro obousměrnou komunikaci mezi samotnou webovou stránkou a vzdáleným serverem.

Technologie WebSocket umožňuje plně duplexní[1] komunikační kanál nad protokolem TCP. Díky tomu technologie WebSocket poskytuje enormní snížení zbytečného zatížení sítě a latence ve srovnání s technologiemi, které se pro simulaci plně duplexního spojení používaly v minulosti.

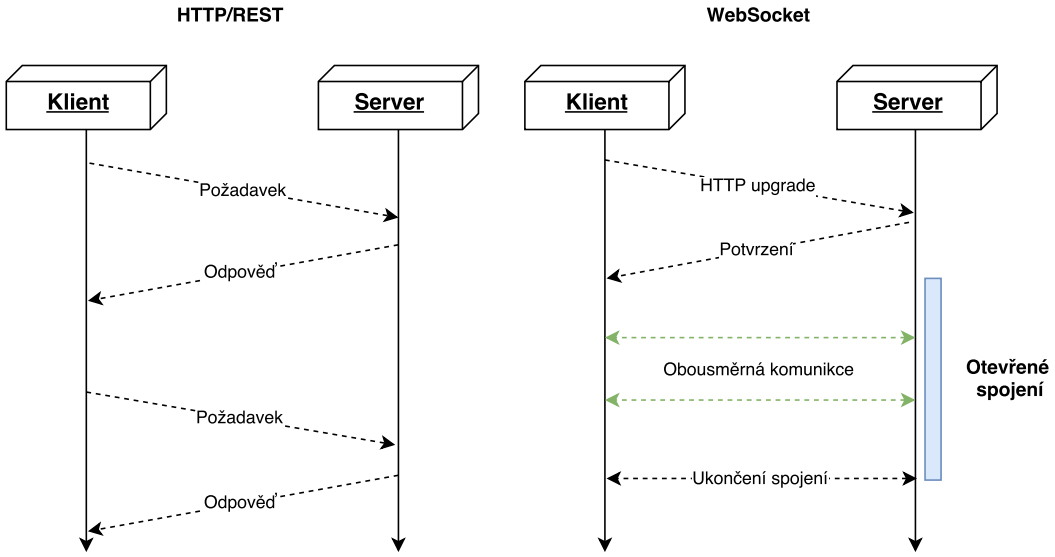
Protokol WebSocket a samotné API jsou blíže popsány v následujících podkapitolách.

3.6.1 Přehled protokolu WebSocket

Informace v této podkapitole jsou čerpány z dokumentu „IETF RFC 6455 Protocol“ [3] a knihy „The Definitive Guide to HTML5 WebSocket“ [8]

Z historickeho hlediska se při vytváření aplikací, které potřebují obousměrnou komunikaci mezi klientskou aplikací a serverem (například instant messaging nebo herní aplikace), zneužívalo spojení HTTP, které se opakovaným dotazováním serveru pokoušelo zjistit, zda na serveru došlo ke změnám. Toto zneužívání HTTP protokolu přinášelo celou řadu problémů. Server byl nucen neustále navazovat nová a nová spojení s klientem, což značně vytěžovalo síť a zvyšovalo zpoždění v komunikaci mezi klientskou a serverovou aplikací.

Alternativou k opakovanému HTTP dotazovaní vzdáleného serveru je nový protokol WebSocket, který v kombinaci s WebSocket API zvládne využít jediného TCP spojení pro provoz v obou směrech mezi klientem a serverem. Na obrázku 3.1 lze pozorovat rozdíl v komunikaci se serverem pro klienta, který se na změny dotazuje opakovaným HTTP dotazováním a druhý příklad, který využívá technologii WebSocket.



Obrázek 3.1: Rozdíl mezi HTTP spojením a WebSocket spojením

V případě HTTP dotazování je nezbytné aby, každá komunikace se serverem byla iniciována ze strany klienta. Klient je tedy nucen se periodicky dotazovat na server, zda pro něj nejsou nějaké nové dostupné informace. Server nedokáže samostatně navázat spojení s klientem. To by bylo velice vhodné v případě, že server obdrží nějaké nové informace pro doručení ke klientovi. Je tedy nutné implementovat mechanimus, kdy se klient periodicky dotazuje serveru. Toto dotazování je však mnohdy zbytečné, protože

[1] Možnost komunikovat v obou směrech spojení ve stejném čase

server pro klienta nemá žádné nové informace, které by mu poskytl. Pro každé toto dotazování je nutné, aby si klient se serverem vyměnil značné množství HTTP hlaviček, díky nimž naváže samotné spojení, které zbytečně vytěžují síť.

Technologie WebSocket řeší toto neustále opakováné navazování spojení. V pravé polovině obrázku 3.1 je zobrazeno, že v případě úspěšného navázání spojení klienta se serverem spojení zůstává otevřené a obě strany komunikační linky mohou libovolně odesílat zprávy do doby, než jedna ze stran spojení ukončí. Tento mechanismus je naprosto odlišný od původní technologie opakovaného dotazování a díky tomu, že nemusí neustále navazovat nová a nová spojení, značně snižuje množství informací odesílaných po síti. Doba dopravení informací ze serveru ke klienentovi je téměř okamžitá, protože server může tyto informace ihned odeslat a nemusí čekat na periodický požadavek ze strany klienta.

WebSocket protokol je stejně jako protokol TCP asynchronní a může sloužit jako vrstva pro protokoly vyšší úrovně. WebSocket spojení musí být vždy navazováno směrem od klienta k serveru a není možné, aby webový klienti přijímali spojení, které sami nenavázali. Protokol WebSocket vytváří most mezi světem Webu a světem Internetu, kdy asynchronní protokoly mohou komunikovat s webovými aplikacemi právě pomocí WebSocket protokolu.

Otevírací handshake

Protokol WebSocket má dvě části. Samotné navázání spojení a následný přenos dat. Následující příklad popisuje navázání spojení, ve kterém klient žádá server o otevření WebSocketu a server mu odpovídá.

Ukázka kódu 3.1: HTTP požadavek klienta na server, převzato z [8].

```
1 GET /echo HTTP/1.1
2 Host: echo.websocket.org
3 Origin: http://www.websocket.org
4 Sec-WebSocket-Key: 7+C600xYyb0v2zmJ69RQsw==
5 Sec-WebSocket-Version: 13
6 Upgrade: websocket
```

Ukázka kódu 3.2: HTTP odpověď ze serveru, převzato z [8].

```
1 HTTP/1.1 101 Switching Protocols
2 Connection: Upgrade
3 Date: Wed, 20 Jun 2012 03:39:49 GMT
4 Sec-WebSocket-Accept: fYoqiH12DqI+5y1EMwM2OLzOi0=
5 Server Kaazing Gateway
6 Upgrade: websocket
```

Na předchozím příkladu lze pozorovat, že WebSocket spojení obsahuje speciální hlavičku HTTP `Upgrade`, která říká, že klient chce přejít na jiný protokol, v tomto případě protokol WebSocket. V případě, že dotazovaný server tento protokol podporuje, odpoví hlavičkou HTTP 101. Aby bylo spojení úspěšně navázáno, musí server zaslat odpověď s vypočteným klíčem. Výpočet probíhá tak, že se vezme příchozí klíč, který se spojí s konstantním řetězcem a následně se vypočte hash pomocí algoritmu SHA-1, který se zakóduje pomocí base64.

Poté, co je spojení úspěšně sestaveno, mohou obě strany odesílat data nezávisle na straně druhé, tak jak potřebují. Pomocí protokolu může být přenášeno několik typů zpráv. Textové zprávy, které jsou přenášeny jako UTF-8. Binární data, jejichž inerpretace je ponechána a aplikaci a řídící zprávy, které nejsou určeny na přepravu dat pro aplikaci, ale pro samotný WebSocket protokol a slouží například k signalizaci uzavření WebSocketu.

Uzavírací handshake

Specifikace protokolu WebSocket definuje celou řadu číselných kódů, definujících důvod uzavření WebSocketu. Pomocí těchto kódů se obě strany dozvědí, proč došlo k uzavření spojení. Číselný kód standardního uzavření, který indikuje, že účel proč byl WebSocket otevřen byl splněn, má číslo `1000`. Ostatní kódy uzavření jsou definovány ve specifikaci protokolu WebSocket [3] v kapitole 7.4.

Možnost rozšíření protokolu

Protokol WebSocket je rozšiřitelný. Rošíření protokolu si může vyžádat pouze klient, pokud toto rozšíření server podporuje tak, je použito. Rozšíření jsou definována ve speciálních specifikacích. V žádném případě se nemůže stát, že server použije rozšíření, které si klient nevyžádal. Díky tomu lze mechanismus rozšíření ve specifikaci protokolu WebSocket udržet jednodušší.

Například rozšíření definující kompresi je specifikováno v dokukumentu „Compression Extensions for WebSocket“ [26].

Podpora protokolů vyšší vrstvy

Specifikace WebSocket podporuje protokoly vyšší vrstvy pomocí takzvaných „Sub protokolů.“ Jeden z příkladů je návrh specifikace „An XMPP Sub-protocol for WebSocket“ [27].

WebSocket bezpečnost

Pro zabezpečení protokolu WebSocket lze používat WebSocket Secure, jedná se o normální WebSocket, který běží nad TLS[2]. Všechna komunikace je šifrována. WebSocket Secure používá standardní port 443, stejně jako HTTPS (HTTP protokol nad TLS).

WebSocket latence

Protokol Websocket výrazně snižuje nutnost posílání dalších dat, jako jsou HTTP hlavičky nutné pro neustálé otevírání a zavírání nových spojení v případě komunikace pomocí technologie AJAX. Dalo by se tedy předpokládat, že bude možné posílat podstatně více zpráv za stejný čas.

Malý test [2], který v roce 2012 provedl Peter Bengstoon, ale ukazuje výrazné zvýšení rychlosti pouze na lokálním stroji, kde se mu podařilo dosáhnout šestinásobnému zvýšení propustnosti. V případě pokusu mezi Angilí a Kalifornií znamenalo použití protokolu WebSocket pouze 10% zlepšení.

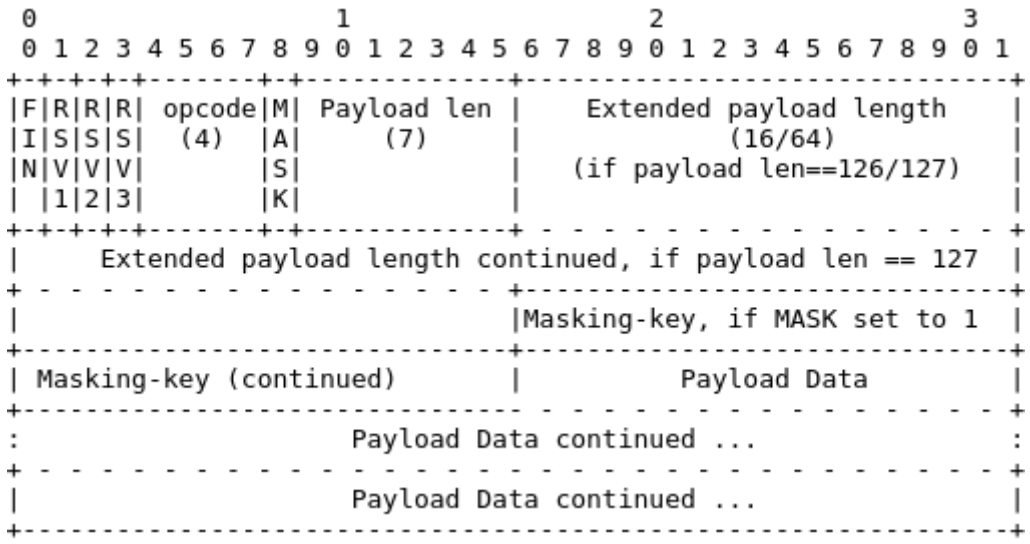
Rámce protokolu WebSocket

Datové rámce ve kterých jsou posílány zprávy protokolu WebSocket obsahují na rozdíl od protokolu HTTP jen velice malé množství režijních informací. Toto množství informací je v řádu bajtů. Jednotlivé položky můžeme vidět na obrázku 3.2, který zobrazuje strukturu rámce ze specifikace protokolu.

Datové rámce se dokáží dále fragmentovat - dělit na menší části. Fragmentace také umožňuje okamžité posílání zpráv u kterých ještě není známá celá jejich délka. Mezi

[2] Transport Layer Security

fragmenty jedné zprávy je možné zasílat fragmenty jiných zpráv. Díky tomu je možné v případě, že bude odesílána jedna příliš dlouhá zpráva, odesílat i jiné menší zprávy ve stejném směru spojení. V případě, že by tomu takto nebylo, obsadila by dlouhá zpráva spojení a ostatní by byly odeslány až ve chvíli, kdy by byla dlouhá zpráva celá odeslána a přišly by tak se zpožděním.



Obrázek 3.2: Ukázka úspěšného připojení klienta ke skupině

Každý paket protokolu WebSocket obsahuje záhlaví, které je zobrazeno na obrázku 3.2. V následujících odstavcích jsou popsané jednotlivé položky, které hlavička protokolu WebSocket obsahuje

fin (1 bit): indikuje, zda tento rámec je poslední snímek, který tvoří zprávu. Většinou se zpráva vejde do jednoho snímku a v těchto případech bude zpráva první a zároveň poslední. Proto je jeho hodnota ve většině případů nastavena na 1.

rsv1, rsv2, rsv3 (1 bit pro každý): Tyto bity jsou momentálně nevyužité.

opcode (4 bity): Operační kód specifikuje typ dat přenášených WebSocketem. Přes WebSocket můžeme posílat textová, binární data a jiné typy zpráv, které jsou řídícího charakteru. Například uzavírací handshake, který je posílán při ukončování spojení.

V případě, že dlouhou dobu nejsou posílány žádné zprávy se může stát, že některé síťové prvky (proxy servery nebo problematické routery) budou uzavírat navázané spojení. Tomu WebSocket předchází zasláním jiných řídících rámců, které jsou typu `Ping` a `Pong`. Po odeslání `Ping` přichází odpověď `Pong`. Volání `Ping` může být provedeno z webového prohlížeče, stejně jako ze serveru. Tato řídící volání, která se snaží udržet perzistentně otevřené spojení, nejsou obsažena v aplikačním rozhraní na straně klienta (WebSocket API). To, jak často je volán `Ping` z klienta, je plně v režii webového prohlížeče a liší se dle implementace samotného webového prohlížeče.

Následující hodnoty operačního kódu jsou v současné době obsaženy v protokolu WebSocket:

- **0x00**: Tento typ rámce navazuje na předchozí rámec,
- **0x01**: tento typ rámce obsahuje utf-8 textová dat,
- **0x02**: tento typ rámce obsahuje binární data,

- **0x08**: tento typ rámce ukončí spojení,
- **0x09**: tento typ rámce je ping,
- **0x10**: tento typ rámce je pong.

Na soupisu číselných kódů lze pozorovat, že některé číselné hodnoty jsou vynechány a jsou tak přípraveny pro budoucí použití.

Mask (1 bit): Maska indikuje, zda je spojení maskováno (dále v textu). Každá zpráva od klienta na server musí být maskována, pokud není, server musí ukončit spojení.

payload len (7 bitů): hodnota, která nám řekne, jak velká bude zpráva.

- `Hodnota 0-125` znamená délku zprávy.
- `Hodnota 126` znamená, že následující dva bajty označují délku zprávy.
- `Hodnota 127` znamená, že dalších 8 bajtů označuje délku zprávy.

Masking-key (32 bitů): Maskovací klíč se Nastavuje v případě, že je nastaven maskovací bit. Klíč se mění po každém odeslání rámce. Toto maskování nezajišťuje šifrování, pouze znemožňuje útok typu „Cache poisoning.“ Pro potřeby šifrování je nutné použít WebSocket Secure.

Payload Data: Skutečná data o velikosti, kterou zjistíme díky hodnotě umístěné v `payload len`.

3.6.2 WebSocket API

Jedná se o rozhraní, které je jednoduché a vhodně odděluje programátory od WebSocket protokolu. WebSocket API je řízené událostmi, kdy klient naslouchá notifikacím a změnám dat. Specifikaci WebSocket API [25] navrhuje pracovní skupina Web Hypertext Application Technology Working Group (WHATWG5).

WebSocket konstruktor

Úkolem konstruktoru je na základě předaných parametrů vytvořit připojení k serveru a vrátit nový objekt WebSocket, se kterým může programátor dále pracovat. Po zavolání WebSocket konstruktoru s parametrem URL, které reprezentuje koncový bod na serveru, se začíná sestavovat spojení a je vrácen WebSocket objekt. WebSocket konstruktor obsahuje ještě druhý volitelný parametr. Tento parametr se využívá v případě, že se serverem chceme komunikovat pomocí nějakého specifického protokolu. Pokud server tento protokol podporuje, spojení s námi bude naváznano.

Ukázka kódu 3.3: Příklad základního použití konstruktoru

```
1 var myWebSocket = new WebSocket( "ws://www.websockets.org" );
```

Ukázka kódu 3.4: Použití konstruktoru s volitelným speciálním protokolem

```
1 var websocket = new WebSocket ( "ws://www.websockets.org" , "special_protocol" );
```

Události

WebSocket API je řízené událostmi. Před navázáním spojení může programátor přiřadit posluchače událostí tak, jak je uvedeno v následujícím příkladu. V příkladu se zprávy pouze logují do vývojářské konzole prohlížeče, programátor tedy v konzoli přesně uvidí jaká událost byla vyvolána, případně obsah zprávy, která byla doručena.

Ukázka kódu 3.5: Události WebSocket API

```
var myWebSocket = new WebSocket("ws://www.websockets.org");

myWebSocket.onopen = function(evt) {
    console.log("Spojení bylo úspěšně otevřeno ...");
};
myWebSocket.onclose = function(evt) {
    console.log("Spojení bylo uzavřeno.");
};
myWebSocket.onerror = function(evt) {
    console.log( "Nastala událost onError: " + evt);
};
myWebSocket.onmessage = function(evt) {
    console.log( "Byla přijata zpráva: " + evt.data);
};
```

Událost onOpen: Tento typ události nastává poté, co je úspěšně sestaveno spojení. Je důležité tuto událost odchytávat a dbát na to, aby byly zprávy odesílány nejdříve poté, co je spojení úspěšně navázáno. Pokud pošleme zprávu dříve, než je spojení sestaveno, dojde k vyvolání chyby.

Událost onClose: V případě, že dojde k uzavření spojení, je volána událost `onClose`. V případě pokračování v odesílání zpráv po přijetí této události, budou odesíláné zprávy vyvolávat chybu. Událost `onClose` obsahuje několik atributů, které programátorovi mohou blíže popsat, proč k uzavření došlo. Jedná se o atributy `wasClean`, `code` a `reason`. Atribut `wasClean` je nastaven na hodnotu `true` v případě, že bylo spojení WebSocketu uzavřeno korektně. Pokud k uzavření došlo z jiného důvodu (například bylo uzavřeno TCP spojení), bude v tomto atributu hodnota `false`. V atributu `code` bude uložen číselný kód uzavření podle 3.6.1.

Událost onMessage: Velké zprávy mohou být rozděleny do více datových rámců. Událost `onMessage` je volána až ve chvíli, kdy je doručena celá zpráva.

Událost onError: V případě, že dojde k nečekaným chybám, je volána událost `onError`. Po této chybě dochází k okamžitému uzavření spojení. Toto je místo, kde je vhodné implementovat logiku, která se v případě, že to je možné, pokusí o nové navázání spojení.

Metody WebSocket API

WebSocket API definuje pouze dvě záladní metody. Metodu pro odeslání dat `send()` a metodu pro zavření spojení `close()`. Pokud chceme odeslat zprávu na server, stačí zavolat na metodu `send("Obsah Zprávy")`, kterou zavoláme s obsahem, který chceme na server doručit. V případě, že je již komunikace mezi klientem a serverem dokončena, stačí zavolat metodu `close()`. Metoda close má dva volitelné argumenty, které se znamenají stejé hodnoty jako `code` a `reason` v události `onClose`.

Ukázka kódu 3.6: Metody WebSocket API

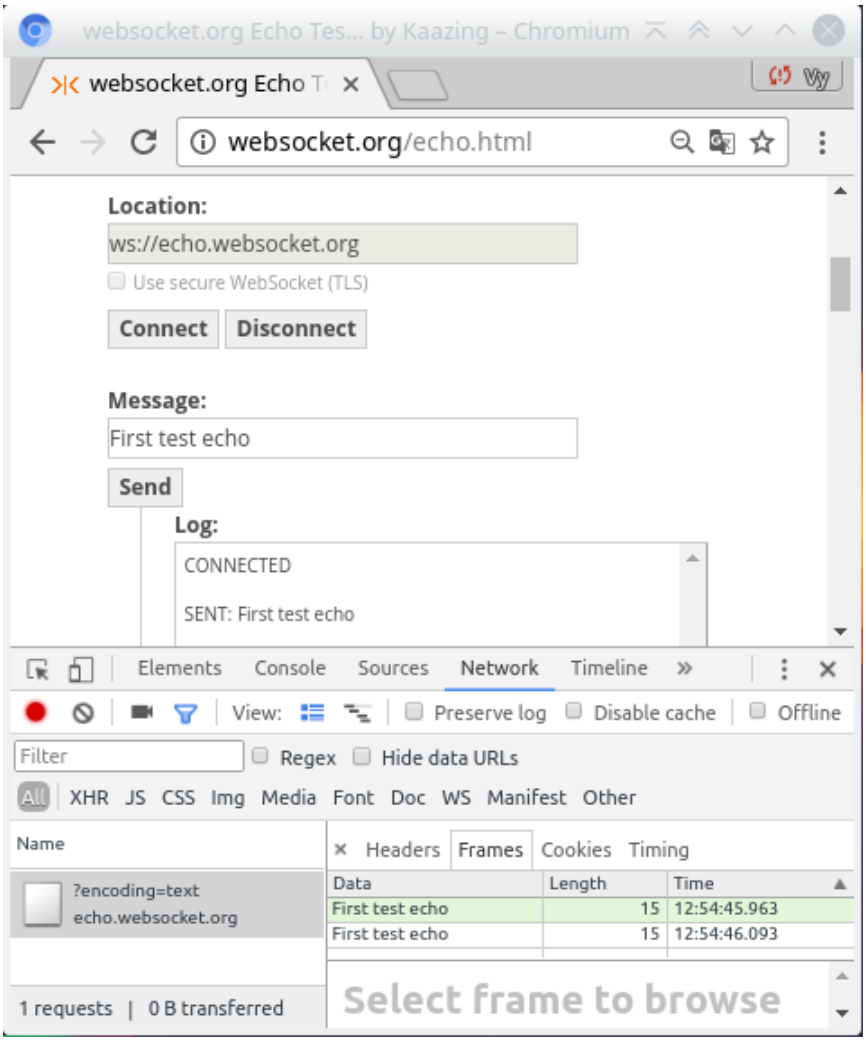
```
myWebSocket.send("Obsah Zprávy");
```

```
2 myWebSocket.close();
```

Volání obou metod je možné až po tom, co je spojení skutečně navázáno. Indikaci toho, že je spojení navázáno nám poskytuje vyvolání události `onOpen`. Poté, co je událost `onOpen` zavolána, je spojení úspěšně navázáno. Jak bylo možné pozorovat na předchozích příkladech, zpracování zpráv ze serveru a jejich odesílání zpráv na server, je velice jednoduché a bude názorně použito v ukázkové aplikaci.

Debugování komunikace přes WebSocket

To jaké zprávy jsou po síti pomocí protokolu WebSocket posílány, lze pozorovat například pomocí nástroje Chrome Developer Tools 3.3, který je standardní součástí webového prohlížeče Chrome.



Obrázek 3.3: WebSocket zprávy v nástroji Chrome Developer Tools

Kapitola 4

Návrh architektury aplikace

V této kapitole bude rozebrán obecný návrh aplikace. Bude popsán základní princip serverové část aplikace a základní vlastnosti výměny zpráv mezi serverem a klienty. Implementační detaily obecného návrhu jsou blíže popsány v kapitole 5, která se detailněji zabývá demonstrační aplikací.

Bude se jednat o obecný návrh aplikacem, který je zobrazen na obrázku 4.1. Klienti se budou obecně moci připojit k datové struktuře na straně serveru, který bude udržovat skupinu klientů a zajišťovat, že všichni připojení klienti k této struktuře budou mít synchronizovaný stav aplikace i grafický výstup. Těchto struktur na straně serveru může být obecně více. Bude tedy možné mít jedinou synchronizovanou skupinu klientů nebo větší množství synchronizovaných skupin klientů. To, zda bude jedna nebo více, se bude odvíjet od návrhu konkrétní aplikace.

Všichni klienti připojení ke konkrétní skupině budou moci stav této konkrétní skupiny ovlivňovat pomocí zpráv odesílaných z klientské aplikace na server. Ze serveru budou odesílány zprávy, které budou zajišťovat, že všechny klientské aplikace připojené ke konkrétní skupině budou synchronizované.

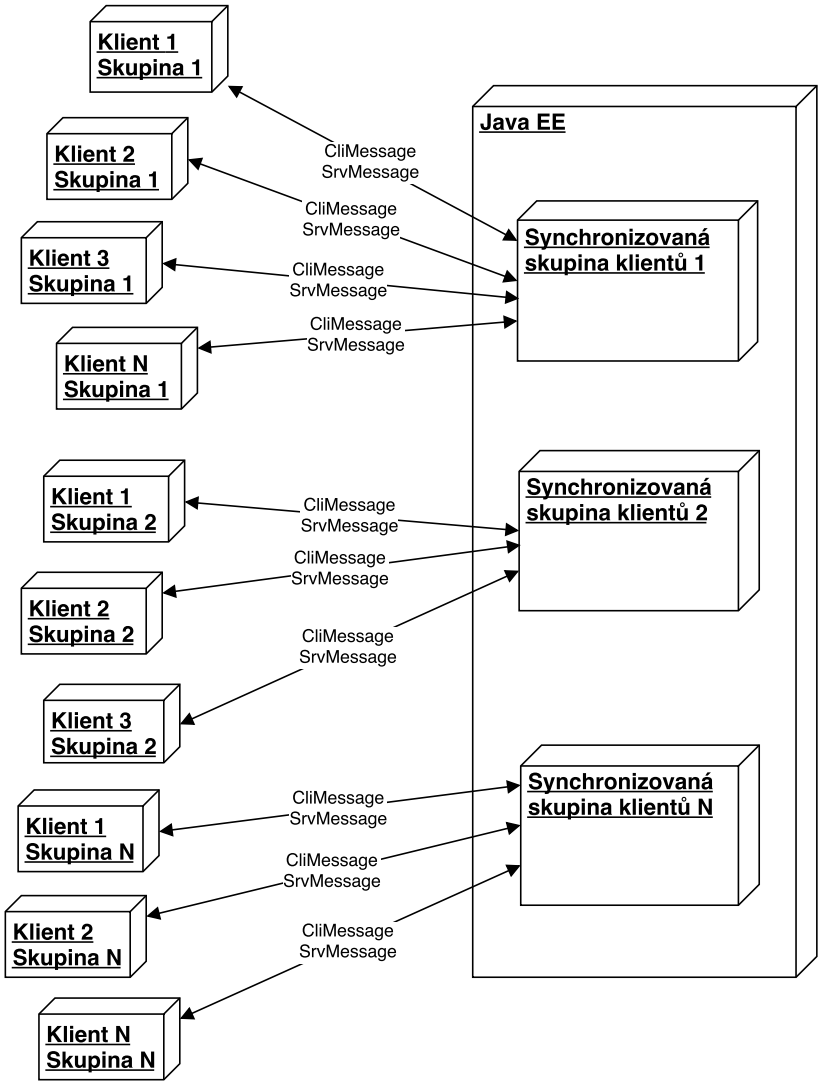
Aplikace bude webová. To, ke které skupině na serveru se chce klient připojit lze určit například pomocí html prvku `SELECT`, který bude nabízet identifikátory konkrétních skupin, ke kterým je možné se připojit.

4.1 Serverová část aplikace

Serverová část aplikace bude napsána na platformě Java EE. Aplikace bude pracovat v EJB kontejneru, který nám zajistí správu instancí a řízení transakcí. S aplikací budou klienti komunikovat pomocí zpráv `CliMessage` a `SrvMessage`, které budou po síti mezi klientem a serverem přenášeny ve formátu `JSON` [16]. K jednotlivým operacím v serverové části aplikace bude moci přistupovat větší množství klientů, je tedy nutné zajistit výlučný přístup klientů. Toho docílíme použitím zámků nad kritickými sekcemi. Serverová část aplikace bude udržovat větší množství klientů ve skupinách, kterým bude zajišťovat synchronizovaný stav grafického výstupu pomocí rozesílání zpráv `SrvMessage` ze serveru do klientské aplikace.

4.2 Klientská část aplikace

Klientská část aplikace je ta část, u které musíme zajistit, aby byl její stav synchronizovaný pro všechny připojené klienty. Tato část bude napsána z velké části



Obrázek 4.1: Propojení klientů a synchronizovaných skupin

v javascriptu. Javascriptová aplikace bude reagovat na zprávy `SrvMessage`, které přicházejí po WebSocketu ze strany serveru. Na základě těchto zpráv se bude měnit stav aplikace na straně klienta. Ze strany klienta do serverové části aplikace budou odesílány zprávy `CliMessage`. Tyto zprávy mohou ovlivňovat stav aplikace na straně serveru. V případě, že zpráva ze strany klienta ovlivní stav aplikace na straně serveru, bude tato změna okamžitě rozeslána ostatním klientům v podobně zpráv `SrvMessage`, které budou klienty informovat o změnách ve stavu na serveru. Klienti zpracováním této zprávy zajistí úpravu grafického výstupu aplikace tak, aby byl u všech klientů stejný. Tím bude docíleno toho, že grafický výstup aplikace konkrétní skupiny klientů bude pro všechny připojené klienty synchronizován.

4.3 Komunikace mezi serverovou a klientskou částí aplikace

Zprávy jsou z důvodu přehlednosti rozděleny na dva typy. Typ `SvrMessage`, který bude odesílán ze serverové části ke klientům. A typ `CliMessage`, který bude odesílán z klientské

části na server. Tyto zprávy bude možné logovat v Javascriptové konzoli prohlížeče klientské stanice. Tento log bude užitečný při ladění aplikace a bude z něj možné pochopit, jak jednotlivé části aplikace fungují.

4.3.1 Třída zpráv CliMessage

Ukázka kódu 4.1: Datová struktura CliMessage

```
public class CliMessage {

    //identifikátor klienta který zprávu posílá
    //na serveru se kontroluje, zda má klient opravnění odesílat zprávu do
        konkrétní skupiny
    private String srcClient;

    //pokud se zpráva váže ke konkrétní skupině, je zde identifikátor skupiny
    //některé zprávy se nemusí vázat k žadné skupině, pak je zde null
    private String srcGroup;

    //typ zprávy
    private int msgType;

    //konkrétní typy zpráv tuto třídu rozšiřují

    //kontruktor, gettery, settery
}
```

4.3.2 Třída zpráv SvrMessage

Ukázka kódu 4.2: Datová struktura SvrMessage

```
public class SvrMessage {

    //časové razítko, kdy byla zpráva odeslána
    //na straně klienta v javascriptu slouží k internímu počítání odchylky času
        prohlížeče
    private long ts;

    //identifikátor klienta pro kterého je zpráva určena
    protected String dstClient;

    //na serveru může současně existovat více synchronizovaných skupin klientů
    //je tedy nutné jasně identifikovat skupinu, které zpráva patří
    protected String dstGroup;

    //typ zprávy
    private int msgType;

    //konkrétní typy zpráv tuto třídu rozšiřují

    //kontruktor, gettery, settery
}
```

4.4 Připojování klientů k serverové části

Pro připojení klienta ke skupině je třeba znát unikátní tajný identifikátor klienta a unikátní identifikátor skupiny. Identifikátor klienta se vytváří při zakládání sezení a je platný po celou dobu sezení. Tento indentifikátor je tajný a nesmí se jej dozvědět nikdo, kromě samotné klientské aplikaci, které patří, a serveru. Klientský identifikátor se používá jako součást

URL pro navázání WebSocketu a pro identifikaci klienta na straně serveru. Klient načte ve svém webovém prohlížeči webovou stránku, na které klientská aplikace běží, a připojí se k WebSocketu, který je definován právě pomocí tohoto tajného identifikátoru. Na tomto WebSocketu je klientská aplikace připojena po celou dobu, kdy běží.

4.5 Důležité datové struktury aplikace

`ClientData`, `ClientDataInfo`, `GroupData` a `GroupDataInfo` jsou důležité datové struktury celého návrhu aplikace. Tyto struktury jsou záměrně pojmenovány obecným názvem, protože název této struktury může být v aplikacích s různým zaměřením odlišný. Například v herním typu aplikace se tyto struktury mohou jmenovat jmenovat PlayerData, PlayerDataInfo, GameData a GameDataInfo, a tím usnadnit čitelnost zdrojových kódů aplikace.

Struktura `ClientData` udržuje veškeré informace o klientovi na straně serveru. Obsahuje položku `privateId`, která je unikátním tajným identifikátorem klienta, který nesmí být nikdy sdělen ostatním klientům. V případě, že by se tento identifikátor dozvěděl někdo jiný než jeho vlastník, tak jej může použít pro podvrhávání zpráv a vydávat se za vlastníka. Položka `privateId` tedy nikdy nesmí v žádné zprávě opustit serverovou část aplikace.

Ukázka kódu 4.3: Datová struktura ClientData

```
1  public class ClientData {
2
3    //publicId klienta
4      String privateId;
5
6      //publicId klienta
7      String publicId;
8
9      //přezdívka klienta pro výpis v grafickém prostředí
10     String nickname;
11
12     //a další informace závislé na konkréním typu aplikace
13 }
```

Pokud potřebujeme posílat infomace o klientech po WebSocketu mimo serververovou část aplikace, použijeme Datavou Strukturu `ClientDataInfo`, která se vytváří ze struktury ClientData, avšak neobsahuje všechny položky. Struktura ClientData může obsahovat desítky položek, které jsou nutné pro samotné fungování aplikace na straně serveru, ale nepotřebné v klientské části aplikace. Tím, že použijeme jednodušší strukturu, kterou budeme posílat po WebSocketu v serializované podobě ve formátu `JSON`, také snížíme datovou náročnost celé aplikace.

Ukázka kódu 4.4: Datová struktura ClientDataInfo

```
1  public class ClientDataInfo {
2
3      //publicId klienta
4      String publicId;
5
6      //přezdívka klienta pro výpis v grafickém prostředí
7      String nickname;
8
9      //a další informace závislé na konkréním typu aplikace
10
11     //kontruktor, gettery, settery
```

```
}
```

`GroupData` a `GroupDataInfo` fungují na stejném principu jako ClientData a ClientDataInfo. GroupData je datová struktura, která sdružuje skupiny klientů, kteří budou mít synchronizovaný grafický výstup prohlížeče. Navíc obsahuje spoustu položek nutných pro běh aplikace na straně serveru. Struktura GroupDataInfo je její zjednodušená verze, která obsahuje pouze položky nutné pro správnou funkci aplikace v klientské části aplikace.

Ukázka kódu 4.5: Datová struktura GroupData

```
public class GroupData {

    //identifikátor hry
    String hash;

    //seznam klientů, kteří jsou připojeni k této skupině
    List<ClientData> clientDataList;

    //a další informace závislé na konkrétním typu aplikace

    //kontruktor, gettery, settery
}
```

Kapitola 5

Demonstrační aplikace

Tato kapitola pojednává o demonstrační aplikaci, která byla vytvořena v rámci diplomového projektu. Aplikaci jsem pracovně pojmenoval názvem ANET. Jedná se o aplikaci, která je postanena na obecném návrhu aplikace z kapitoly 4 a má za cíl ukázat použití technologie WebSocket v kombinaci technolotí Java EE a HTML 5. Po dohodě s vedoucím mé diplomové práce jsme se dohodli na herním typu aplikace, která bude zmíněné technologie plně využívat.

Při popisu demonstrační aplikace budou velice často použity tyto pojmy:

- `Hráč` - jedná se o klienta (klientskou část aplikace), která je pomocí WebSocketu připojena k serverové části aplikace.
- `Můj Hráč` - jedná se o hráče, kterého jsme schopni ovládat ovládat a dávat mu příkazy pomocí myši nebo klavesnice.
- `Robot` - jedná se o hráče, který není připojen pomocí WebSocketu ke klientské části aplikace a jeho chování se nastavuje v serverové části aplikace. Jeho pohyb po mapě a používání příkazů je náhodné. Díky použití robotů je možné si hru vyzkoušet i bez nutnosti připojení více hráčů do jedné hry.
- `Hra` - jedná se o jednu synchronizovanou skupinu klientů na straně serveru.

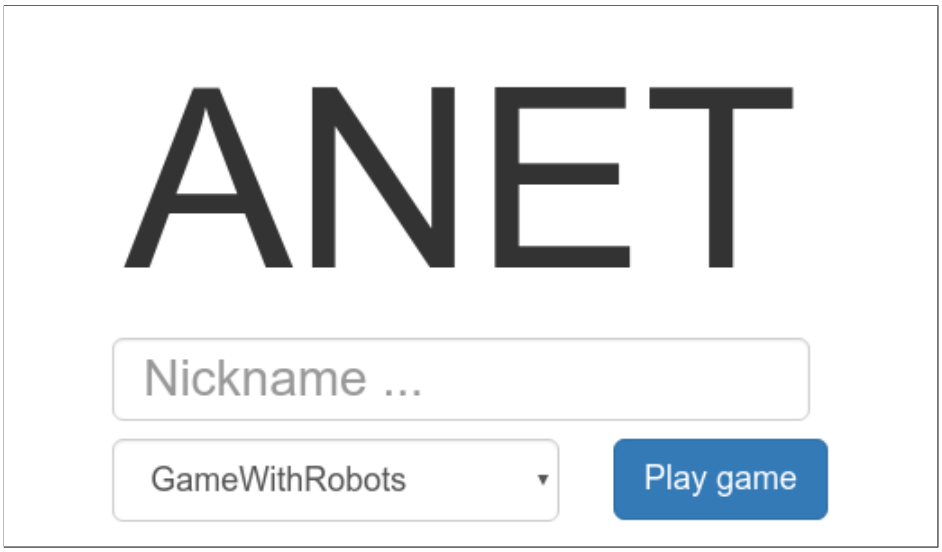
5.1 Princip aplikace

Princip aplikace je velice jednoduchý. Existuje skupina hráčů, kteří jsou připojeni ke stejné hře. Hra samotná se odehrává na určitém obdelníkovém území, ve kterém je možné se pohybovat pohybem myši ve směru od našeho hráče. Každý hráč na mapě je zobrazen jako kulička s textem uvnitř kuličky, který obsahuje jméno hráče a počet bodů, které ve hře získal. Hráči, kterého ovládáme, je možné udělovat příkazy pomocí stisku specifických kláves na klávesnici. Jedná se o příkazy:

- `zrychlit` - Hráči je zvýšena rychlost, kterou se může pohybovat po herní ploše.
- `explodovat` - Poloměr kuličky hráče na mapě se na určitou dobu zvýší a hráči, kteří jsou v době exploze v dosahu kuličky jsou pohlceni.
- `skrýt se` - Hráč se skyrje ostatním a není možné jej po dobu skrytí pohltit expolozí.

Hráči soupeří v počtu bodů, které nasbírají. Počet bodů které hráč získá závisí na tom, kolik protihráčů se mu během hry pomocí explozí podaří pohltit. Například pokud náš hráč při explozi pohltí 3 nové soupeře, jeho bodový zisk se zvýší o 3 body. Dále každý hráč, disponuje vlastností `energie`, která se dobíjí v závislosti na čase. Pokud chceme aby náš hráč vykonal nějaký příkaz, musí mít dostatečnou velikost této hodnoty. V případě, že nemá dostatek energie, příkaz se neprovede. Tato hodnota se vynuluje při každém příkazu. A její nabytí zpět závisí na čase a je konfigurovatelné pro každý typ příkazu.

5.2 Ukázka aplikace

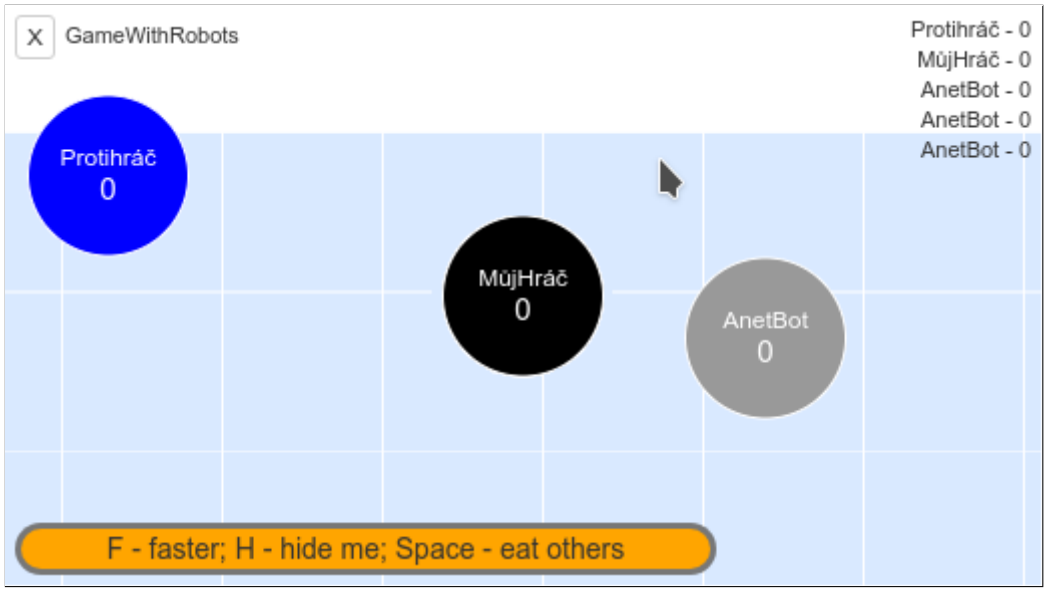


Obrázek 5.1: Úvodní stránka demonstrační aplikace

Na obrázku 5.1 je zobrazená úvodní stránka aplikae, jedná se o velice jednoduchou stránku, kde je pouze pracovní název demonstrační aplikace, vstupní pole pro zadání přezdívky a HTML prvek s nabídkou dostupných her. V demonstrační aplikaci jsou pouze dvě hry. Jedna s předdefinovanými roboty a druhá bez robotů. Aplikace je ale navržena tak, že by šlo velice jednoduše přidat další hry a dělit je například na veřejné a soukromé. Veřejné by byly v tomto seznamu na úvodní stránce a kdokoli by se k nim mohl připojit, a soukromé by měly vlastní URL, kterou by tvůrce rozeslal kamarádům, se kterými by si chtěl zahrát.

Na obrázku 5.2, kde je zobrazen záznam obrazovky z demonstrační aplikace. Na záznamu obrazovky lze pozorovat několik vlastností této hry. Hra na klientské stanici je vždy roztažena na celou obrazovku. Ale obdélníková herní oblast, ve které se hráči mohou pohybovat, může mít odlišné rozměry. Rozměry herní plochy pro každou hru jsou definovány v datové struktuře `GameInfo`, která obsahuje veškeré informace o hráčích a hře samotné na straně serveru. Na obrázku lze pozorovat několik oblastí, které si nyní blíže popíšeme:

- `Můj hráč` - hráč, kterého ovládáme má černou barvu, a je umístěn vždy ve středu okna prohlížeče.
- `Herní plocha` - je oblast znázorněna modrým pozadím a může být obecně větší než okno prohlížeče. Na herní ploše je vykreslena mřížka, aby byl jasně patrný pohyb po



Obrázek 5.2: Ukázka z demonstrační aplikace

herní ploše. Na obrázku je vidět bílá oblast nad herní plochou. To je oblast kam se žádný z hráčů nemůže přesunout, protože to již není herní oblast. Pokud se pokusí přesouvat tímto směrem, zůstane zablokovaný na okraji herní plochy. Herní plocha se překresluje na pozadí okna prohlížeče tak, aby náš hráč byl vždy zobrazen na správném místě herní plochy a zároveň uprostřed okna prohlížeče.

- `Ostatní protihráči` - ostatní protihráči se dělí na modré kuličky, které reprezentují ostatní hráče (klientské aplikace) připojené pomocí WebSocketu do stejné hry a roboty, kteří mají šedou barvu a jejich chování je náhodně generováno na straně serveru.
- `Tlačítko pro opuštění hry` - tlačíko v levém horním rohu, na které když hráč klikne, tak opustí aktuální hru a dostane se zpět na úvodní obrazovku.
- `Název hry` - vedle tlačíka pro opuštění synchronizované skupiny je název skupiny, který odpovídá názvu, který jsme si vybrali na úvodní stránce aplikace.
- `Žebříček hráčů` - v pravém horním rohu se objevuje žebříček 10 nejlepších hráčů. Tento seznam je aktualizovaný při každém připojení hráče, odpojení hráče nebo úspěšné explozi, kdy je pohlcen jiný hráč a někomu se tak změní počet bodů. Pouze deset hráčů se zobrazuje z důvodu možnosti připojení velkého množství klientů. Velké množství připojených klientů by znamenalo značnou velikost tohoto seznamu a pro spoustu znich by to znamenalo, že by na stránce vůbec nebyli vidět. Navíc by se po síti přenášelo velké množství informací, které by ve výsledné klientské aplikaci z tohoto důvodu nebylo vůbec zobrazeno. Proto je tento žebříček omezen pouze na 10 nejlepších hráčů.
- `Energie` - v levé dolní části herní aplikace je zobrazen panel, který má žluté pozadí. Tento panel slouží jako indikátor energie. V případě, že se provede nějaká akce jako například na obrázku 5.3, energie je použita a postupně se v čase doplňuje. Ve chvíli, kdy je energie opět na 100% své hodnoty, může hráč opět použít akci.

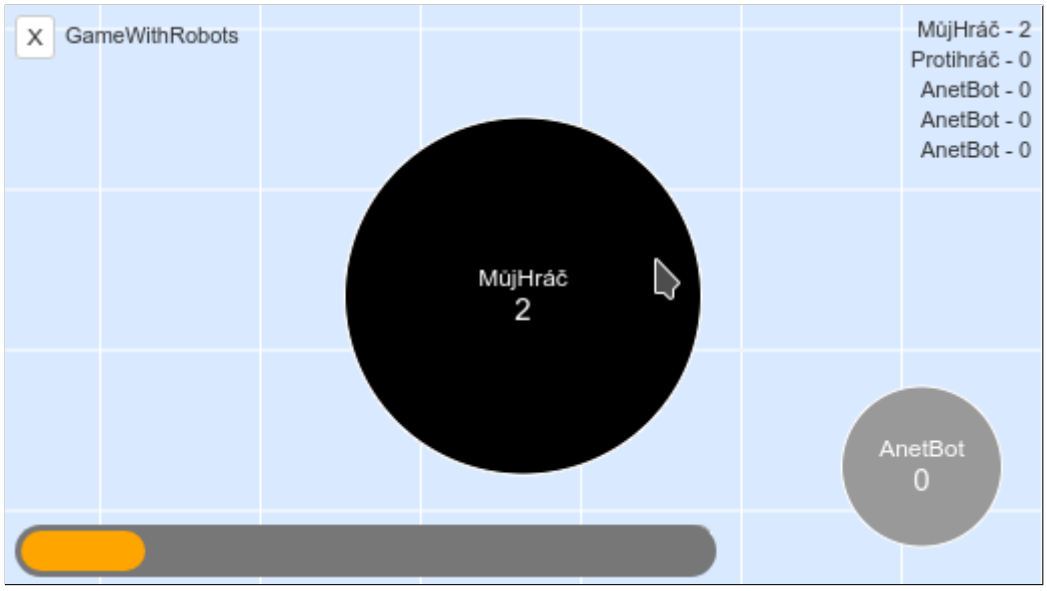
5.3 Ovládání aplikace

K ovládání aplikace slouží myš a klávesnice. Pohybem myši od středu obrazovky prohlížeče, kde je umístěn náš hráč, měníme směr pohybu hráče. Pokud myší zajedem do prostoru kuličky, která reprezentuje našeho hráče, tak se hráč nepohybuje. Dále je možné používat akce hráče, které jsou v demonstrační aplikaci reprezentovány klavesami:

- `Mezerník` - žádost o explozi hráče,
- `klávesa písmene S` - žádost o skrytí hráče,
- `klávesa písmene F` - žádost o vyšší rychlost hráče.

Záměrně je uvedeno, že se jedná o žádosti. Protože to, zda se akce provede se vyhodnocuje na straně serveru, kde se zkontroluje, zda má hráč právo a prostředky tuto akci provést.

5.3.1 Exploze a pohlcení protihráče

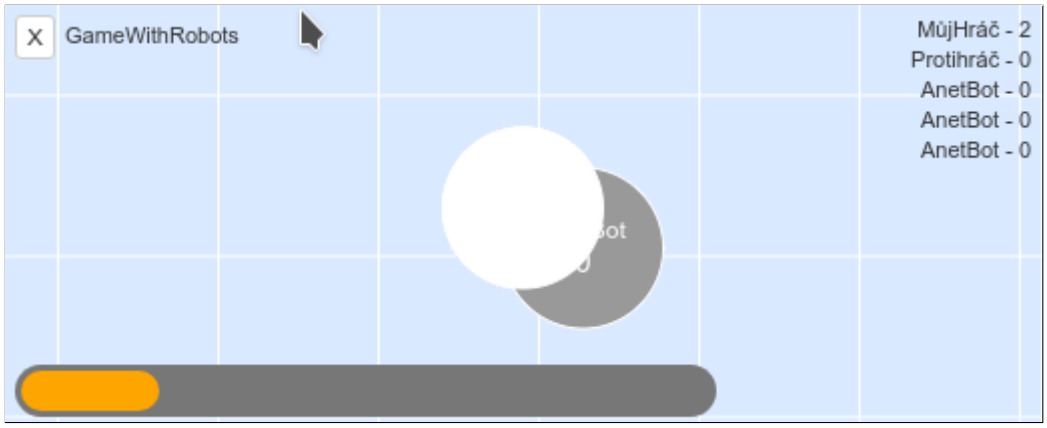


Obrázek 5.3: Ukázka exploze hráče z demonstrační aplikace

Na obrázku 5.3 je zobrazena ukázka exploze hráče. Tato exploze je plynulá a poloměr kuličky hráče se postupně roztahuje a pohlcuje všechny hráče v dosahu. Explozi lze vyvolat stisknutím klávesy mezerníku v případě, že má hráč dostatečné množství energie.

5.3.2 Skrývání vlastního hráče

Na obrázku 5.4 je zobrazeno skrytí hráče. Jedná se o snímek obrazovky hráče, který akci provedl. Hráč je pro jasné odlišení tohoto stavu zobrazen bílou barvou. Pro ostatní hráče ve hře není zobrazen vůbec. Pomocí skrývání a explozí se můžeme nepozorovnaně přiblížit k jinému hráči a zničit ho explozí.



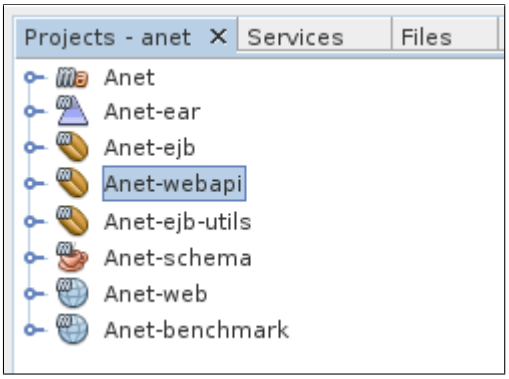
Obrázek 5.4: Ukázka skyrtí hráče z demonstrační aplikace

5.3.3 Zvýšení rychlosti

Zvýšení rychlosti probíhá stisknutím klávesy F. Tato akce způsobí, že přepočítávání pozice na straně serveru bude rychlejší oproti standardní hodnotě a hráč se tak bude po herní ploše pohybovat vyšší rycholstí.

5.4 Struktura zdrojových souborů

V této kapitole bude popsáno, ze kterých částí se aplikace skládá. V následujících podkapitolách se budeme věnovat zdrojovým souborům aplikace. Znalost, ve kterých částech se konkrétní soubory aplikace nacházejí, je pro pochopení funkčnosti celé aplikace vhodná.



Obrázek 5.5: Části aplikace zobrazené v prostředí NetBeans

Aplikace je rozdělená na několik částí a sestavuje se pomocí buildovacího nástroje Maven, kterým jsou řešeny jak závislosti jednotlivých částí mezi sebou, tak závislosti na externích knihovnách, které aplikace obsahuje. Na obrázku 5.5 jsou zobrazeny jednotlivé části, ze kterých se aplikace skládá. Rozdělení aplikace vychází z vlastních zkušeností s platformou Java EE a má své důvody, které budou popsány v následujících podkapitolách.

5.4.1 Anet

První část se jmenuje `Anet`, což je pracovní název celé aplikace. Jedná se o konfigurační soubor `pom.xml` nástroje Maven, který se stará o sestavování celé aplikace.

5.4.2 Anet-ear

Další část aplikace `Anet-ear`, reprezentuje pouze speciální typ archivu technologie Java EE, do kterého jsou sestavené moduly aplikace (`.jar`, `.war`, `.ejb`), který zavádíme na aplikační server.

5.4.3 Anet-ejb-utils

Tento modul existuje protože nechci, aby byla přímá závislost modulu `Anet-war` na modulu jádra aplikace `Anet-ejb`. Části aplikace určené pro více modulů, jako je například třída zajišťující konfiguraci aplikace, jsou umístěny zde.

5.4.4 Anet-schema

Modul `Anet-schema` obsahuje třídy datových strukur a výjimek, které jsou použity ve všech částech aplikace. Tento modul je uveden v závislostech ostatních modulů demonstrační apliakce a ostatní moduly s těmito třídami a výjimkami pracují.

- `PlayerData.java` - jedná se o datovou strukturu, která reprezentuje hráče na straně serveru. O hráči si udržujeme několik zásadních informací, jako například to, v jaké hře hráč aktuálně je, pozici hráče na herní ploše, směr pohybu a rychlost. Dále obsahuje výčtový typ `PlayerStatus`, který nám o hráči říká, ve kterém stavu je. Tento typ může nabývat například hodnot `NEW`, pro hráče kteří byli právě vytvořeni. Po úspěšném připojení se tento status mění na stav `JOINED`. V případě některé z herních akcí se status hráče může měnit na některý z `EXPLODED`, `FASTER`, `HIDDEN`. V případě, že je hráč zničen explozí, bude jeho status nastaven na hodnotu `DESTROYED`. V případě opuštění hry bude status nastaven na `LEFT`. Stavy je možné nastavit i dočastně. Mimo svůj číselný kód obsahují ještě hodnotu, která reprezentuje dobu trvání tohoto stavu, po vypršení této doby je nastaven status odpovídající hodnotě v proměnné `PlayerStatus nextPlayerStatus`. Například stav `HIDDEN( 14, 4500 )` říká, že číselný kód stavu je číslo 14 a doba trvání tohoto stavu je 4500 miliseknud. Po tomto časovém okamžiku je stav přepnut do stavu, který je nastaven v proměnné `nextPlayerStatus`. Tento mechanismus nám zajistí jednoduchou konfigurovatelnost doby jednotlivých stavů.

- `GameData.java` - tato datová struktura reprezentuje skupinu hráčů na serveru. Jednou z vlastností, které třída obsahuje, je proměnná `Set<PlayerData> playerDataSet`, která udržuje informaci o tom, kteří hráči jsou ke hře připojeni. Mezi další důležité proměnné této třídy patří například `int width` a `int height`, které udržují informaci o tom, jaká je velikost herní oblasti hry.

- `GameDataInfo.java` - tato datová struktura slouží pro přenos dat protokolem WebSocket do klientské části aplikace. Obsahuje pouze informace, které jsou důležité pro správnou funkci aplikace na klientské stanici. Je to tedy zjednodušená verze struktury `GameData`, která obsahuje méně informací a tím šetří množství informací přenesených mezi servererovou a klientskou částí aplikace.

- `PlayerDataInfo.java` - stejně jako `GameDataInfo`, se v případě této struktury jedná o zjednoduššenou verzi datové struktury `PlayerData`. `PlayerData` obsahuje některé privátní informace, které by se nikdy neměly dozvědět ostatní hráči. V této zjednodušené struktuře tedy nejsou obsaženy. Struktura obsahuje informace důležité pro vykreslení jako je například jméno, počet bodů, pozice hráče na herní ploše nebo to, zda se jedná o robota nebo jinou klientskou aplikaci.

- `PlayerScoreInfo.java` - tato datová struktura je využita k aktualizaci žebříčku hráčů. V případě, že se některému z hráčů změní počet bodů, nebo v případě, že se do hry připojí nový hráč, je žebříček hráčů aktualizován. K aktualizaci žebříčku ale vůbec nejsou potřeba informace o tom, v jaké pozici na herní ploše se hráč nachází. Jedná se tedy o více zjednodušenou datovou strukturu než je `PlayerDataInfo`. Je pravdou, že by informace mohly být přenášeny v původní datové struktuře, která nebude mít nastavené některé položky. Při serializaci do formátu JSON by se jednalo o stejně velké množství dat. Nicméně jiný název pro jiný typ použití v aplikaci zajistí lepší čitelnost zdrojového kódu.

- `Message.java` - základní datová struktura, která je dále děděna. Obsahuje pouze číslený identifikátor `int msgType`, který označuje typ zprávy. Třídy, které tuto základní třídu rozšiřují - `CliMessage.java` a `SvrMessage.java`, obsahují konstanty s číselnými kódy zpráv. Hodnoty těchto konstant musejí odpovídat hodnotám, které jsou posílány po WebSocketu z klientské nebo do klientské části aplikace.

- `CliMessage.java` - tato třída rozšiřuje základní třídu `Message.java`. Jedná se o typ zpráv, které jsou posílány z klientské aplikace na server. Jelikož je zpráva posílána z klientské aplikace na server, je nutné serveru říct, o kterého hráče se jedná. Klient tedy svoji zprávu pošle s vlastním tajným identifikátorem v instanční proměnné `String srcPlayer`. Dále je nutné specifikovat ke které hře zpráva patří, abychom byli schopni ověřit, zda má hráč opravnění této hře vůbec zasílat zprávy. Infomrace o tom, které hře zpráva náleží, je obsažena v proměnné `String srcGame`, která obsahuje textový identifikátor konkrétní hry.

- `SvrMessage.java` - třída rovněž rozšiřuje třídu `Message.java` a obsahuje proměnné pro identifikaci hráče `String dstPlayer` a hry `String dstGame`. Informace o hráči je zadána pouze v případě, že zpráva je určena pro konkrétního hráče. V případě, že je zpráva určena pro celou synchronizovanou skupinu hráčů, není tato položka vyplněna a mechanismus rozesílající zprávy klientů je rozešle všem hráčům ve hře identifikované pomocí hodnoty `String dstGame`.

5.4.5 Anet-webapi

Tento modul je pouze prostředník mezi, jádrem aplikace `Anet-ejb`, a klientskou částí aplikace `Anet-war`. Při pohledu do zdrojových souborů se může zdát, že je tento modul zbytečný a bylo by možné většinu souborů přenést do modulu jádra aplikace `Anet-ejb`. Modul `Anet-webapi` slouží ke komunikaci mezi jádrem aplikace a webovým klientem aplikace. V případě, že by bylo potřeba doimplementovat specifické změny, nebo volání pro webového klienta aplikace, je možné tyto změny udělat právě v této části. V případě, že bychom chtěli implementovat komunikační rozhraní pro mobilní aplikaci, nebylo by potřeba zashovat do stávajících částí určených pro webového klienta `Anet-war` a `Anet-webapi`, ale vytvořili bychom si novou část aplikace, například `Anet-mobileapi`,

do které bychom vložili specifické části pro mobilní aplikaci. Pokud bude mobilní i webový klient pracovat přes tyto prostředníky, kteří budou zpracovávat a ošetřovat specifické stavy, či poskytovat specifická volání pro konkrétní typ klientské aplikace, získáme jednoduší a přehlednější jádro aplikace, které se bude starat pouze o to, co má, a nebude obsahovat další části kódu, které jsou specifické pro daného klienta.

5.4.6 Anet-war

Jedná se o modul s webovou klientskou částí aplikace. Tento archiv obsahuje části aplikace, které jsou napsané v javascriptu, který běží na klientské stanici. Tento modul je závislý na modulu `Anet-webapi`, který je zde prostředníkem mezi webovou částí aplikace a jádrem aplikace. Tato část obsahuje několik důležitých souborů s následující funkcí:

- `index.xhtml` a jeho controller v souboru `GameController.java`. Jedná se soubor, který je zpracovaný pomocí technologie `Java Server Faces`. Tento soubor obsahuje podmíněné vykreslování, které dokáže vykreslit dvě různé stránky. Informace o aktuálním hráči se drží v sezení hráče na serveru. Pokud hráč není připojen k žádné konkrétní hře, zobrazí se formulář, kde si hráč může zadat přezdívku a vybrat z nabídky her. Pokud je hráč připojen v nějaké konkrétní hře, vykreslí se druhá větev, která obsahuje JavaScriptový kód. Tento JavaScriptový kód se po načtení spustí a samotná klientská aplikace je od té doby funkční.

- `Anet.js` - samotná klientská aplikace, která je mimo část se vstupním formulářem, který je napsaný v Java Server Faces, napsaná v JavaScriptu, je rozdělena na dva soubory. `Anet.js` je ten obecnější z nich. Po načtení zdrojových souborů do prohlížeče je zde navázáno spojení pomocí WebSocket API. Dále je zde implementováno přijímání a odesílání zpráv po WebSocketu a kalibrace času, které řeší možný problém s posunem času mezi serverem a klientskými aplikacemi. Snaha byla mít v tomto obecnějším souboru co nejméně implementačních detailů, které jsou určené pouze na zvolenou demonstrační aplikaci.

- `AnetViewComponent.js` - naopak tato část klientské aplikace obsahuje konkrétní implementační detaily samotné demonstrační aplikace. Ve funkci `buildHtml` se vykreslí celé okno aplikace. Změny grafického výstupu aplikace se provádějí v metodě `update`, která je volána v závislosti zprávách přicházejících ze serveru.

- `cache.xhtml` a jeho controller v souboru `CacheController.java`. Tato část pouze zobrazuje obsah v paměti. Po vykreslení této stránky v okně prohlížeče bude zobrazen seznam her v paměti a seznam hráčů v konkrétních hrách. Tato stránka aplikace slouží převážně pro kontrolu struktury dat v paměti při vývoji aplikace.

- `PlayerSessionBean.java` - tato třída má anotaci `@SessionScoped`. Tato anotace v technologii Java EE zajistí, že informace udržované v obejktu budou udržované po celou dobu sezení. V proměnné `String playerHash` držíme tajný identifikátor hráče, pomocí kterého jsme schopni získat datovou strukturu `PlayerData` z jádra aplikace. Pokaždé, když v průběhu požadavku potřebujeme strukturu `PlayerData`, tak ji pomocí volání `Anet-webapi` získáme z jádra aplikace, a můžeme s ní libovolně nakládat. V modulu `Anet-war` není ideální držet strukuturu `PlayerData`, protože jak již bylo řečeno `PlayerSessionBean.java` je `@SessionScoped` a při každé serializaci a deserializaci tohoto objektu, mezi požadavky na server, je vytvářen nový obejkt,

který není možné porovnávat na rovnost objektů při operacích v jádru aplikace. Tento, na první pohled složitější postup, nám umožní jednodušší zpracování v jádru samotné aplikace.

- `PlayerEndpoint.java` - jedná se o koncový bod, na který se WebSocket připojuje na straně serveru a zpracovává příchozí zprávy po WebSocketu. Pomocí anotace `@ServerEndpoint` je definována url samotného websocketu, encoder a decoder zpráv, které budou po WebSocketu putovat. Soubor obsahuje standardní zpracování zpráv dle WebSocket API 3.6.2, které jsou definovány pomocí anotací `@OnOpen` `@OnError` `@OnClose` a `@OnMessage`. V případě otevření WebSocketu je zavolána metoda s anotací `@OnOpen`, tato metoda obsahuje objekt Session, pomocí kterého jsme schopni připojenému klientovi poslat zprávu zpět. Tento objekt je tedy nutné si uchovat. Objekt je uchováván pomocí třídy `SocketSessionManagerBean.java`, popsané dále. V metodě `onError` dochází pouze k logování důvodů, proč k problému došlo. Metoda `onClose` je vyvolána pokud dojde k uzavření spojení. V tom případě například uživatel uzavřel prohlížeč a není nutné, aby dále existoval ve hře. Proto je jádro pomocí volání v `Anet-webapi` upozorněno na to, že hráč odešel. Veškeré zprávy, které přijdou z klietnské stanice jsou zpracovány v metodě `onMessage`. Každá zpráva z klientské stanice obsahuje číselný kód zprávy, díky kterému můžeme zprávu zpracovat dle konkrétních požadavků.

- `SocketSessionManagerBean.java` - v případě připojení nebo odpojení od WebSocketu v souboru `PlayerEndpoint.java` je v proměnné této třídy udržován seznam, který mapuje tajný identifikátor hráče, na objekty Session z metod koncového bodu WebSocketu. Abychom zajistili, že mapa bude v aplikaci pouze jedenkrát, je zaanotována pomocí anotace `@Singleton`, technologie Java EE se postará o to, aby byl objekt jedináček.

- `OutgoingMessageBean.java` - tato třída slouží k rozesílání zpráv ze strany serveru ke konkrétním klientům. Pomocí proměnné, která udžuje mapování tajných identifikátorů hráčů na objekty Session v `SocketSessionManagerBean.java`, je možné rozesílat zprávy konkrétním klientům. Tuto metodu je pomocí anotace `@Observer` a vlastní anotace `@OutgoingMessage` možné volat i z jádra aplikace.

- `JsonMessageDecoder.java` - slouží k dekodování zpráv. Je zde využita knihovna `Gson` od společnosti Google, která nám zajišťuje kontrolu zpráv a dekóduje pouze ty typy zpráv, které v aplikaci podporujeme.

- `JsonMessageEncoder.java` - slouží k zakódování zpráv ze serveru ke klientovi. Zde pomocí knihovny `Gson` dochází k serializaci objektu do formátu `JSON`, který je následně odesílán po WebSocketu ke klientovi.

5.4.7 Anet-ejb

Jádro aplikace je umístěno v modulu `Anet-ejb`. Jedná se o modul, který pracuje v EJB kontejneru 2.2.3. Pomocí tohoto modulu jsou drženy veškeré informace. Tato konkrétní demonstrační aplikace pracuje pouze v paměti serveru a nevyužívá ke svému běhu žádnou databázi. Pro bližší pochopení příkladů v následujících kapitolách zde nastíníme, které soubory tento modul obsahuje:

- `GameInitBean.java` - tento Bean je opatřen anotací `@Startup`, která zajišťuje vytvoření instance hned po startu běhu aplikace na aplikačním serveru. Jedná se o jedináčka, což zajišuje anotace `@Singleton`. Díky těmto dvoum anotacím zajistíme, že bude objekt vytvořen pouze jednou po zavedení aplikace na aplikační server. V tomto místě tedy zavoláme metodu, která přidá nějaké objekty datových struktur her do jádra aplikace. K těmto datovým strukturám se budou moci následně hráči připojit.

- `GameDataBean.java` a rozhraní pomocí kterého je s tímto beanem komunikováno `GameDataBeanLocal.java`. Jedná se o Bean typu jedináček, což zajišťuje anotace `@Singleton`. Tento bean uchovává objekty všech her a hráčů v paměti. Bean obsahuje `HashMap<String, GameData> gameHashMap`, která mapuje identifikátor hry, na konkrétní objekt GameData a `ConcurrentHashMap<String, PlayerData> playerHashMap`, která mapuje tajný identifikátor hráče na objekt PlayerData v paměti. Ještě je zde proměnná `ConcurrentHashMap<PlayerData, Long> leavingGameTsMap`, která je si udržuje seznam hráčů, kteří chtějí opustit skupinu. Důvod tohoto seznamu je popsán v kapitole 5.5.5, která pojednává o problému obnovení okna prohlížeče při hře.

- `GameBean.java` a jeho rozhraní `GameBeanLocal.java`. Jedná se o třídu, která provádí veškeré operace nad daty z `GameDataBean.java`. Modul klientské aplikace volá metody modulu jádra aplikace pomocí prostředníka `Anet-webapi`. Jádro v této třídě zpracuje požadavky, změní stav aplikace v paměti a vrací klientům čerstvá data po provedených změnách. Tento Bean je opatřena anotací `@Stateless`, to znamená, že si metody nedrží žádný stav, můžeme si je představit jako černou skříňku. Pokud by se neměnila data, která metody upravují v `GameDataBean.java`, vracely by metody pro stejný vstup vždy stejný výstup. Metody mohou být volány paralelně od různých klinetů. V některých případech je tedy nutné zajistit zamčení objektu `GameData`, dokud není práce s tímto objektem dokončena.

- `GameSchedulerBean.java` - V kapitole 5.5.5 bude blíže vysvětleno, proč potřebujeme periodicky volat určité operace v jádru aplikace. Tento Bean obsahuje plánovač, který periodicky volá metody, které nějakým způsobem ovlivňují stav konkrétní hry.

- `GameManagerBean.java` a rozhraní `GameManagerBeanLocal.java`. Jedná se o část, která se stará o klienty, kterým se rozpojilo WebSocket spojení. Klient není ze hry odstraněn okamžitě, protože nevíme, zda klient opravdu zavřel okno prohlížeče a odešel, nebo pouze provedl obnovení stránky. Při rozpojení WebSocketu mezi klientem a serverm si tuto informaci poznačíme a v případě, že jej klient během několika málo vteřin spojení opět nenaváže, tak jej skutečne odstraníme ze hry.

5.4.8 Anet-benchmark

Tento modul je samostatný war archiv, který je na výše uvedeném nezávislý. Tato malá webová aplikace slouží k připojování velkého množství hráčů k demonstrační aplikaci. Připojením velkého množství hráčů, kteří jsou schopni přijímat a odesílat zprávy po připojeném WebSocketu se pokoušíme zjistit, jak velké množství klientů zvládne serverová část aplikace obsloužit.

5.5 Implementační detaily z demonstrační aplikace

Pro pochopení některých detailů z demonstrační aplikace bude v této kapitole popsáno několik případů, kterými se pokusím vysvětlit funkčnost celé aplikace. Vysvětlování bude obsahovat i fragmenty zdrojových kódů, avšak spousta částí zdrojových kódů bude nahrazena pouze krátkým komentářem, říkajícím, co se v dané části skutečně vyskytuje, nebo uplně vynechána. Kompletní zdojové kódy jsou k nalezení na přiloženém médiu.

5.5.1 Připojení hráče ke hře

První věc, kterou vidíme při spuštění aplikace, je formulář na stránce `index.xhtml`, který slouží pro zadání přezdívky hráče a pro výběr hry, ke které se chceme připojit. Podmínka vykreslení říká, že struktura `playerData` pro našeho hráče ještě nebyla vytvořena, nebo není připojen k žádné hře. Odeslání formuláře zajistí vytvoření struktury `playerData` a naplnění položky `gameData`, která říká, ke které hře jsme připojeni.

Ukázka kódu 5.1: Podmínky pro zobrazení úvodního formuláře

```
<ui:fragment rendered="#{empty gameController.playerData
            or empty gameController.playerData.gameData}">
    <h1>ANET</h1>
    <!-- formulář -->
</ui:fragment>
```

Po odeslání a zpracování požadavku již podmínka pro vykreslení formuláře nebude splněna a vykreslí se tak druhá část v souboru `index.xhtml`. Pokud je hráč připojen ke hře, je možné vykreslit část s javascriptem, která se po spuštění vykreslí na celou obrazovku herního monitoru a bude reprezentovat samotnou hru. V následujícím úryvku kódu si můžete všimnout části `p:poll`, jedná se o komponentu z knihovny PrimeFaces, která pouze zajišťuje periodické volání nějaké funkce na serveru. Tato část slouží pouze k prodlužování životnosti našeho sezení na serveru, protože po celou dobu hraní budeme se serverem komunikovat pouze po WebSocketu, který nám životnost sezení nedokáže prodloužit. V případě, že bychom se po hře delší, než nastavení délky sezení, vrátili zpět na úvodní stránku, vykreslila by se nám zpráva o vypršení sezení a ztratili bychom náš tajný indetifikátor hráče, který se drží v našem sezení na straně serveru.

Ukázka kódu 5.2: Podmínky pro načtení klientské aplikace

```
<ui:fragment rendered="#{not empty gameController.playerData
                        and not empty gameController.playerData.gameData}">
    <p:poll autoStart="true" interval="900"
            update="@this" process="@this"
                listener="#{keepSessionController.keepSession()}"/>
    <script>
        var anet = null;
        $(document).ready(function () {
            anet = new Anet(
                "#{gameController.playerData.playerHash}",
                "#{gameController.playerData.publicId}",
                 #{gameController.gameDataInfoJson},
                 $("#gameview")
                );
        });
    </script>
    <main id="gameview"></main>
</ui:fragment>
```

V úryvku kódu je vidět inicializace javascriptového objektu, kterému předáváme informace o našem hráči a o hře samotné. Také mu předáváme `jQuery` objekt, do kterého má být celá hra vykreslena. V kaskádových stylech je zajištěno, aby tento objekt byl vykreslený na celou obrazovku.

5.5.2 Navázání WebSocket spojení

V konstruktoru javascriptového objektu `Anet.js` se mimo jiných náležitostí nachází i část kódu, která zajišťje navázání spojení pomocí WebSocket API. Jedná se o použití konstruktoru `WebSocket()`, kterému předáme URL, na které se na serveru nachází koncový bod pro WebSocket spojení, ke kterému se chceme připojit. Můžeme zde pozorovat, že se připojujeme ke koncovému bodu, který obsahuje tajný indetifikátor hráče. Tento identifikátor pomůže serveru při přijímání a odesílání zpráv zjistit, o kterého hráče na straně serveru se jedná.

Pokud se nám spojení opravdu podaří navázat, je klientská aplikace napsaná v JavaScriptu spuštěna voláním metody `that.init();`.

Ukázka kódu 5.3: Navázání spojení na straně klienta v souboru Anet.js

```
Anet = function (playerHash, playerId, gameDataInfo, htmlId) {

    // inicializace proměnných

    var that = this;
    var s = (window.location.protocol === "https:" ? "s" : "");
    this.ws = new WebSocket("ws" + s + "://" + window.location.host +
        "/socket/player/" + this.playerHash);

    this.ws.onopen = function (evt) {
        console.log("WebSocket opened");
        that.init();
    };

    // další funkce objektu this.ws
};
```

Na straně serveru se o navázání WebSocket spojení stará třída `PlayerEndpoint.java`. Anotací `@ServerEndpoint` je určeno, které třídy se budou starat o převod tříd z a do formátu `JSON`. Při navazování spojení se volá metoda s anotací `@OnOpen`, ve které se kromě uložení mapování mezi tajným identifikáorem hráče `playerHash` a objektu `Session` volá funkce `gameWebApiBean.playerConnected( playerHash );`. Jak již bylo nastíněno a bude vysvětleno v části 5.5.5, jedná se o řešení problému s rozpojením WebSocket spojení. V metodě s anotací `@OnClose` je hráč přidáván do speciálního seznamu v jádru aplikace, ve kterém si jádro aplikace drží informaci o klientech, kterým se v nedávné době rozpadlo WebSocket spojení. V metodě `@OnOpen` je potřeba hráče z tohoto seznamu smazat, pokud je v seznamu obsažen. Jádro aplikace tento seznam zpracovává a zpětne hráče odstraňuje z datových struktur.

Ukázka kódu 5.4: Navázání spojení na straně serveru v souboru PlayerEndpoint.java

```
@ServerEndpoint(
    value = "/socket/player/{playerHash}",
    encoders = { JsonMessageEncoder.class },
    decoders = { JsonMessageDecoder.class }
)
public class PlayerEndpoint {
```

```
7
8       String playerHash;
9
10      //inicializace proměnných a další jiné metody v této třídě.
11
12      @OnOpen
13      public void onOpen(
14          Session session,
15          EndpointConfig conf,
16          @PathParam( "playerHash" ) String playerHash
17      ) {
18          this.playerHash = playerHash;
19
20          //mapovaní objektu Session na tajný identifikátor hráče
21          socketSessionManagerBean.addSession( playerHash, session );
22
23          //řešení obnovení okna prohlížeče
24          gameWebApiBean.playerConnected( playerHash );
25      }
26
27      // další metody v této třídě
28  }
```

5.5.3 Odesílání zpráv z klienta na server

Pokud je hráč připojen ke hře, je načtena jeho klientská aplikace, která vypadá podobně jako na obrázku 5.3. Nyní můžeme z klientské aplikace odesílat zprávy po WebSocketu, pomocí kterých budeme měnit stav aplikace na serveru. Tyto změny jsou dále ze serveru odesílány ostatním hráčům, kterých se týkají.

Pomocí několika úryvků zdrojových kódů se pokusím nastínit, jak probíhá odesílání zpráv z klientské stanice na server. O vykreslení herní plochy a hráčů na herní ploše se stará Javascriptová třída `AnetViewComponent`. V této třídě implementovaná metoda `onKeyDown`, která zpracovává klávesové příkazy od hráče. Následující ukázka obsahuje zpracování události „exploze“, pomocí které se pokoušíme pohltit ostatní hráče v blízkosti.

Ukázka kódu 5.5: Metoda onKeyDown v souboru AnetViewComponent.js

```
1  onKeyDown: function (event) {
2
3    //kontrola, zda je možné tuto zprávu odeslat
4
5    if (event.keyCode === Keyboard.SPACE) {
6        var explodeMsg = new CliMsg(CliMsg.MSG_GO_EXPLODE);
7        this.anet.sendMsg(explodeMsg);
8      }
9  }
```

Po stisknutí klávesy enter, se vytvořil objekt zprávy `new CliMsg(CliMsg.MSG_GO_EXPLODE);`, který je pomocí metody `sendMsg` ve třídě `Anet` odesílán na server. Tento konkrétní objekt neobsahuje žádné doplňující informace. Na serveru je nutné vědět pouze to, o jaký typ zprávy se jedná. Každý typ zprávy odesílaný na server má speciální číselný kód. Kód zprávy je obsažen v konstaně `CliMsg.MSG_GO_EXPLODE`, která je umístěna v souboru `Anet.js`. Na straně serveru jsou v souboru `CliMessage.java` rovněž umístěny konstanty s číselnými kódy zpráv, které mohou přijít od klienta. Pro korektní zpracování zpráv je nutné, aby tyto číselné kódy na straně klienta i serveru byly stejné.

Ukázka kódu 5.6: Metoda sendMsg v souboru Anet.js

```
sendMsg: function (msg) {

    //nastavení základních informací o zprávě
    msg.srcPlayer = this.playerHash;
    msg.srcGame = this.gameDataInfo.id;

    //logování zpráv do konsole
    if (this.dumpMessages) {
        console.log("-> " + msg.msgType + " " + msg.toString());
        console.log(msg);
    }

    //odeslání zprávy po navázaném WebSocketu
    this.ws.send(JSON.stringify(msg));
},
```

Metoda `sendMsg` ve třídě `Anet` se používá pro odesílání všech zpráv z aplikace. Každé zprávě se před odesláním nastaví tajný identifikátor hráče, od kterého pochází a identifikátor hry, ze které hráč zprávu odesílá. Dále tato metoda umožňuje logování odesílaných zpráv do javascriptové konzole. Více o logování zde 5.5.8. Na konci této metody dochází k odeslání na server. Zprávu převedeme do formátu `JSON` a odešleme pomocí metody `send()` WebSocket objektu na server.

Přijetí zprávy na serveru probíhá v metodě třídy `PlayerEndpoint`, která vlastní anotaci `@OnMessage`. Všechny zprávy, které přijdou od klientů, jsou zpracovávány v této metodě. Metoda nejdříve ověřuje zda hráč, který zprávu poslal a umístil do ní svůj tajný identifikátor existije. Pokud ne, zpracování zprávy se ukončí. V normálním případě však hráč existuje.

Pokud hráč existuje, vezme se číselný kód přijaté zprávy, který musí odpovídat některému z kódů ve třídě `CliMessage.java`. Pokud existuje implementace pro tento typ zprávy a její kód je obsažen ve struktuře `switch`, je tato zpráva odeslána ke zpracování.

Ukázka kódu 5.7: Přijetí zprávy v metodě onMessage v souboru PlayerEndpoint.java

```
@OnMessage
public void onMessage( Session session, CliMessage message ) {
    try {
        // kontrola, zda má hráč právo zprávu odesílat
        checkPlayerHash( message );
    } catch ( TPAccessDeniedException ex ) {
        Logger.getLogger( PlayerEndpoint.class.getName() ).log( Level.SEVERE,
            null, ex );
        return;
    }

    switch ( message.getMsgType() ) {

        // různé typy zpráv, které zpracováváme

        case CliMessage.MSG_GO_EXPLODE:
            processMessage( ( GoExplodeMessage ) message );
            return;
  }
}
```

Pro každý typ zprávy, který je na serveru podporován, existuje metoda, která ji zpracuje a posílá dále do jádra aplikace.

Ukázka kódu 5.8: Zpracování konkrétní zprávy v souboru PlayerEndpoint.java

```
public void processMessage( GoExplodeMessage message ) {
    try {
        gameWebApiBean.goExplode(
            message.getSrcPlayer(),
            message.getSrcGame()
        );
    } catch ( TPRecordNotFoundException | InterruptedException |
        TPInvalidStateException ex ) {
        Logger.getLogger( PlayerEndpoint.class.getName() ).log( Level.SEVERE,
            null, ex );
    }
}
```

Samotné zpracování zprávy probíhá v jádru aplikace a liší se dle typu zprávy. V případě, že zpracování zprávy vyvolá nějaké změny stavu aplikace, tak jádro aplikace rozesílá zprávy připojeným klientům. Klienti tyto zprávy zpracují a tímto zpracováním zpráv je docíleno, že grafický výstup klientů bude synchronizován.

5.5.4 Odesílání zpráv z serveru na klienta

V této podkapitole se pokusím vysvětlit, jak funguje odesílání zpráv ze serveru ke klientům. V sekci 5.5.2 se pojednává o tom, že je v aplikaci udržována proměnná, které udržuje mapovaní tajných indentifikátorů hráčů na seznam otevřených WebSocket spojení. Abychom byli schopni odeslat zprávu konkrétnímu klinetovi (hráči), potřebujeme se dostat k objektu `Session` z tohoto seznamu, pomocí kterého můžeme odeslat zprávu.

Problém je v tom, že se všechny změny v aplikaci odhrávají modulu jádra aplikace, který běží v EJB kontejneru. Na obrázku 2.3 je vizualizováno, že technologie WebSocket je podorována pouze ve webovém kontejneru. Mapování hráčů na objekty Session, příjímání zpráv a odesílání zpráv po WebSocketu je tedy umístěno ve webovém kontejneru. Pomocí technologie Java EE, která se jmenuje Observer, je zajištěna komunkace z jádra aplikace směrem k webovému kontejneru a díky této komunikaci lze z jádra aplikace odesílat zprávy pomocí webového kontejneru.

V následujícím úryvku programu je vidět metoda `joinGame`, která zajišťuje připojování hráčů ke hře. Většina chybových stavů, jako to, že tento hráč neexistuje, nebo hra ke které se chce připojit neexistuje, se ošetřuje před voláním této metody. Tato metoda počítá s tím, že jsou všechna data v pořádku. Jelikož budeme zapisovat do objektu `gameData`, tak si jej zamčeme, provedeme operace a opět odemčeme. V rámci připojování hráče ke hře se z tohoto Beanu, který pouze vykonává kód a nemá žádnou pamět volá, bean s pamětí, který díky anotaci `@Singleton` existuje v jednom exempláři. Beanu, který uchovává všechna data se řekne, aby změnil obsah datových struktur, do seznamu hráčů ve hře přidá nového hráče a hráči přidá referenci na hru, do které se právě připojil. Pokud vše proběhne hladce a nebude vyvolána výjíma, tak se pošlou dvě zprávy. Jedna zpráva se posílá pouze hráči který se připojil. A je to signalizace o tom, že připojení bylo úspěšné. Druhá zpráva, u které lze v konstruktoru pozorovat pouze proměnnou `gameData`, se posílá všem hráčům ve hře. Jedná se o aktualizovaný seznam hráčů. Tento seznam můžeme vidět na obrázku 5.4 v pravém horním rohu.

Ukázka kódu 5.9: Odesílání zpráv z jádra aplikace po WebSocketu

```
@Stateless
public class GameBean implements GameBeanLocal {

    //inicializace proměnných a závislostí
```

```
5
6       @Inject
7       @OutgoingMessage
8       Event<SvrMessage> outgoingMessageEvent;
9
10      @Override
11      public void joinGame( PlayerData playerData, GameData gameData )
12          throws InterruptedException, TPInvalidStateException {
13
14          KeyedMutex<GameData> mutex = gameDataBean.lockGameData( gameData );
15          try {
16              gameDataBean.joinGame( playerData, gameData );
17
18              JoinedGameMessage joinedGameMessage = new JoinedGameMessage( gameData,
                    playerData );
19              outgoingMessageEvent.fire( joinedGameMessage );
20
21              PlayerListUpdateMessage playerListUpdateMessage = new
                    PlayerListUpdateMessage( gameData );
22              playerListUpdateMessage.setPlayerScoreList( getPlayerScoreInfoList(
                    gameData ) );
23              outgoingMessageEvent.fire( playerListUpdateMessage );
24
25          } finally {
26              mutex.unlock( gameData );
27          }
28      }
29
30      //další metody
31  }
```

V případě vyvolání události odeslání zprávy z jádra metodou `fire`, v předchozí ukázce kódu, se vykoná metoda `outgoingMessage` ve webovém kontejneru. Konkrétně ve třídě `OutgoingMessageBean.java`. V této metodě se kontroluje, zda má zpráva nastavený tajný identifikátor hráče, pokud ano, tak se automaticky posílá pouze jemu. Pokud tento identifikátor nastaven není, tak se díky identifikátoru hry získá datová struktura `GameData`, pro kterou je zpráva určena a zpráva se rozešle všem hráčům připojeným k této hře.

Ukázka kódu 5.10: Implementace odesílání zpráv jednomu hráči nebo celé skupině hráčů

```
1   @LocalBean
2   @Stateless
3   public class OutgoingMessageBean {
4
5       // inicializace, závislosti a jiné metody
6
7       public void outgoingMessage(
8       @Observes @OutgoingMessage SvrMessage outMessage
9       ) {
10        // zpráva je pro jednoho klienta
11          if ( outMessage.getDstPlayer() != null ) {
12
13              outMessage.setTs( System.currentTimeMillis() );
14              sendToPlayer( outMessage.getDstPlayer(), outMessage );
15
16          //zpráva je pro všechny klienty ve hře
17          } else {
18              GameData = gameWebApiBean.getGameDataByHash( outMessage.getDstGame() );
19
20              outMessage.setTs( System.currentTimeMillis() );
21              for ( PlayerData playerData : gameData.getPlayerDataSet() ) {
```

```
                sendToPlayer(playerData.getPlayerHash(), outMessage);
            }
        }
    }
}
```

Ve chvíli, kdy je zřejmé komu chceme zprávu poslat, stačí pouze ze seznamu, který drží mapování tajného identifikátoru na objekt Session, vytánou tyto objekty a zprávu jim rozeslat.

Ukázka kódu 5.11: Metoda sendToPlayer, sloužící k odeslání zprávy po WebSocketu

```
private void sendToPlayer( String playerHash, SvrMessage outMessage ) {

    Set<Session> sessionSet = socketSessionManagerBean.getSession(playerHash);

    if ( sessionSet == null ) {
        return;
    }
    // rozeslání zprávy
    for ( Session session : sessionSet ) {
        session.getAsyncRemote().sendObject( outMessage );
    }
}
```

5.5.5 Odpojování od hry

Jedna ze základních věcí, jako obnovení okna prohlížeče, byl v demonstrační aplikaci problém. Pokud by vás jádro aplikace odstranilo z datových struktur ihned poté, co by došla událost `onClose`, která značí uzavření WebSocket spojení, tak by hráč přišel o všechny nahrané body. Bylo by například možné implementovat chování, ve kterém by se klinetská aplikace po rozpojení WebSocket spojení pokusila automaticky znovunavázat toto spojení. Hráč by se tedy odpojil a následně připojil, ale jeho skóre by se během těchto dvou akcí vynulovalo, což určitě není to, co by hráč čekal. V případě jiného typu aplikace by mohlo dojít k jiným problémům.

Celý problém s rozpojením WebSocketu a následným uvolněním datových struktur hráče jsem vyřešil tak, že v jádře aplikace existuje proměnná `leavingGameTsMap`, do které si při rozpojení klienta uložím tajný identifikátor hráče a čas, kdy chci aby byl hráč odstraněn z datových struktur.

Ukázka kódu 5.12: Řešení odstraňování datových struktur z paměti po rozpojení WebSocket spojení

```
@Singleton
@ConcurrencyManagement( ConcurrencyManagementType.BEAN )
public class GameDataBean implements GameDataBeanLocal {

    // ostatní proměnné

    // seznam klientů, kteří se odpopojili od websocketu
    private ConcurrentHashMap<PlayerData, Long> leavingGameTsMap;

    @Override
    public void addLeavingGame( PlayerData playerData, int leavingInMillis )
        throws TPInvalidStateException {

        if ( leavingGameTsMap.containsKey( playerData ) ) {
```

```
14              throw new TPInvalidStateException( "Player " + playerData.toString() +
                    " is already marked as leaving" );
15          }
16          // do mapy přidáváme hráče, a časové razítko, kdy jej chceme odpojit
17          leavingGameTsMap.put( playerData, System.currentTimeMillis() +
                leavingInMillis );
18      }
19
20      // ostatní metody
21  }
```

Tento seznam hráčů se periodicky zpracovává pomocí metody `gameSchedulerTimer` v `GameSchedulerBean`, která je díky časovači volána každých 5 vteřin. Toto zpoždění pět vteřin lze pozorovat i v demonstrační aplikaci. Pokud proti sobě budou hrát dva hráči a pokud jeden opravdu zavře okno prohlížeče a odejde, tak jej druhý hráč na herní ploše uvidí ještě 5 vteřin. V případě, že se jednalo o překlep nebo obnovení okna prohlížeče, tak se hráč dostane do původního stavu, jako byl před rozpojením WebSocket spojení.

Ukázka kódu 5.13: Použití časovače při odstraňování datových struktur z jádra aplikace

```
1  @Singleton
2  @LocalBean
3  public class GameSchedulerBean {
4
5      // inicializace proměnných
6
7      @Schedule( hour = "*", minute = "*", second = "*/5", persistent = false )
8      public void gameSchedulerTimer() {
9
10     // metoda odstraňující hráče ze hry
11         gameManagerBean.processLeavingGame();
12   }
13 }
```

V této podkapitole bylo popsáno chování aplikace v případě rozpojení WebSocket spojení. Ještě existuje druhý způsob jak opustit hru. Pokud hráč klikne na křížek v levém horním rohu, je jádru aplikace odeslána zpráva s požadavkem na opuštění hry. A jádro aplikace zařídí, aby hráč korektním způsobem opustil hru.

5.5.6 Pohyb hráče po herní ploše

Každý klient ovládá jednoho hráče. Jeho vlastní hráč je zobrazen uprostřed prohlížecího zařízení. Pozice herní plochy a ostatních hráčů se přesouvá relativně ke středu prohlížecího zařízení, kde je zobrazen hráč, kterého ovládáme. Každý klient tedy na server odesílá pouze úhel myši, který svýrá se středem prohlížecího zařízení a vzdálenost, ve které se od středu nachází.

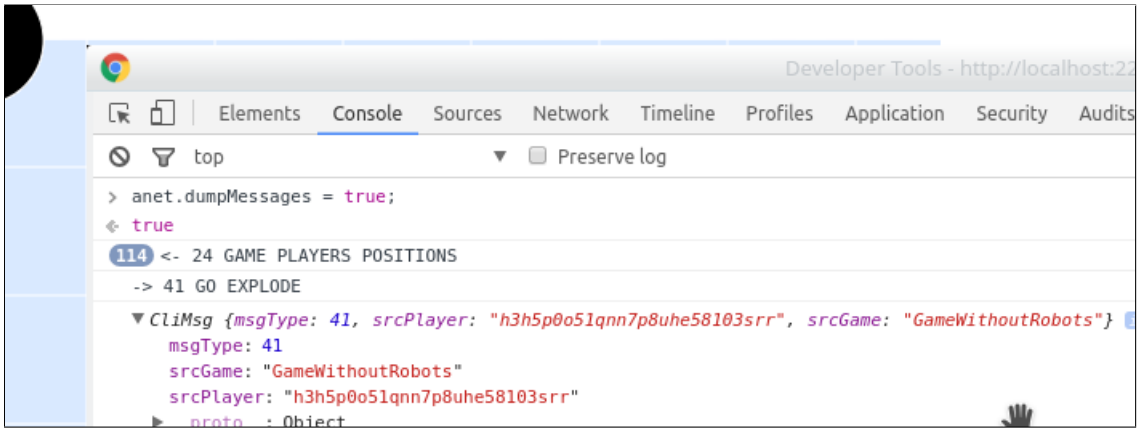
Veškerý pohyb po herní mapě se vyhodnocuje na serveru a zpět do klientské aplikace se posílají nové souřadnice všech hráčů na herní ploše. Tyto informace jsou v souboru `AnetViewComponent.js` v metodě `update` vykresleny. Vše je vykresleno pomocí HTML 5 technologie `Canvas`. Je vykreslena herní plocha a následně iterací přes seznam hráčů jsou vykresleni ti hráči, které na našem zobrazovacím zařízení můžeme vidět. Herní plocha může být rozsáhlá a zasahovat daleko mimo zobrazovací zařízení, proto vykreslujeme pouze hráče kteří jsou v naší viditelnosti.

5.5.7 Synchronizace času mezi serverem a klienty

V průběhu vývoje demonstrační aplikace byl řešen problém synchronizace času mezi připojenými klientskými stanicemi. Rozdíl času mezi klientskou statnicí a serverem je udržován ve třídě `Anet.js`. V poslední chvíli před odesláním zprávy ze serveru ke klientské stanici je nastavena proměnná `ts` aktuálním systémovým časem `outMessage.setTs( System.currentTimeMillis() );`. Při přijetí zprávy v klientské aplikaci je porovnáno časové razítko serveru a klientské stanice. Rozdíl v časech mezi serverm a klientskou stanicí je uložen a může být použit na straně klienta. V poslední verzi demonstrační aplikace není reálné využití této vlastnosti, protože samotná délka hry není časově omezená. Kdyby časově omezená byla a trvala například minutu, tak díky tomuto časovému rozdílu mezi klintskou stanicí a serverem můžeme přesně zorbazovat časový údaj ve vteřinách kdy hra končí.

5.5.8 Logování zpráv do konzole prohlížeče

Pro případy ladění aplikace je možné logovat příchozí a odchozí zprávy. Na obrázku 5.6 je zobrazeno, že po nastavení proměnné `dumpMessages` objektu `anet` na hodnotu `true`, se začnou příchozí a odchozí zprávy logovat do javascriptové konzole prohlížeče. Díky tomu je možné při vývoji efektivně sledovat, zda klientská aplikace přijímá a odesílá data tak, jak by měla. Dále objekt anet obsahuje proměnnou, kterou jsme schopni zastavit vykonávání aplikace. V některých připaděch se může hodit. Stačí v JavaScriptové konzoli prohlížeče takto nastavit proměnnou `anet.devStop = true;` a aplikace se zastaví.



Obrázek 5.6: Logování zpráv pomocí javascriptové konzole prohlížeče

Kapitola 6

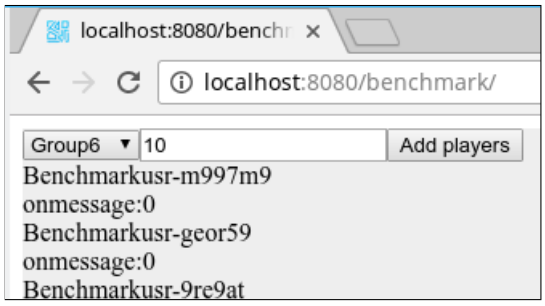
Testování výkonu aplikace

Při vývoji aplikace byl brán ohled na výkonnost a vyvíjena snaha, aby se k aplikaci mohlo připojit co největší množství klientů. Díky tomuto požadavku byla klientská část aplikace několikrát upravena tak, aby se zvýšil počet hráčů, kteří mohou být současně v jedné hře, aniž by jakkoli pozorovali zpomalení klientské aplikace.

6.1 Způsob testování

Pro možnosti otestování aplikace s velkým množstvím připojených klientů, byla napsána velice jednoduchá aplikace `Anet-benchmark`, která se dá zavést na aplikační server stejně tak, jako hlavní aplikace. V demonstrační aplikaci byly doimplementovány dvě třídy, pomocí kterých aplikace, která simuluje klienty, založí datavou strukturu pro klienta a připojí ho do skupiny. Jedná se o třídy v balíčku `net.topoul.anet.benchmark`. Třída `BenchmarkServlet.java` slouží k vygenerování tajného identifikátoru hráče a k připojení klienta do skupiny, která je specifikována pomocí GET parametru `groupId`.

Aplikace simulující zátěž serveru funguje tak, že se pomocí fromuláře zvolí počet klientů a skupina, ke které je chceme připojit. Obrázek 6.1 ukazuje, jak lze pomocí testovací aplikace připojit určitý počet klientů k určité skupině pomocí jiného prohlížeče. V jednom prohlížeči budeme ovládat vlastního hráče a v jiném přidávat klienty a sledovat, že komunikace opravdu probíhá. Celkem v aplikaci může být jeden klient kterého



Obrázek 6.1: Simulace připojení většího množství klientů

ovládáme z okna našeho prohlížeče a desítky dalších, kteří jsou pouze připojeni a nikdo je neovládá. Nicméně jakmile se připojí, tak jsou jim ze serveru odesílány zprávy, čímž se snažíme zjistit, jak velké množství klientů je možné připojit k herním skupinám aniž by zpomalení hry bylo pozorovatelné.

Server na kterém aplikace běžela byl testován na lokálním počítači. Klientské aplikace byly připojeny z jiného počítače. Jiný počítač byl zvolen z důvodů, že velké množství klientů v prohlížeči značně zatížilo prohlížeč a nebylo tedy možné určit, jakého zatížení na serveru dochází při určitém počtu klientů.

6.1.1 Konfigurace počítače se serverem

- **Procesor**: Intel Core i5-2410M (2.30GHz)
- **Operační systém**: Kubuntu 16.10
- **Paměť**: 8GB (2x 4096) 1333MHz DDR3
- **Grafika**: Integrovaná Intel GMA HD
- **Pevné disky**: 500GB Serial ATA (7200RPM)

6.1.2 Konfigurace počítače se spuštěnými klienty

- **Procesor**: Intel Core 2 Quad CPU Q8300 (2.5GHz)
- **Operační systém**: Windows 7
- **Paměť**: 4GB
- **Grafika**: Integrovaná Intel G41 Expres Chipset

6.2 Výsledky testování

Ve všech případech jsem použil větší množství prohlížečů na obou počítačích, kde klient i server běžel.

Postupně jsem se pokoušel připojovat více a více klientů ke skupinám a pozoroval, jak se mění zatížení procesoru na stroji, kde běžela serverová část aplikace. Při měření byly odečteny i hodnoty vyšší než 100%, protože vytížení 100% znamená plné vytížení jednoho jádra procesoru. Při pokusech připojit různé množství hráčů k různým synchronizovaným skupinám jsem naměřil následující výsledky:

- 3 synchronizované skupiny po 10 hráčích. Hra ve 3 prohlížečích plynulá, procesor se nepřesáhl 20%.
- 3 skupiny po 20 klientech. Hra plynulá, zatížení procesoru nepřesáhlo 30%.
- 3 skupiny po 30 klientech. Hra plynulá, zatížení procesoru nepřesáhlo 50%.
- 3 skupiny po 50 klientech. Hra stále plynulá, zatížení nepřesáhlo 90%.
- 2 skupiny po 100 hráčích, třetí skupina pouze 50. Zatížení procesoru se pohybovalo okolo hodnoty 140%. Klientské stanice nezvládají zpracovávat množství zpráv ze serveru. Hra již není plynulá.
- 4 skupiny po 25 klientech zatížení se pohybovalo okolo 50%.
- 7 skupin po 10 klientech. Zatížení procesory na 25%.

Při měření bylo možné pozorovat, že server má menší problém s více skupinami o menším počtu hráčů, než s jednou skupinou o větším množství hráčů. Také nastal problém, že vždy existoval alespoň jeden prohlížeč, který připojil desítky klientů a byl tak značně zatížen přijímáním a zpracováváním všech zpráv po WebSocketu, přesto že tyto zprávy pouze příjímal a zahazoval.

Také bylo možné pozorovat, že při větším množství připojených klientů má demonstrační apliakce problémy s vykreslováním. Vykreslování herní plochy na Canvas bylo několikrát optimalizováno. Avšak při více jak 60 připojených klientech do jedné synchronizované skupiny přestává být zpracování zpráv o pozicích hráčů plynulé a hra působí zpomaleně.

Kapitola 7

Zhodnocení a závěr

Díky hlubšímu studiu technologie Java EE a nových prvků HTML 5, jejichž detailnější znalosti byly potřeba pro vytvoření funkčního návrhu a implementaci demonstrační aplikace, jsem získal mnoho nových zkušeností, které uplatním při dalším rozvoji demonstrační aplikace nebo na jiných projektech. WebSocket protokol použitý pro okamžité předávání klientským stanicím dat ze sereru, je opravdu revoluční technologie v porovnání se zastaralým a na data náročným asynchroním načítáním částí stránek pomocí javascriptu.

Koncept demonstrační aplikace je z velké části postaven na zasílání zpráv ve formátu JSON z klientské aplikace na server a zpět. Tento koncept, řešící odesílání zpráv z jádra aplikace v EJB kontejneru pomocí Webového kontejneru do samotné klientské aplikace, je jednoduchý a velice účinný. Pro kompletní změnu funkčnosti aplikace stačí přepsat pouze části, které se na klientské straně starají o zpracování přijatých zpráv a vykreslování grafického výstupu a na straně serveru o manipulaci s daty. Smysl demonstrační aplikace může být po přepsání těchto dvou částí značně jiný.

Demonstrační aplikace byla zvolena tak, aby bylo možné názorně na příkladech ukázat, jak celý navržený systém výměny zpráv funguje. Bohužel se ukázalo, že typ demonstrační aplikace není úplně nejvhodnější. Moje představa byla, že aplikace zvládne obsluhovat stovky připojených klientů k jedné hře. Při testování jsem však zjistil, že počet zpráv odesílaných z klientských stanic na server, které signalizují změny pohybu klienta po herní ploše, a zpráv odesílaných ze serveru na klientské stanice, značících nové pozice protihráčů, je opravdu veliký. Při velkém množství připojených klientských stanic má prohlížeč problémy s okamžitým překreslením hráčů na herní ploše. Avšak tento problém je způsoben pouze nevhodně zvolenou demonstrační aplikací.

Demonstrační aplikaci bylo možné v průběhu optimalizací částečné přepsat tak, aby nebylo nutné posílat souřadnice všech hráčů na herní ploše takovou rychlostí. Bylo by možné odesílat z klietnských stanic pouze informace o tom, že se změnil jejich směr pohybu. Těchto zpráv by bylo řádově méně, než ve stávající implementaci. Všechny změny souřadnic na herní ploše by bylo nutné dopočítávat na klientských stanicích. Tento způsob optimalizace jsem však zamítl, protože na faktu, že výměna zpráv mezi klienty a serverem zajištující synchronizaci grafického výstupu klientů funguje, by nic nezměnilo.

7.1 Závěr

V rámci diplomové práce byly nastudovány technologie Java EE, HTML 5 a protokol WebSocket, pomocí kterých byl navržen a implementován způsob, jak lze synchronizovat grafické výstupy webových aplikací. Princip komunikace mezi serverovou částí a skupinami klietských stanic, ve kterých je zajištěná synchronizace grafického výstupu, je postaven na zasílání zpráv typu JSON. Klientská i serverová část aplikace je rozdělena tak, že hlavní kostra, která zajišťuje výměnu zpráv, je oddělena od části, která se stará o samotné grafické zobrazení aplikace.

Zvolená demostrační aplikace nebyla příliš vhodná, jedná se o herní aplikaci která, zobrazuje pohyb hráčů po herní ploše. V případě velkého množství připojených hráčů k jedné synchronizované skupině (padesát a více), dochází k problému na straně prohlížeče, který je značně zatížen překreslováním grafického výstupu. Tento problém je však způsoben velkým množstvím přenášených zpráv u tohoto konkrétního typu herní aplikace a na navrženém principu synchronizace grafického výstupu webových aplikací to nic nemení.

Literatura

[1] Ashmore, D.: *The J2EE Architect's Handbook: How to be a Successful Technical Architect for J2EE Applications*. DVT Press, 2004, ISBN 9780972954891.

[2] Bengtsson, P.: Are WebSockets faster than AJAX? [online]. `https://www.peterbe.com/plog/are-websockets-faster-than-ajax`, 2012-04-22 [cit. 2017-02-12].

[3] Fette, I.; Melnikov, A.: RFC 6455 - The WebSockets Protocol. 2011. URL https://tools.ietf.org/html/rfc6455

[4] Goldstein, A.; Lazaris, L.; Weyl, E.: *HTML5 a CSS3 pro webové designéry*. Encyklopedie webdesignera, Zoner Press, 2011, ISBN 9788074131660.

[5] Jendrock, E.; Evans, I.; Gollapudi, D.; aj.: *The Java EE 7 Tutorial*. číslo sv. 2 in Java Series, Pearson Education, 2014, ISBN 9780133901955.

[6] MACH, V.: Návrh a implementace interaktivní grafické HTML5 aplikace [online]. 2014 [cit. 2017-05-14]. URL http://is.muni.cz/th/374430/fi_b/

[7] Margorín, M.: *jQuery bez předchozích znalostí*. Computer Press, 2016, ISBN 9788025144954.

[8] Wang, V.; Salim, F.; Moskovits, P.: *The Definitive Guide to HTML5 WebSocket*. Books for professionals by professionals, Apress, 2013, ISBN 9781430247418.

[9] WWW stránky: Thoughts on Flash [online]. `http://www.apple.com/hotnews/thoughts-on-flash/`, 2010-04-01 [cit. 2016-12-10].

[10] WWW stránky: Komunikace v reálném čase díky Server-Sent Events a Web Sockets [online]. `https://www.interval.cz/clanky/komunikace-v-realnem-case-diky-server-sent-events-a-`
2015-03-11 [cit. 2016-12-10].

[11] WWW stránky: Flash and Chrome [online]. `https://blog.google/products/chrome/flash-and-chrome/`, 2016-09-08 [cit. 2016-12-10].

[12] WWW stránky: Inkscape [online]. `https://inkscape.org/`, [cit. 2016-01-15].

[13] WWW stránky: Apache Tomcat [online]. `http://tomcat.apache.org/`, [cit. 2016-12-15].

[14] WWW stránky: Glassfish [online]. https://glassfish.java.net/, [cit. 2016-12-15].

[15] WWW stránky: HTML5 [online]. https://www.w3.org/TR/html5/, [cit. 2016-12-15].

[16] WWW stránky: Introducing JSON [online]. http://www.json.org/, [cit. 2016-12-15].

[17] WWW stránky: Payara [online]. http://www.payara.fish/, [cit. 2016-12-15].

[18] WWW stránky: Wildfly [online]. http://wildfly.org/, [cit. 2016-12-15].

[19] WWW stránky: Can I Use [online]. http://caniuse.com/, [cit. 2016-12-20].

[20] WWW stránky: PrimeFaces [online]. http://www.primefaces.org/, [cit. 2016-12-20].

[21] WWW stránky: RichFaces [online]. http://richfaces.jboss.org/, [cit. 2016-12-20].

[22] WWW stránky: The Asynchronous WebSocket/Comet Framework [online]. https://github.com/Atmosphere/atmosphere, [cit. 2017-01-10].

[23] WWW stránky: Canvas tutorial [online]. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial, [cit. 2017-01-10].

[24] WWW stránky: HTML5 SVG [online]. http://www.w3schools.com/html/html5_svg.asp, [cit. 2017-01-10].

[25] WWW stránky: Web sockets [online]. https://html.spec.whatwg.org/multipage/comms.html, [cit. 2017-02-10].

[26] WWW stránky: Compression Extensions for WebSocket [online]. https://datatracker.ietf.org/doc/rfc7692/, [cit. 2017-02-12].

[27] WWW stránky: An XMPP Sub-protocol for WebSocket [online]. https://tools.ietf.org/html/draft-moffitt-xmpp-over-websocket-00, [cit. 2017-02-12].

Přílohy

Příloha A

Vývojové prostředí NetBeans

Demonstrační aplikace byla vyvíjena ve vývojovém prostředí NetBeans verze 8.2. Toto prostředí obsahuje sestavovací nástroj Maven a možnost jednoduchého zavádění na aplikační server. Prostředí pro svůj běh vyžaduje JDK verze 8. Pokud bude pro sestavování i zavádení aplikace na server využito prostředí NetBeans 8.2, je nutné v konfiguračním souboru `/cesta-k-adresáři/netbeans-8.2/etc/netbeans.conf` nastavit cestu k JDK verze 8. Tuto cestu provedeme změnou proměnné netbeans_jdkhome, do které nastavíme aboslutní cestu k souboru JDK verze 8: `netbeans_jdkhome="/usr/lib/jvm/java-1.8.0-openjdk-amd64"`. Následně v prostření NetBeans klikneme pravým tlačíkem myši na otevřený projekt `Anet` a zvolíme možnost `Build with Dependencies`. Poté co je aplikace sestavena, je možné ji zavést na server. Zavedení z prostředí NetBeans probíhá tak, že pravým klikem na pravé tlačíko myši klikneme na projekt `Anet-ear` a zvolíme možnost `Run`. Sestavování aplikace i samotné zavádění apliakce na server lze řešit i bez prostředí NetBeans a je popsáno v následujících kapitolách.

Příloha B

Sestavení aplikace pomocí nástroje Maven

Aplikaci sestavíme pomocí nástroje Maven. Aplikace byla vyvíjena v operačním systému Kubutnu 16.10 a následující řádky popisují konkrétní příkazy z operačního systému na kterém jsem aplikaci vyvíjel. V operačním systému Windows se mohou příkazy lišit.

Instlace sestavovacího nástroje probíhá spuštěním příkazu:

```
$ sudo apt install maven
```

Po úspěšném nainstalování sestavovacího nástroje Maven, aplikaci sestavíme tak, že přejdeme do adresáře ve kterém jsou zdrojové soubory k jednotlivým modulům aplikace:

```
$ cd /cesta-k-adresari-s-aplikaci/Anet/
```

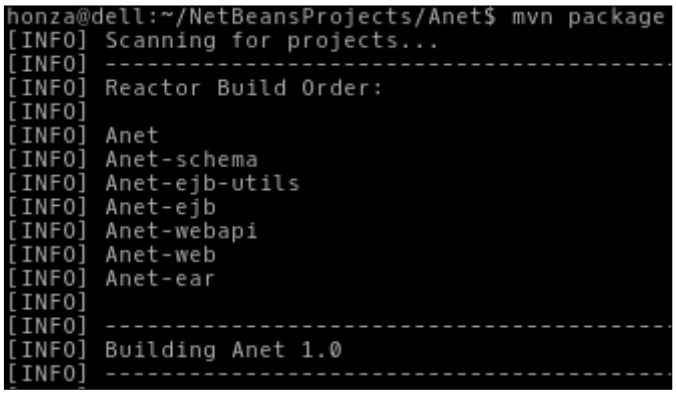
A v tomto adresáři spustíme příkaz, který moduly aplikace sestaví. Při prvním sestavení využijeme příkaz zajišťující stažení závislostí z centrálního repozitáře sestavovacího nástroje Maven - `install`. Před samotným příkazem `mvn install` je nastavena cesta k JDK verze 8, kterým budeme aplikaci sestavovat.

```
$ JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64 mvn install
```

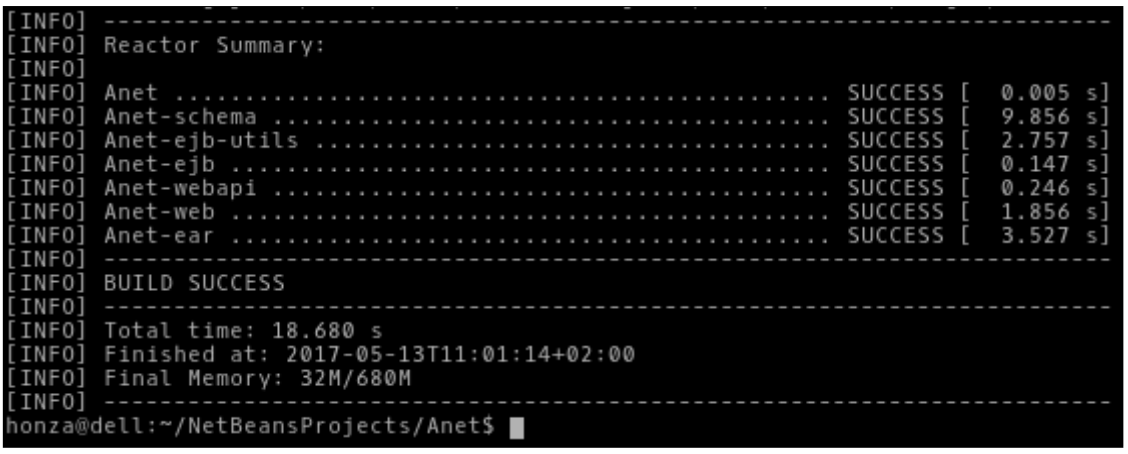
Pokud chceme odstranit již sestavené části a sestavit celou aplikaci znovu využijeme příkazu `mvn clean install`

```
$ JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64 mvn clean install
```

Na obrázku B.2 je zobrazený výpis, který sestavovací nástroj vypíše na standardní výstup při úspěšném sestavení. V případě, že se sestavení povedlo, máme připravený archiv typu `.ear`, který lze zavést na aplikační server. Tento archiv je umístěn v podadresáři `Anet`, který se jmenuje `Anet-ear`. Jeho přesné umístění je zde `/cesta-k-adresari-s-aplikaci/Anet/Anet-ear/target/Anet-ear-1.0.ear`



Obrázek B.1: Spuštění sestavovacího nástroje Maven



Obrázek B.2: Výpis po úspěšném sestavení aplikace pomocí nástroje Maven

Příloha C

Zavedení na aplikační server

V předchozím textuB je popsáno, jak získat archiv `.ear`, který lze zavést na aplikační server[1]. V této příloze bude popsán návod, jak spustit aplikační server a sestavenou aplikaci na něj zavést.

Jako aplikační server, na kterém aplikace poběží byl zvolen server Payara[17]. Jedná se o odnož aplikačního serveru Glassfish. Proč byl vybrán tento aplikační server, je podrobněji popsáno v kapitole 2.1.2, ve které jsou popsány i některé další aplikační servery.

Zvolený aplikační server je možné stáhnout na internetové adrese http://www.payara.fish/downloads[2]. Další text, který se týká bližšího popisu spuštění a zavedení aplikace na aplikační server, bude ukazován pro operační systém Kubuntu 16.10. Příkazy pro systém Windows se mohou lišit.

Po rozbalení aplikačního serveru do adresáře je třeba nastartovat doménu, na kterou budeme aplikaci zavádět. Název této domény je určen podle názvu adresáře umístěného v adresáři `domains`, přesněji `/cesta-k-aplikačnímu-serveru/glassfish/domains`. Zde se bude pravděpodobně vyskytovat přednastavená doména aplikačního serveru `payaradomain`. Tento název domény použijeme sestavení příkazu pro nastartování domény:

```
1 $ /cesta-k-aplikačnímu-serveru/glassfish/bin/asadmin start-domain payaradomain
```

Po nastartování domény je možné spustit administrační rozhraní aplikačního serveru, které běží na standardním portu 4848. Po nastartování domény aplikačního serveru by na adrese `localhost:4848` mělo být zobrazené administrační rozhraní aplikačního serveru. Pokud doména aplikačního serveru běží, je možné na ni zavést připravenou aplikaci. V levém menu hlavní nabídky vybereme záložku `Applications`. Na kartě Applications, klikneme v horní části administračního rozhraní na tlačíko `Deploy`. Po tomto kliknutí se dostaneme na stránku, kde budeme moci vybrat soubor, který chceme zavést na aplikační server. Vybereme soubor `Anet-ear-1.0.ear` umístěný `/cesta-k-adresari-s-aplikaci/Anet/Anet-ear/target/` a kliknutím na tlačíko `Ok` ve spodní části obrazovky zavedeme soubor na aplikační server[3].

[1] Stejně tak, je pro zavedení na aplikační server vhodný archiv typu „.war“

[2] Aplikace byla vyvíjena na konkrétní verzi aplikačního serveru 4.1.1.171.1

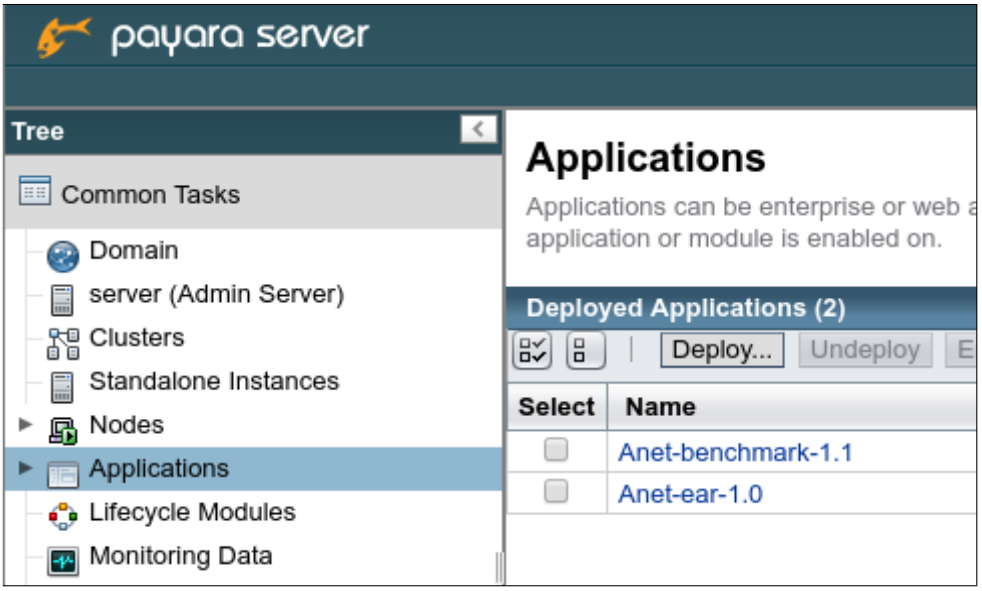
[3] V případě, že již stejná aplikace na serveru existuje a pouze nahráváme upravenou verzi aplikace, je potřeba zaškrtnout možnost „Force Redeploy:“

Obrázek C.1: Administrační rozhraní aplikačního serveru

Po úspěšném zavedení bude aplikace zobrazena v administračním rozhraní, tak jak je zobrazeno na obrázku C.2. A samotnou aplikaci nalezneme na standardním portu 8080. Bude tedy dostupná na adrese localhost:8080.

V případě chyby při zavádění aplikace doporučuji odstranit veškeré aplikace z aplikačního serveru. Restartovat server. Provést nové sestavení všech částí aplikace s odstraněním původních sestavených částí aplikace[4] a nově sestavenou aplikaci zavést na aplikační server.

[4]V prostředí NetBeans se jedná o funkci Clean and Build



Obrázek C.2: Administrační rozhraní aplikačního serveru

Příloha D

Obsah přiloženého paměťového média

- **Anet.zip** - archiv se zdrojovými soubory demonstrační aplikace.
- **Anet.ear** - sestavený archiv připravený pro nasazení na aplikační server.
- **AS.zip** - použitý aplikační server.
- **DP** - adresář obsahující zdrojové soubory od technické zprávy.
- **DP.pdf** - technická zpráva.