Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Mobilní aplikace pro univerzitní knihovnu

Bc. Petr Ilek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Petr Ilek

Systémové inženýrství a informatika
Informatika

Název práce

Mobilní aplikace pro univerzitní knihovnu

Název anglicky

Mobile app for university library

Cíle práce

Diplomová práce je tématicky zaměřena na problematiku vývoje mobilní aplikace pro univerzitní knihovní systém. Hlavním cílem práce je navrhnout a naprogramovat aplikaci pro platformy Android a iOS pomocí programovacího jazyka JavaScript a frameworku React Native.
Dílčí cíle práce jsou:
- Popsat postupy vývoje aplikace
- Popsat použité technologie

Metodika

Diplomová práce sestává ze dvou částí – teoretické a praktické.

Metodika zpracování teoretické části práce je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce.

V praktické části bude proveden návrh a implementace klient/server aplikace. Klientská část aplikace bude implementována pomocí programovacího jazyka JavaScript a frameworku React Native. Implementace serverové části bude provedena s využitím programovacího jazyka Java a frameworku Spring. Proces tvorby aplikace bude popsán a budou shrnuty zkušenosti z vývoje aplikace.

Doporučený rozsah práce

60-80 stran

Klíčová slova

react native, javascript, aleph, programování, mobilní aplikace

---

Doporučené zdroje informací

EISENMAN, Bonnie. Learning React Native – Building native mobile apps with JavaScript. Second Edition. O'Reilly Media, 2017. ISBN 978-1-491-98914-2.

CHERNY, Boris. Programming TypeScript: Making Your JavaScript Applications Scale. USA: O'Reilly Media, 2019. ISBN 978-1492037651.

ZAMMETTI, Frank. Practical React Native: Build Two Full Projects and One Full Game using React Native. Pottstown: Apress, 2018. ISBN 978-1-4842-3939-1.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 06. 04. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Mobilní aplikace pro univerzitní knihovnu" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 6. 4. 2020 _____

Poděkování

Rád(a) bych touto cestou poděkoval vedoucímu diplomové práce panu Ing. Jiřímu Brožkovi, Ph.D. za vstřícný přístup a věcné připomínky. Dále bych chtěl také poděkovat všem blízkým, kteří mě podporovali v průběhu práce.

Mobilní aplikace pro univerzitní knihovnu

Abstrakt

Teoretická část práce se zabývá problematikou vývoje mobilních aplikací. Velká pozornost je věnována multiplatformnímu vývoji aplikací pomocí technologie React Native. Na konci teoretické části je popsána technologie Spring a knihovní systém Aleph.

Praktická část začíná analýzou a návrhem mobilní aplikace. V další části je navržena komunikace mezi mobilní aplikací a systémem Aleph. Následná implementace se rozděluje do dvou částí – serverové a klientské.

V serverové části je navrženo a implementováno webové API pomocí technologií Java a Spring. Klientská část se zabývá implementací mobilní aplikace komunikující s knihovním systém Aleph. Výstupem práce je aplikace pro platformy Android a iOS, která má usnadnit uživatelům knihovny přístup ke čtenářskému účtu, katalogu a aktualitám knihovny.

Klíčová slova: React Native, Aleph. Spring, Java, JavaScript, Expo, Android, iOS

Mobile app for university library

Abstract

Topic of this thesis is about developing mobile apps. Theoretical part is focused on cross platform development with technology React Native. End of theoretical part contains basic overview about Spring technology and library system Aleph.

Practical part begins with analysis of mobile app. Next chapter continues with wireframe and infrastructure design. Implementation is divided into two parts – client side and server side.

Server-side part is focused on developing web API with Java and Spring technologies.

Client-side part contains implementation of mobile app which communicates with library system Aleph. The result of this thesis is Android and iOS app which should help library users with accessing their library account and searching through library catalog.

Keywords: React Native, Aleph. Spring, Java, JavaScript, Expo, Android, iOS

Obsah

Seznam obrázků

1 Úvod

Od příchodu iPhonu v roce 2007 se změnil způsob používání mobilních telefonů. V roce 2009 použila firma Apple v reklamě na iPhone 3G dnes již zlidovělé rčení „Na to existuje aplikace!“. I když tehdy bylo dostupných přibližně 15 tisíc aplikací, až intenzivní rozvoj v následujících letech vytvořil nové odvětví softwarového průmyslu. V dnešní době dle odhadů existuje již okolo dvou milionů mobilních aplikací. [1] [2]

S ohledem na aktuální trendy vznikla na ČZU potřeba vytvoření mobilní aplikace univerzitní knihovny pro hlavní platformy mobilního odvětví – Android a iOS. Mobilní aplikace bude uživatelům poskytovat jednodušší přístup ke čtenářskému účtu, katalogu, aktualitám knihovny a dalším informacím z knihovního systému Aleph. Tato práce popisuje komplexní moderní přístup k tvorbě mobilních aplikací, představuje řadu postupů a technologií, včetně multiplatformní technologie React Native, a jejich praktickou implementaci použitou v mobilní aplikaci univerzitní knihovny.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je tematicky zaměřena na problematiku vývoje mobilní aplikace pro univerzitní knihovní systém. Hlavním cílem práce je navrhnout a naprogramovat aplikaci pro platformy Android a iOS pomocí programovacího jazyka JavaScript a frameworku React Native.

Dílčí cíle práce jsou:

- Popsat postupy vývoje aplikace
- Popsat použité technologie

2.2 Metodika

Diplomová práce sestává ze dvou částí – teoretické a praktické.

Metodika zpracování teoretické části práce je založena na studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce.

V praktické části bude proveden návrh a implementace klient/server aplikace. Klientská část aplikace bude implementována pomocí programovacího jazyka JavaScript a frameworku React Native. Implementace serverové části bude provedena s využitím programovacího jazyka Java a frameworku Spring. Proces tvorby aplikace bude popsán a budou shrnuty zkušenosti z vývoje aplikace.

3 Teoretická východiska

3.1 Vývoj mobilních aplikací

Mobilní aplikace je typ aplikačního softwaru, který je určen pro mobilní zařízení. Dnes se mezi mobilní zařízení řadí převážně chytré mobily, hodinky a tablety. [3] [4]

Mobilní aplikace jsou základem mobilní ekonomiky. Od příchodu iPhonu v roce 2007 a distribuční platformy App Store v roce 2008 se změnil způsob používání mobilních telefonů. Zlidovělé rčení „na to existuje aplikace“ použil Apple v roce 2009 v reklamě na iPhone 3G, tehdy bylo na App Store dostupných 15 tisíc aplikací. O rok později jich bylo přes 140 tisíc a dnes jich Apple registruje zhruba 1,8 milionu. Došlo k vytvoření nového odvětví softwarového průmyslu. [2] [1]

Vývoj aplikací pro mobilní platformy se stal finančně výhodný nejen pro velké firmy, ale i pro nezávislé vývojáře. Počet vyvinutých a stažených aplikací tak roste každým rokem. Vývoj mobilní aplikace se trochu liší od vývoje desktopových aplikací, protože existuje více typů aplikací. [5]

3.1.1 Typy mobilních aplikací

Mobilní aplikace se dělí na 4 typy: webové, nativní, hybridní a progresivní webové aplikace.

1.1.1.1 Nativní aplikace

Aplikace vyvíjené pro jeden konkrétní operační systém, jako je například iOS nebo Android. Pro iOS se používá programovací jazyk Objective-C nebo Swift, pro Android je to Java nebo Kotlin. Jsou distribuovány prostřednictvím obchodů s aplikacemi pro danou platformu – na Androidu Obchod Play na iOS App Store.

Aplikace jsou nainstalovány na mobilní zařízení a fungují bez internetového připojení. Mají přístup k souborovému systému a funkcionalitám, kam patří například akcelerometr, Bluetooth, SMS atd. [6]

Výhody:

- Nevyžadují internetové připojení, protože jsou nainstalovány přímo v interní paměti zařízení.
- Rychlejší a responzivnější.

- Mají přístup k fotoaparátu, GPS, Bluetooth atd.
- Kvalitní vývojářské nástroje – Android Studio nebo XCode.

Nevýhody:

- Náročné na údržbu – s příchodem nových verzí OS mohou přestat fungovat.
- Vyšší náklady na vývoj – nutné vyvíjet pro každou platformu zvlášť. [6]

1.1.1.2 Webové aplikace

Aplikace, které nejsou instalovány na zařízení a mohou být spuštěny prostřednictvím nativního internetového prohlížeče. Jedná se o interaktivní a responzivní webové stránky optimalizované pro mobilní zařízení. Data aplikace nejsou uložena v paměti zařízení, proto je vyžadováno neustálé připojení k internetu. Aplikace je závislá na kvalitě použitého internetového prohlížeče a nemůže být distribuována prostřednictvím obchodů s aplikacemi.

Výhody:

- Není je potřeba instalovat přímo do interní paměti zařízení.
- Fungují na desktopu i mobilních zařízeních.
- Nízké náklady na vývoj.
- Jeden zdrojový kód pro všechny platformy.
- Jednoduché aktualizace – stačí zveřejnit novou verzi na url aplikace.

Nevýhody:

- Nefungují bez internetového připojení.
- Menší výpočetní výkon.
- Omezený přístup k hardware.
- Nejsou distribuovány prostřednictvím obchodů s aplikacemi – menší dosah k uživatelům aplikace.

[6]

1.1.1.3 Hybridní aplikace

Hybridní aplikace jsou webové aplikace zabalené do nativní aplikace. Spouští se v enginu nativního internetového prohlížeče pomocí komponenty WebView. Hybridní aplikace přistupují k nativnímu API pomocí JavaScriptu. Na rozdíl od webových aplikací je možné pomocí tohoto API přistupovat například k akcelerometru nebo fotoaparátu.

Distribuují se stejně jako nativní aplikace prostřednictvím platforem Obchod Play nebo AppStore. [7] [8]

Výhody:

- Méně nákladný a rychlejší vývoj.
- Přístup k fotoaparátu a GPS.
- Jeden zdrojový kód pro všechny platformy.

Nevýhody

- Většinou vyžadují internetové připojení.
- Omezený přístup k systémovým zdrojům. [6]

1.1.1.4 Progresivní webové aplikace

Progresivní webové aplikace (zkráceně PWA) jsou webové aplikace, které se chovají a vypadají jako nativní aplikace. PWA reprezentují novou filozofii pro tvorbu webových aplikací. Na první pohled není zřejmé, zda se jedná o normální nebo progresivní webovou aplikaci. Aplikace je považována za progresivní, pokud má následující atributy:

- Responzivnost.
- Nezávislost na internetovém připojení.
- Stejná interakce jako s nativní aplikací.
- Šifrovaná pomocí TLS.
- Může být nainstalována na plochu zařízení.
- Aktualizuje se na pozadí.
- Umožňuje posílat push notifikace.
- Dostupnost prostřednictvím internetových vyhledávačů.

Mezi nevýhody progresivních webových aplikací patří větší spotřeba baterie a omezený přístup k systémovým zdrojům (např. Bluetooth). [9] [6]

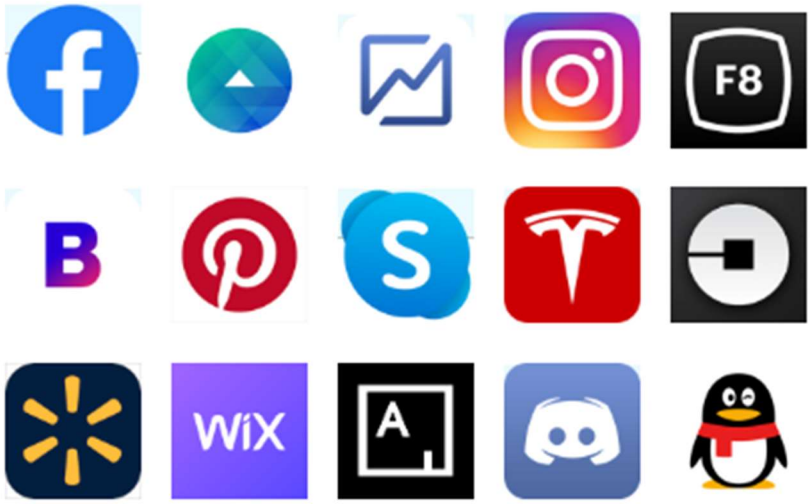
3.2 Multiplatformní vývoj mobilních aplikací

Technologie se v programování neustále vyvíjí, což má za následek dodání kvalitních produktů v kratším čase. Vývojové nástroje, které toto umožňují mají za úkol abstrahovat složité problémy při tvorbě aplikací. Dříve byli vývojáři mobilních aplikací ohledně multiplatformního vývoje velmi skeptičtí. V porovnání s multiplatformním vývojem byl vývoj nativní aplikace mnohem náročnější, ale nativní aplikace měla vždy mnohem vyšší kvalitu. S příchodem multiplatformních nástrojů React Native, Xamarin a Flutter však došlo ke zlomu. [10]

3.3 React Native

React Native je JavaScriptový framework pro psaní nativních mobilních aplikací pro iOS a Android. React Native vychází z ReactJS – velmi populární JavaScriptové knihovny pro tvorbu webových uživatelských rozhrání od společnosti Facebook. [11]

React Native vznikl jako vedlejší projekt na interním hackathonu. K oficiálnímu představení došlo v březnu 2015 na F8 konferenci. React Native dnes pohání populární mobilní aplikace jako jsou: Facebook, Airbnb, Discord, Instagram, Walmart, Gyroscope, Wix a Skype. [11]



Obrázek 1: Aplikace používající React Native, zdroj:https://reactnative.dev/

Aplikace jsou v React Native vyvíjené pomocí speciální syntaxe JavaScriptu nazývané JSX. Výsledná mobilní aplikace je vykreslena pomocí nativních UI komponent. Tím se React Native odlišuje od ostatních multiplatformních frameworků jako je Cordova nebo Ionic, které používají pro vykreslování WebView. React Native také umožňuje

přistupovat k funkčnostem mobilních zařízení jako je kamera nebo GPS pomocí JavaScriptového API. [12]

3.3.1 Výhody

- Vzhled a pocit z aplikace naprogramované v React Native je mnohem blíže nativnímu, než u jiných frameworků (např. Cordova a Ionic).
- Většina konceptů je shodná s webovým Reactem.
- Sdílený kód pro více platforem, z čehož plynou nižší náklady na vývoj.
- Hot Reloading.
- Změny v JavaScriptovém kódu nemusí procházet schvalovacím procesem Applu a Googlu.
- Možnost psaní nativního kódu, který lze následně volat z JavaScriptu.

3.3.2 Nevýhody

- Zpožděná podpora nových nativních API – potřeba čekat na aktualizaci rozhrání mezi React Native a nativním kódem.
- Horší debuggování jakožto důsledek přidání React Native vrstvy. [11]

3.4 API

API je zkratka pro Application Programming Interface. Používá se v softwarovém inženýrství pro označení sbírek procedur, funkcí, tříd, protokolů a nástrojů pro tvorbu software. API je typ rozhrání, které umožňuje programátorům přistupovat k určitým funkčnostem nebo datům aplikací, operačních systémů nebo jiných služeb. [13]

3.5 Javascript

V květnu 1995 představil Brendan Eich skriptovací programovací jazyk s názvem JavaScript. Jeho účelem bylo vytváření programů ve webových stránkách pro prohlížeč Netscape Navigator. Od té doby byl jazyk integrován do všech grafických internetových prohlížečů. To mělo za následek příchod webových aplikací, se kterými je možné interagovat bez nutnosti přenačtení stránky pro každou akci. JavaScript je také používán na tradičních webových stránkách k poskytnutí rozmanitých forem interaktivity. [14] [15]

V současnosti je JavaScript lehký, dynamicky typovaný, interpretovaný skriptovací jazyk. Jedná se o multiparadigmatický jazyk, podporuje imperativní, funkcionální a objektově orientované styly programování. Používá se i mimo internetový prohlížeč například v aplikacích běžících na technologii Node.js. [16]

3.6 ECMAScript

ECMAScript je skriptovací jazyk normovaný neziskovou organizací Ecma International podle standardu ECMA-262. Mezi nejznámější implementace tohoto standardu patří JavaScript, JScript a ActionScript. [14]

Název ECMA původně vznikl jako zkratka pro „European Computer Manufacturers Association“, což v překladu znamená „Evropská asociace výrobců počítačů“. K reflektování globálních aktivit došlo v roce 1994 ke změně jména organizace na Ecma International. [17]

Od roku 2012 všechny moderní prohlížeče plně podporují ECMAScript 5.1. V červnu 2015 byla vydána šestá verze ECMAScriptu pojmenována ECMAScript 2015, ale známá také jako ECMAScript 6 nebo ES6. Od té doby vychází standardy ECMAScriptu v ročních cyklech. [16]

3.7 TypeScript

TypeScript je programovací jazyk vyvíjený jako OpenSource společností Microsoft. Lze jej považovat za nadmnožinu JavaScriptu, protože podporuje všechny jeho funkce a přidává k nim něco navíc. Výstupem kompilace TypesScriptu je JavaScript. [18]

Vlastnosti TypeScriptu:

- Všechny benefity z ECMAScriptu 2016.
- Statické typování.
- Generiky.
- Modifikátory přístupu.

3.8 Node.js

Node.js je platforma založená na programovacím jazyku JavaScript. Její autor Ryan Dahl ji představil v roce 2009 na konferenci JSConf. Platforma byla navržena pro psaní vysoce škálovatelných internetových aplikací. Umožňuje spouštět JavaScriptový kód mimo

internetový prohlížeč. Dokáže tak pracovat se souborovým systémem nebo databázovými systémy. [11]

3.9 Node Package Manager – NPM

NPM je správce balíčků pro programovací jazyk JavaScript. Jeho instalace probíhá společně s Node.js. NPM používá soubor package.json k ukládání metadat o projektu a jeho závislostech. [11]

3.10 Expo

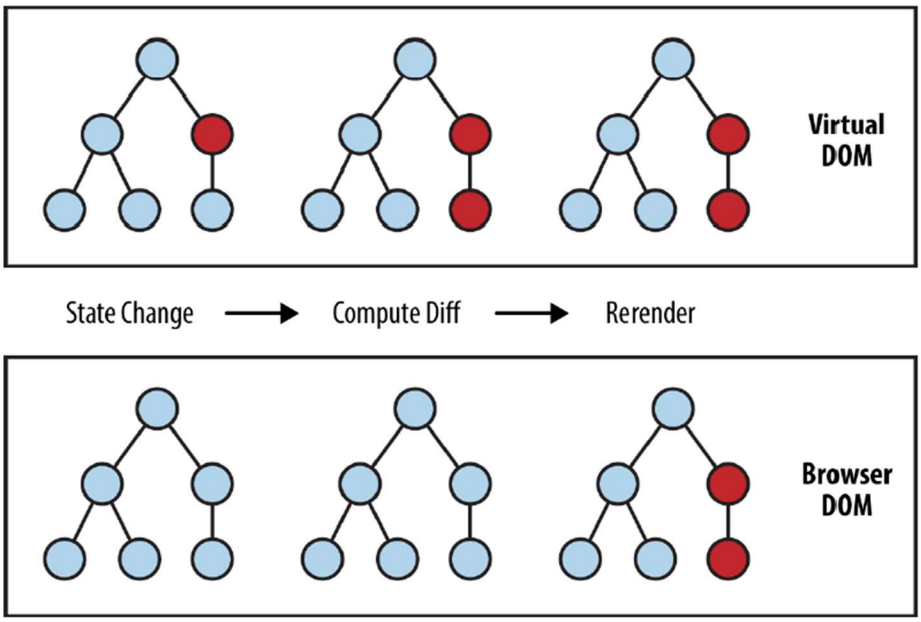
Nástroj Expo umožňuje vyvíjet React Native aplikace bez přítomnosti vývojových prostředí XCode nebo Android Studio. Cílem nástroje je eliminovat problémy při vývoji aplikace s nasazením do mobilního zařízení pomocí mobilní aplikace Expo Client. [12]

3.11 Koncepty React Native

3.11.1 Virtual DOM

DOM neboli „Document Object Model“ (v překladu „objektový model dokumentu“) je aplikační programové rozhraní, pomocí kterého lze přistupovat k jednotlivým objektům XML dokumentu a pracovat s nimi. Mezi tyto objekty řadíme elementy, atributy a text. DOM používá stromovou datovou strukturu a umožňuje na přesně určené místo přidat jakýkoli obsah. Nejčastěji se používá pro manipulaci HTML elementů ve webových aplikacích s interaktivním uživatelským rozhraním. Samotná editace DOMu je výpočetně náročná a má vliv na výkon aplikace. [19]

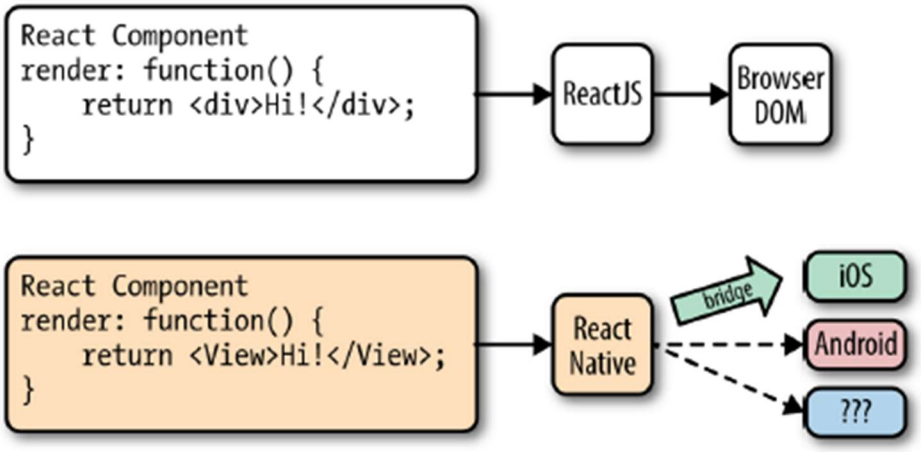
ReactJS i React Native používá programátorský koncept virtuálního objektového modelu dokumentu – *Virtual DOM*. Zde je reprezentace uživatelského rozhraní uložena v paměti a synchronizována se skutečným objektovým modelem dokumentu. ReactJS tedy vypočítá nezbytné změny DOMu v paměti a následně v prohlížeči překreslí jen ty elementy, které se změnily (viz. obrázek níže). [12]



Obrázek 2: Výpočet rozdílu ve Virtual DOM a překreslení DOMu v prohlížeči, zdroj [12]

3.11.2 **React Native Bridge**

ReactJS poskytuje abstraktní vrstvu mezi zdrojovým kódem vývojáře a vykreslováním v prohlížeči. React Native tuto abstraktní vrstvu modifikuje pomocí tzv. *bridge.* Bridge místo vykreslování elementů v DOMu prohlížeče volá nativní vykreslovací API dané platformy. Na iOS se volá API v programovacích jazycích Objective-C a Swift, na Androidu pak API v jazycích Java a Kotlin (viz. obrázek níže). Protože React Native neběží na hlavním vlákně pro uživatelské rozhrání, provádí tato volání asynchronně, aniž by narušoval uživatelský zážitek. Oficiální verze React Native Bridge podporuje pouze platformy iOS a Android, existují však i implementace pro Windows, Ubuntu a Web. [12]



Obrázek 3: Vykreslování ReactJS a React Native, zdroj: [12]

3.11.3 **JSX**

JSX je zkrácené označení pro JavaScript XML, JSX umožňuje vkládat XML značky do JavaScriptu, aniž by se jednalo o syntaktickou chybu. Používá se jak v ReactJS, tak v React Native. Soubory, které používají JSX mají koncovku *„.jsx“*. JSX je transformováno do běžného JavaScriptu, který je následně spuštěn. Pro zavolání JavaScriptového kódu uvnitř XML elementu se používají složené závorky. [11]

Následující ukázka zdrojového kódu v JSX vykreslí HTML element: „ Hello World “.

```
class HelloMessage extends React.Component {
    render() {
        return <div>Hello {this.props.name}</div>;
    }
}

ReactDOM.render(<HelloMessage name="World" />, mountNode);
```

Stejného výsledku lze dosáhnout bez JSX viz ukázka: [12]

```
class HelloMessage extends React.Component {
    render() {
        return React.createElement(
            "div",
            null,
            "Hello ",
            this.props.name
        );
    }
}

ReactDOM.render(
    React.createElement(HelloMessage, { name: "World" }), mountNode);
```

3.11.4 **Komponenty**

Komponenty jsou v Reactu soběstačné kusy zdrojového kódu, které zapouzdřují popis vizuálních elementů – zahrnují jejich stav, atributy a operace, které mohou provádět. Všechny komponenty jsou potomci třídy React.Component a vždy musí implementovat metodu render(). Metoda render() vrací jeden kořenový element, který popisuje danou komponentu. Tento element může obsahovat další potomky. [11]

Komponenty lze definovat dvěma způsoby:

1. **pomocí třídy z ECMAScriptu 6:**

```
class Hello extends React.Component {
    render() {
        return <Text>Hello World</Text>;
    }
}
```

2. **pomocí JavaScriptové funkce**

```
function Hello(props) {
    return <Text>Hello World</Text>;
}
```

Existují dva typy dat, které komponenta používá **Props** a **State**:

Props

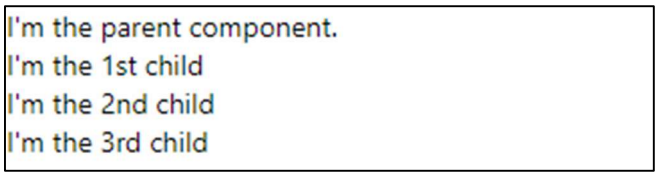
Props je speciální objekt komponenty, jehož název je odvozený ze slova properties – v překladu vlastnosti. Props označují neměnná data (immutable), která jsou předávána nadřazenou komponentou. Je důležité zdůraznit, že toto předávání funguje pouze jednosměrně – z rodiče do potomka. Hodnoty uvnitř Props jsou určené pouze ke čtení, data z rodiče tedy nemohou být měněna potomkem. Do Props je možné vkládat primitivní hodnoty, elementy nebo funkce. [20] [21]

Props se předávají jako XML atribut. V komponentě se k nim pak přistupuje pomocí tečkové notace, viz. ukázka.

```
class ParentComponent extends React.Component {
    render() {
        return (
            <View>
                <Text>I'm the parent component.</Text>
                <ChildComponent text={"I'm the 1st child"} />
                <ChildComponent text={"I'm the 2nd child"} />
                <ChildComponent text={"I'm the 3rd child"} />
            </View>
        );
    }
}

class ChildComponent extends React.Component {
    render() {
        return (
            <View>
                <Text>{this.props.text}</Text>
            </View>
        );
    }
}
```

Po vykreslení komponenty ParentComponent z ukázky výše dojde k zobrazení textu na obrázku dále. [20]



Obrázek 4: Vykreslená komponenta ParentComponent, zdroj: [20]

State

State je speciální objekt komponenty, který reprezentuje vnitřní stav komponenty. Obsahuje proměnlivá data (mutable), která mohou být v budoucnosti změněna pomocí metody setState(). Inicializace stavu se provádí v nepovinném konstruktoru komponenty. [11]

```
class Person extends React.Component {
    constructor(props) {
        super(props);
        this.state = { name: '' };
    }
    render() {
        return (
            <Text onPress={() => this.setState({ name: 'Petr' })}>
                My name is:{this.state.name}
            </Text>
        );
    }
}
```

Deklarovaná komponenta Person v ukázce výše má ve State inicializovanou proměnnou *name* jako prázdný řetězec. Hodnota této proměnné je zobrazena v komponentě Text za statickým textem „My name is:“. Po vykreslení je zatím tedy zobrazen pouze statický text „My name is:“.

Komponenta Text má definovanou metodu onPress(), která se zavolá po stisknutí komponenty. V těle metody onPress() se pomocí metody setState() nastaví hodnota proměnné *name*, což vyvolá překreslení komponenty (viz. další kapitola). Po stistknutí komponenty Text tedy dojde k zobrazení textu: „My name is:Petr“. [21]

3.11.5 Životní cyklus komponent

Komponenty mají definovaný životní cyklus metod, které je možné přetížit a spustit tak kód v dané době procesu. Komponenta může procházet třemi procesy: Mounting, Updating a Unmounting. [22]

Mounting

V tomto procesu dochází k vytváření instance komponenty pomocí konstruktoru. Ten nastaví Props a State komponenty, která je následně vložena do DOMu a zobrazena pomocí metody render(). Nakonec je inicializace ukončena voláním metody componentDidMount(). [22]

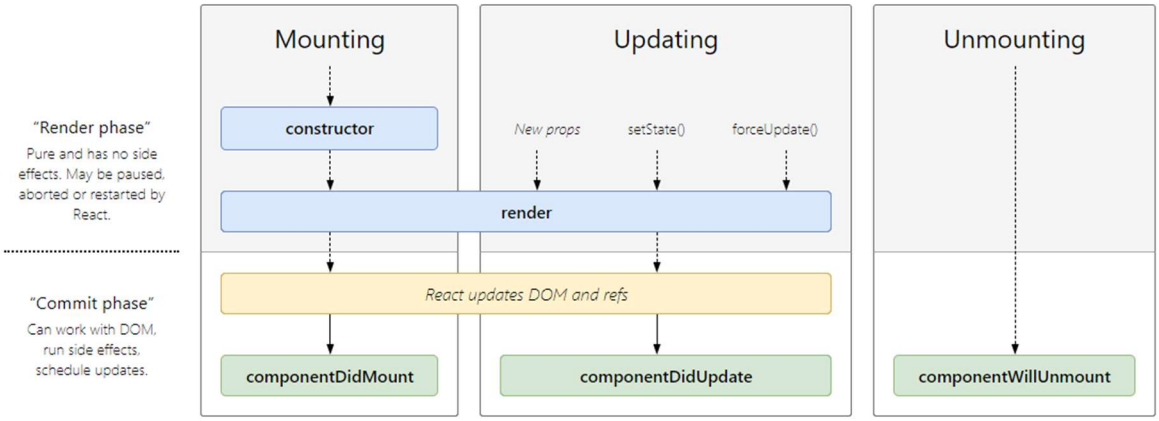
Updating

Tento proces je nejčastěji spuštěn při změně stavu (State) komponenty pomocí metody setState() nebo změnami ve vlastnostech (Props) nadřazené komponenty. V procesu dochází k volání metody render(), která vrátí aktualizovanou vizuální reprezentaci. Následně dojde k překreslení komponenty a provolání metody componentDidUpdate(). Pokud chceme

překreslování komponenty zabránit je možné přetížit metodu shouldComponentUpdate(). Proces Updating je možné vynutit pomocí metody forceUpdate(). [22]

Unmounting

K volání toho procesu dojde, pokud je komponenta odebrána z DOMu.



Obrázek 5: Metody životního cyklu, které jsou nejčastěji přetěžovány, zdroj:http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

3.11.6 **Základní komponenty**

1.1.1.5 View

Základním stavebním kamenem pro tvorbu uživatelského rozhrání je komponenta View. Na obrazovce je zobrazena jako obdélníková plocha a zodpovídá za vykreslování a dotykovou reakci pomocí definovaných událostí (eventů). Při vykreslení se zobrazí její ekvivalent pro danou platformu, například u iOS bude zobrazeno UIView. Komponenta View může mít vnořeno nula nebo více komponent, tyto komponenty mohou tvořit hlubokou hierarchii k dosažení požadovaného rozložení. Komponenta podporuje stylování pomocí Flexboxu prostřednictvím vlastnosti (Props) *style*. [11]

1.1.1.6 Text

Komponenta Text je v mnoha ohledech podobná komponentě View, akorát je specializovaná pro zobrazení textu. Podporuje vkládání dalších komponent a reakci na dotykové události pomocí zpětného volání metod *onPress* a *onLongPress*. Stylování komponenty pomocí vlastnosti *style* je uzpůsobeno pro text, není tedy provedeno pomocí flexboxu. V praxi to znamená, že se vnořené komponenty vkládají na konec řádku. [11]

1.1.1.7 Image

Komponenta Image slouží k zobrazení obrázků. Podporuje statické obrázky ze souborového systému nebo obrázky stažené přes síť. Komponenta umožňuje reagovat na události prostřednictvím zpětného volání metod onLoad(), onLoadStart(), onLoadEnd() a onError(). Komponenta Image poskytuje statické metody mezi nejpoužívanější lze zařadit metody: getSize() a prefetch(). Pomocí getSize() je možné zjistit rozměry obrázku. Metoda prefetch() slouží k načtení obrázku do paměti. [11]

1.1.1.8 Button

Button je základní interaktivní tlačítko, které umožňuje nastavit barvu pozadí, textu, a reakci na jeho stisknutí. [12]

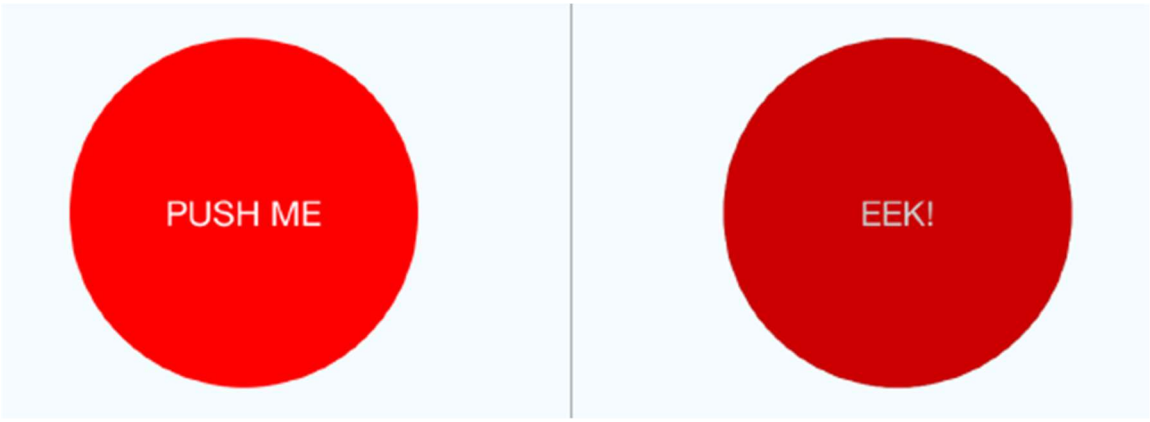
1.1.1.9 TouchableHighlight

Mobilní platformy mají své normy pro interakce. Ty se mohou lišit podle dané platformy, iOS se při dotyku může chovat jinak než Android. React Native poskytuje API pro tvorbu dotykových rozhrání. [12]

Každý element, který reaguje na dotyk uživatele by měl být obalen komponentou TouchableHighlight. Tato komponenta překryje vnořené komponenty vrstvou, která poskytuje uživateli nativní vizuální odezvu na dotyk. Pomocí zpětného volání metod onPressIn(), onPressOut() a onLongPress() je možné reagovat na počátek dotyku, konec dotyku a dlouhý dotyk. [12]

V následující ukázce zdrojového kódu je pomocí komponenty TouchableHighlight implementováno tlačítko, které poskytuje uživateli vizuální odezvu při stisknutí. Stavová proměnná *pressing* říká, zda je tlačítko v daném čase stisknuté. Tato proměnná se nastavuje při zpětném volání onPressIn() a onPressOut() prostřednictvím metody setState().

```
class Button extends React.Component {
    constructor(props) {
        super(props);
        this.state = { pressing: false };
    }
    _onPressIn = () => {
        this.setState({ pressing: true });
    };
    _onPressOut = () => {
        this.setState({ pressing: false });
    };
    render() {
        return (
            <View style={styles.container}>
                <TouchableHighlight
                    onPressIn={this._onPressIn}
                    onPressOut={this._onPressOut}
                    style={styles.touchable}
                >
                    <View style={styles.button}>
                        <Text style={styles.welcome}>
                            {this.state.pressing ? "EEK!" : "PUSH ME"}
                        </Text>
                    </View>
                </TouchableHighlight>
            </View>
        );
    }
}
```



Obrázek 6: Nativní odezva na stisknuté tlačítko, zdroj: autor

3.11.7 **Stylování komponent**

V React Native se stylování všech hlavních komponent provádí pomocí proměnné *style* v Props. Názvy stylů a jejich hodnoty jsou velmi podobné jako u kaskádových stylů na webu, hlavní rozdíl je v použití velbloudí notace (Camel case). Místo background-color

v CSS se v React Native používá backgroundColor. Dalším rozdíl se nachází v rozložení prvků, to je provedeno výhradně prostřednictvím Flexboxu. Hodnota proměnné *style* je JavaScriptový objekt, ten může obsahovat i pole dalších objektů se styly, kde má poslední prvek přednost. Styly je možné vkládat inline nebo pomocí objektu StyleSheet. Tento objekt má metodu create(), která umožňuje definovat více stylů v jednom objektu. Do proměnné *style* je pak předána reference na daný styl prostřednictvím tečkové notace. [11] [23]

Ukázka použití inline stylů:

```
export default class App extends React.Component {
    render() {
        return (
            <View style={{
                flex: 1,
                alignContent: 'center',
                justifyContent: 'center',
                backgroundColor: '#4287f5',
                textAlign: 'center',
                alignItems: 'center',
            }}>
                <Text style={{
                    color: 'yellow',
                    fontWeight: 800
                }}>StyleSheet object</Text>
            </View>
        );
    }
}
```

Ukázka definování stylů pomocí objektu StyleSheet a metody create():

```
export default class App extends React.Component {
    render() {
        return (
            <View style={styles.container}>
                <Text style={styles.text}>StyleSheet
object</Text>
            </View>
        );
    }
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignContent: 'center',
        justifyContent: 'center',
        backgroundColor: '#4287f5',
        textAlign: 'center',
        alignItems: 'center'
    },
    text: {
        color: 'yellow',
        fontWeight: 800
    }
});
```

3.11.8 **Kompozice vs dědičnost**

Některé komponenty neznají své potomky (vnořené komponenty) předem. To je typické například při tvorbě menu nebo dialogů, které reprezentují generické obálky. React používá návrhový vzor kompozice, umožňuje tak pomocí proměnné *children* v Props přistupovat ke vnořeným komponentám JSX elementu dané komponenty. Props a kompozice poskytují veškerou flexibilitu, která je potřebná pro úpravu vzhledu a chování komponent. Tvůrci Reactu proto doporučují vyhnout se používání dědičnosti napříč komponentami. [24]

Mějme funkcionální komponentu FancyBorder, která obalí své vnitřní komponenty rámečkem pomocí proměnné *children* v Props:

```
function FancyBorder(props) {
    return (
        <View style={styles.fancyBorder}>
            {props.children}
        </View>
    );
}

const styles = StyleSheet.create({
    fancyBorder: {
        alignContent: 'center',
        justifyContent: 'center',
        padding: 8,
        borderColor: '#4287f5',
        borderWidth: 10,
    },
});
```

Ukázka použití komponenty FancyBorder, která obalí všechny své vnořené elementy text rámečkem:

```
export default class App extends React.Component {
imageUri= 'https://image.shutterstock.com/free-sample-217430497.jpg';

render() {
  return (
   <FancyBorder>
     <Text>Text s rámečkem a obrázkem obalený komponentou FancyBorder</Text>
     <Image source={{uri: this.imageUri}}/>
   </FancyBorder>
  );
 }
}
```

Obrázek 7: Obalení textu rámečkem pomocí komponenty FancyBorder, zdroj: autor

3.11.9 **React Refs**

React Refs zprostředkovávají přímý přístup k instancím komponent deklarovaných v JSX. Umožňují pracovat s komponentami imperativním způsobem pomocí referencí. To se hodí například pro vyvolání přesunu kurzoru (focus) na specifickou komponentu nebo k tvorbě animací. [25]

K vytvoření reference je potřeba použít metodu createRef() třídy React. Poté se provede inicializace reference atributem ref v JSX dané komponenty. Aktuální reference je pak dostupná pod vlastností *current* pomocí tečkové notace. [25]

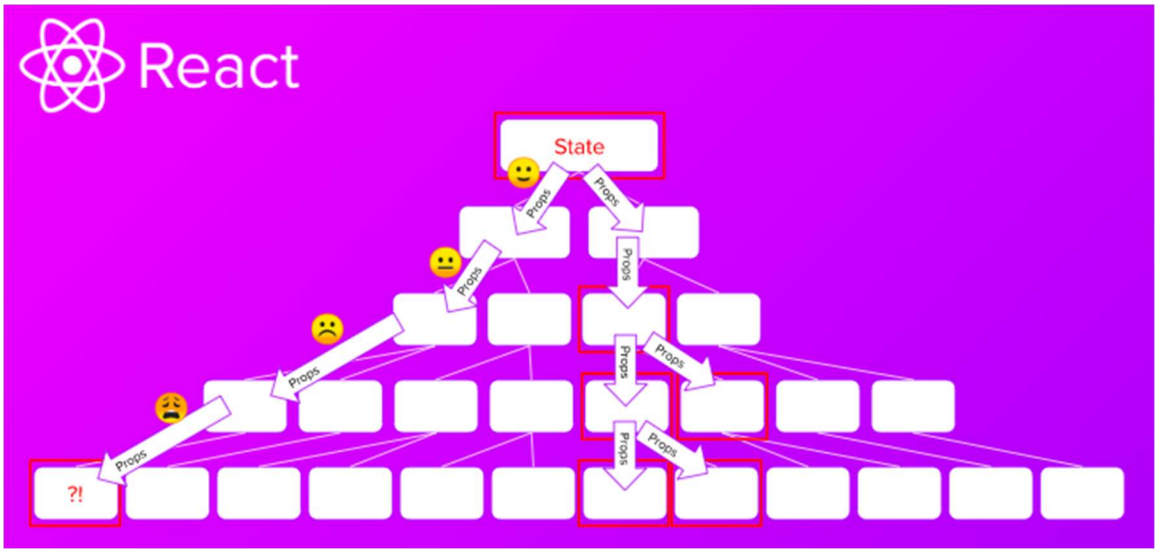
V následující ukázce byla vytvořena reference na textové pole pod názvem *myRef*. Po stisknutí tlačítka s popiskem „Focus Input“ dojde k zavolání metody focus() na textovém poli, což způsobí přesun kurzoru na textové pole.

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.myRef = React.createRef();
    }

    render() {
        return (
            <View>
                <Button title="Focus Input"
                        onPress={() => this.myRef.current.focus()}/>
                <TextInput ref={this.myRef}/>
            </View>
        );
    }
}
```

3.11.10 **React Context API**

Jak již bylo zmíněno, k přenášení dat mezi komponentami ve stromové struktuře slouží Props. Props ale umožňují přenášet data pouze na úrovni jednoho uzlu stromové struktury tedy z nadřazené komponenty do podřazené komponenty. Pokud je potřeba předat data přes více úrovní musí být data postupně projít přes všechny komponenty v podstromu.



Obrázek 8: Předávání props přes komponenty, zdroj: [26]

React Context API umožňuje přenášet data mezi komponentami ve stromové struktuře, bez nutnosti předávání dat přes každý uzel. Přenášenými daty mohou být například přihlášený uživatel, nastavení vzhledu nebo preferovaný jazyk. Pokud dojde ke změně dat v nadřazené komponentě aktualizují se tyto data i v podřazené komponentě. [26] [27]

Nejprve je potřeba vytvořit Context objekt pomocí metody createContext():

```
const MyContext = React.createContext(defaultValue);
//Hodnota parametru default value je použita, pouze pokud není ve stromě
//nalezen Context provider
```

Ke Context objektu se definuje Provider, který umožní přijímat změny v Context objektu. Provider přijímá vlastnost *prop* s názvem *value*, její hodnota bude předána do všech potomků providera:

```
<MyContext.Provider value={/* nějaká hodnota – proměnná, funkce, objekt */}>
```

Ke zpřístupnění hodnoty Context objektu v komponentách podřazeným Context provideru je nejprve potřeba inicializovat třídní statickou proměnnou contextType a přiřadit jí referenci na definovaný Context objekt. Poté je komponenta připravená přistupovat k hodnotám v Context objektu prostřednictvím třídní proměnné *context*. [27]

V následující ukázce je vytvořen Context objekt s názvem *MyContext*. Ve třídě *App* je použit jeho provider, který poskytuje všem podřazeným komponentám hodnotu objektu *contextValue*. Ve třídě Consumer je nejprve zpřístupněn Context *MyContext* pomocí třídní statické proměnné *contextType*. Třídá Consumer vykresluje tlačítko, v jeho popisku je

použita hodnota z Context objektu *someValue*. Při stisknutí tlačítka dojde k provolání funkce sayHello() z Context objektu.

```
const MyContext = React.createContext(null);

export default class App extends React.Component {
    contextValue = {
        sayHello: () => {
            alert('hello from context');
        },
        someValue: 'value from context',
    };

    render() {
        return (
            <MyContext.Provider value={this.contextValue}>
                <View>
                    <View>
                        <Text>Test Context API</Text>
                        <Consumer/>
                    </View>
                </View>
            </MyContext.Provider>
        );
    }
}

class Consumer extends React.Component {
    static contextType = MyContext;

    render() {
        return (
            <Button onPress={this.context.sayHello}
                    title={this.context.someValue} />
        );
    }
}
```

Obrázek 9: Vykreslená komponenta Consumer (vlevo) a interakce po stisknutí jejího tlačítka (vpravo), zdroj: autor

3.11.11 **Funkcionální komponenty**

Jak již bylo zmíněno komponenty je možné vytvářet i pomocí funkcí. Takové komponenty jsou v dokumentaci označovány jako tzv. functional components –

funkcionální komponenty. Props se do těchto komponent předává jako parametr funkce. Návratovou hodnotou funkce je kořenový element komponenty (stejně jako u metody render() v komponentě tvořené třídou). [22]

Funkce v následující ukázce vykreslí pozdrav pro jméno předané v Props.

```
function Hello(props) {
    return (
        <Text>
            Hello:{props.name}
        </Text>
    );
}
```

Hlavní výhodou funkcionálních komponent je menší množství napsaného kódu a přehlednost. Dříve funkcionální komponenty neuměly pracovat s vnitřním stavem komponenty (State), sloužily tak pouze pro bezestavové (stateless) komponenty. To se změnilo s příchodem Hooks v Reactu 16.8. [28]

3.11.12 **React Hooks**

React od verze 16.8 přináší zásadní novinku React Hooks. Takzvané hooky vznikly za účelem vyřešení následujících problémů:

a) Sdílení logiky mezi komponentami
b) Nepraktický způsob práce s životním cyklem komponent
c) Implementace komponent s vnitřním stavem je možná pouze pomocí tříd

React Hooks jsou speciální funkce, které umožňují funkcionálním komponentám používat State a reagovat na životní cyklus komponenty. Použitím React Hooks už není potřeba vytvářet komponenty jako třídy a přetěžovat metody životního cyklu componentDidMount(), componentDidUpdate(), atd. [29] [30]

React Hooks se mohou volat pouze z funkcionálních komponent, nesmějí se volat uvnitř cyklů, podmínek nebo vnořených funkcí. [29]

1.1.1.10 UseState

Jak již bylo zmíněno k inicializaci vnitřního stavu (State) třídní komponenty je potřeba vytvořit konstruktor:

```
class Example extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            count: 0
        };
    }
}
```

Ve funkcionální komponentě není možné přistupovat k proměnné *state*. Místo toho se použije funkce useState(). Nepovinným vstupním parametrem této funkce je počáteční hodnota stavu, která se nastaví pouze při prvním vykreslení komponenty. Návratovou hodnotou funkce useState() je pole, kde první prvek představuje stavovou proměnnou a druhý prvek je funkce pro změnu této stavové proměnné. Prvky tohoto pole se používají stejně jako proměnná *state* a metoda setState() u třídní komponenty. [30]

V následující ukázce je definována komponenta vykreslující tlačítko s textem, kolikrát na něj bylo kliknuto. Pomocí rozloženého přiřazení (Destructuring assignment) jsou uloženy do proměnných *count* a *setCount* prvky z pole, které vrací funkce useState(). Proměnná *count* reprezentuje počet kliknutí s počáteční hodnotou rovnou nule. Pro změnu stavu slouží funkce setCount(), při jejím zavolání si React tuto hodnotu uloží a vyvolá překreslení. Při vykreslování React poskytne aktuální hodnotu prostřednictvím proměnné *count*. [30]

```
function Example() {
    const [count, setCount] = React.useState(0);

    return (
        <View>
            <Button onPress={() => setCount(count + 1)}
                    title={`Clicked times: ${count}`} />
        </View>
    );
}
```

1.1.1.11 UseEffect

React Hook useEffect je funkce, která pokrývá procesy životního cyklu komponenty: mounting, updating a unmounting. Tato funkce se v praxi používá, pokud je potřeba provádět stejnou logiku při tvorbě komponenty (mounting) a při jejím překreslení (updating). [31]

Parametrem funkce useEffect() je funkce s požadovanou logikou. Do návratové hodnoty funkce je možné vložit funkci, která se zpětně zavolá při odebírání komponenty z DOMu (unmounting). [31]

```
React.useEffect(() => {
    // volá se při tvorbě komponenty a jejím překreslení
    return () => {
        // volá se při odebírání komponenty z DOMu
    };
});
```

Funkce useEffect() má nepovinný druhý parametr, vkládá se do něj pole s hodnotami, při jejichž změnách se má vykonat požadovaná logika z prvního parametru. [31]

```
React.useEffect(() => {
    // volá se při tvorbě komponenty a jejím překreslení, ale
    // pouze pokud se změní hodnota proměnné someValue
}, [someValue]);
```

Pokud je požadováno vykonat požadovanou logiku pouze při inicializaci komponenty vkládá se do druhého parametru prázdné pole. [31]

1.1.1.12 UseContext

React Hook useContext je funkce, která funkcionálním komponentám umožňuje přístup k definovanému React Contextu. Slouží ke zpřístupnění aktuální hodnoty Context objektu v podřazených komponentách. Jediným parametrem funkce je Context objekt. [25]

Komponenta Consumer z poslední ukázky v kapitole o React Contextu by pomocí funkcionální komponenty a React Hooks vypadala následovně:

```
function Consumer() {
    const context = React.useContext(MyContext);

    return (
        <Button onPress={context.sayHello}
                title={context.someValue} />
    );
}
```

1.1.1.13 UseReducer

React Hook useReducer je funkce, která se velmi podobá funkci useState. Používá se ke správě vnitřního stavu komponent se složitou stavovou logikou. K jeho použití je nejprve potřeba definovat funkci označovanou názvem *reducer()*. [25]

Tato funkce spravuje změny stavu podle vývojářem definovaných akcí. Jejím prvním parametrem je aktuální stav komponenty. Druhý parametr představuje akci, podle které se má změnit stav. Aktualizovaný stav musí být vrácen jako návratová hodnota.

V následující ukázce je definován *reducer*, který na základě akcí *increment* nebo *decrement* zvyšuje nebo snižuje hodnotu stavové proměnné *count* o 1. [25]

```
function reducer(state, action) {
    switch (action.type) {
        case 'increment':
            return {count: state.count + 1};
        case 'decrement':
            return {count: state.count - 1};
        default:
            throw new Error();
    }
}
```

Funkce useReducer() přijímá dva parametry, prvním je *reducer*, druhým pak počáteční hodnota stavu. Návratovou hodnotu zastává pole, kde první prvek představuje stavovou proměnnou a druhý prvek funkci s názvem *dispatch()*. Tato funkce slouží k vyvolání změny stavu podle definovaných akcí ve funkci *reducer()*. [25]

```
const [state, dispatch] = useReducer(reducer, initialState);
```

Komponenta Counter v ukázce níže umožňuje pomocí tlačítek „-“ a „+“ měnit hodnotu zobrazenou v popisku. Po stisknutí tlačítka dojde k provolání funkce *dispatch*, která díky *reduceru* provede změnu hodnoty v popisku.

```
function Counter() {
    const [state, dispatch] = React.useReducer(reducer, {count: 0});
    return (
        <View>
            <Text>Count: {state.count}</Text>
            <Button onPress={() => dispatch({type: 'decrement'})}>-</Button>
            <Button onPress={() => dispatch({type: 'increment'})}>+</Button>
        </View>
    );
}
```

1.1.1.14 UseMemo

Tento React Hook optimalizuje výkon aplikace. Slouží k uložení výsledku funkce náročné na výpočet. První parametr zastupuje funkci náročnou na výpočet, druhý parametr pak pole hodnot, při jejichž změnách má dojít ke znovu vykonání funkce v prvním parametru. [25]

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

1.1.1.15 UseRef

Hook useRef umožňuje funkcionálním komponentám vytvářet reference na komponenty. Tento hook má nepovinný parametr pro počáteční hodnotu. [25]

Komponenta MyComponent z ukázky v kapitole o React Refs by pomocí funkcionální komponenty a React Hooks vypadala následovně:

```
function MyComponent() {
    const myRef = React.useRef();

    return (
        <View>
            <Button title="Focus Input"
                    onPress={() => myRef.current.focus()} />
            <TextInput ref={myRef}/>
        </View>
    );
}
```

3.11.13 **React Navigation**

Většina mobilních aplikací pracuje pouze v jedné obrazovce. Knihovna React Navigation odstraňuje složité manuální skrývání jednotlivých obrazovek v aplikaci pomocí komponent označovaných jako navigátory. Navigátory obstarávají vykreslování záhlaví obrazovek s nadpisem a přechod mezi obrazovkami. [12]

Mezi nejpoužívanější navigátory patří:

- StackNavigator – přechod mezi obrazovkami definován v zásobníku, kde aktuálně zobrazená komponenta je na jeho vrcholu.
- BottomTabNavigator – přechod mezi obrazovkami definován ve spodní navigaci.

V následující ukázce je demonstrováno používání BottomTabNavigatoru:

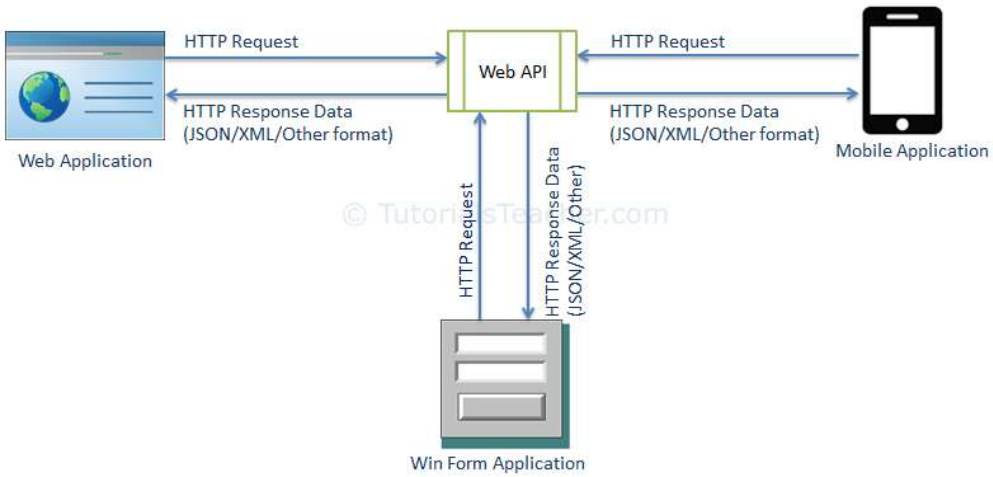
```
function MyTabs() {
    return (
        <Tab.Navigator>
            <Tab.Screen name="Home" component={HomeScreen} />
            <Tab.Screen name="Settings" component={SettingsScreen} />
        </Tab.Navigator>
    );
}
```

K přechodu mezi obrazovkami se používá objekt z hooku useNavigation(). K přechodu na jinou obrazovku se použije metoda navigate().

```
const navigation = useNavigation();
navigation.navigate("Home",{...params})
```

3.12 **Web API**

Web API, jak již napovídá samotný název je API přes web, které je dostupné pomocí HTTP protokolu. Jedná se o koncept, ne o technologii. K tvorbě Web API je možné použít různé technologie jako například Spring, .NET atd. Web API je rozšířená forma webové aplikace poskytující služby na různá zařízení jako jsou mobilní telefony, tablety, notebooky apod. [13]



Obrázek 10: Komunikace více zařízení s Web API, zdroj: [13]

3.13 **Spring Framework**

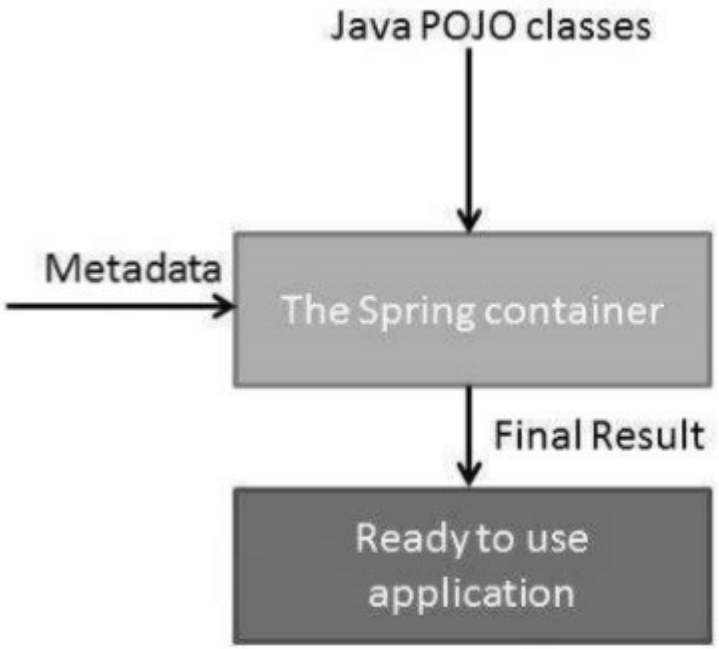
Spring Framework nabízí komplexní podporu infrastruktury pro vývoj aplikací v programovacím jazyku Java. Je vybaven funkcemi jako Dependency Injection a moduly zajišťujícími persistentní vrstvu, architekturu MVC, bezpečnost, testování apod. Tyto

moduly mohou dramaticky zkrátit čas potřebný k vývoji aplikace. Chování Spring aplikace je možné definovat pomocí Java anotací nebo XML souborů. [32]

3.13.1 **Spring Container**

Spring Container je jádrem frameworku Spring. Vytváří objekty, propojuje je, konfiguruje a spravuje jejich životní cyklus od vytvoření až po destrukci. Používá techniku Dependecy Injection ke správě komponent, ze kterých se skládá celá aplikace.

Pomocí konfiguračních metadat získává Spring Container instrukce pro objekty, které má vytvořit, konfigurovat a sestavit. Tyto metadata je možné reprezentovat XML souborem, Java anotacemi nebo Java kódem. Následující diagram prezentuje vysoko úrovňový pohled funkčnosti Spring Containeru. Použitím Java POJO (Plain Old Java Object) tříd a konfiguračních metadat produkuje nakonfigurovanou a spustitelnou aplikaci. [33]



Obrázek 11: Vysokoúrovňový pohled na Spring Container, zdroj: [33]

3.13.2 **Spring Boot**

Spring Boot je rozšíření frameworku Spring, které eliminuje standardní konfiguraci potřebnou pro spuštění Spring Aplikace. Spring Boot se skládá z:

- Automatické konfigurace a závislostí pro zjednodušení sestavení a konfigurace aplikace.
- Vestavěného serveru, který má za úkol usnadnit nasazení aplikace. [32]

3.13.3 Anotace pro tvorbu webového API

@Component

Tato anotace se používá nad názvem třídy k definování Spring komponenty. Říká Spring Containeru, aby automaticky detekoval tyto třídy (například pro Dependency Injection). Spring poskytuje tři specifičtější podtypy komponent:

- **@Service** – označuje třídu poskytující aplikační logiku.
- **@Repository** – označení pro třídy, které provádějí CRUD (Create, Read, Update, Delete) operace například při práci s databází.
- **@Controller** – používá se ve webových aplikacích pro označení komponenty, která vyřizuje požadavky na server a vrací odpověď. [34]

V následující ukázce je definována komponenta SampleComponent:

```
@Service
public class SampleService {
    //Logika
}
```

@Autowired

Anotací Autowired je možné označit konstruktor, třídní proměnnou, setter nebo konfigurační metody. Použitím anotace instruujeme Spring Container, že je potřeba vložit potřebnou závislost pomocí tzv. autowiringu. Autowiring je vlastnost, která umožňuje pomocí Dependency Injection automaticky vyhodnotit závislost objektů. Spring Container tak zajistí vložení závislosti podle typu objektu, který je předmětem anotace. [35]

V následující ukázce je vytvořen objekt třídy SampleComponent pomocí autowiringu:

```
@Autowired
private SampleService sampleService;
```

@RequestMapping

Anotace se používá pro mapování webových dotazů na metody. Povinnými parametry anotace je koncová URL a typ HTTP metody. [36]

@ResponseBody

Tato anotace říká Controlleru, že návratová hodnota metody má být automaticky serializována do formátu JSON a předána v těle HTTP odpovědi. [36]

@RequestBody

Anotace serializuje tělo HTTP dotazu do Java objektu. [36]

V následující ukázce je definován Controller, který při dotazu na koncovou URL „/sample“ zavolá metodu sample():

```
@Controller
public class SampleController {

    @Autowired
    private SampleService sampleService;

    @RequestMapping(value = "/sample", method = RequestMethod.POST)
    @ResponseBody
    public SomeClass sample(@RequestBody BodyClass requestBody) {
        SomeClass object = sampleService.doSomething(requestBody);
        return object;
    }
}
```

@SpringBootApplication

Touto anotací se definuje Spring Boot aplikace, která může být spuštěna jako standartní Java aplikace ve statické metodě main.

Ukázka:

```
@SpringBootApplication
public class Application {

   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }
}
```

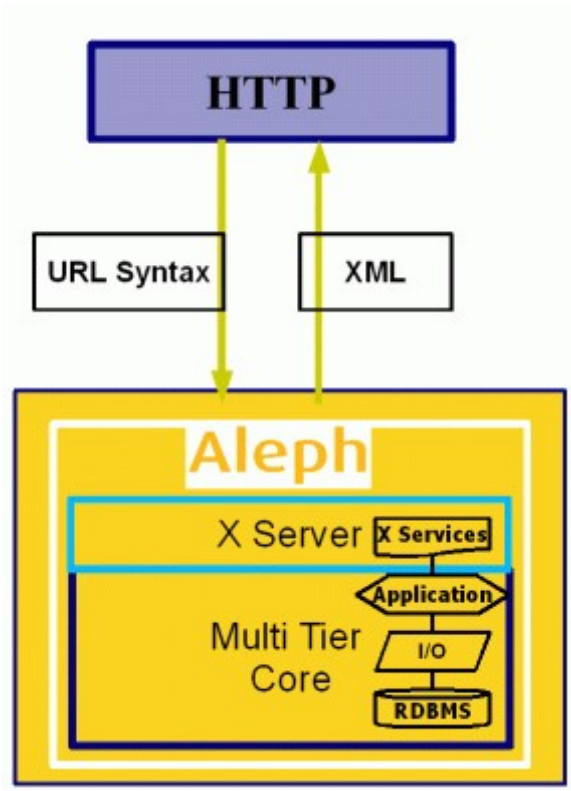
3.14 Knihovní systém ALEPH

Integrovaný knihovnický systém ALEPH je produktem společnosti ExLibris. Tento systém je velmi rozšířený po celém světe, v Evropě jej například používají v British Library a Národní dánské knihovně. V tuzemsku je možné se s tímto systémem setkat v Národní knihovně České republiky nebo na univerzitách. Systém ALEPH vznikl za účelem rychlého a efektivního poskytování informací. Zajišťuje všechny služby knihovny nejen pro knihovníky, ale i pro čtenáře. Mezi tyto služby patří například správa výpůjček a vyhledávání v katalogu knih včetně filtrování. [37]

3.14.1 ALEPH X-Server

Systém ALEPH umožňuje integraci na jiné systémy prostřednictvím komponenty X-Server. ALEPH X-Server je komponenta umožňující komunikaci s ostatními systémy přes

rozhrání ve formátu XML. Funguje to tak, že na X-Server je odeslán HTTP dotaz. V URL tohoto dotazu jsou specifikovány parametry a na základě jejich vyhodnocení vrací X-Server odpověď ve formátu XML. Dokumentace toho rozhrání je dostupná na webu systému Aleph [38].



Obrázek 12: Komunikace se systémem Aleph, zdroj: [38]

4 Vlastní práce

Tato část práce je zaměřena na návrh a implementaci mobilní aplikace pro univerzitní knihovní systém. Nejprve byla provedena analýza požadavků.

4.1 Analýza požadavků

Na základě osobních zkušeností s webovým rozhráním knihovního systému byly stanoveny požadavky pro mobilní aplikaci.

4.1.1 Funkční požadavky

Čtenářské konto

Aplikace zobrazí aktuální stav čtenářského konta přihlášeného uživatele. Bude možné zobrazit aktuální výpůjčky, rezervace na výpůjčky a historii výpůjček. Dále bude uživatel informován o blížícím se konci výpůjčky a případných pokutách za pozdní odevzdání výpůjček.

Katalog knih

Aplikace umožní vyhledávání knih v Katalogu České zemědělské univerzity (dostupný na: https://aleph.czu.cz/). Knihy bude možné najít podle názvu, autora a signatury.

Exempláře knihy

Pro nalezenou knihu z katalogu bude možné zobrazit jednotlivé exempláře včetně jejich dostupnosti.

Rezervace na výpůjčky

Pokud budou všechny exempláře, které je možné si vypůjčit domů zapůjčené, bude možné z aplikace vytvořit rezervaci na výpůjčku. Tato rezervace půjde později prostřednictvím aplikace zrušit, pokud budou splněny podmínky knihovny.

Prodloužení výpůjček

Aplikace umožní prodloužit aktuální výpůjčku, pokud budou splněny podmínky knihovny.

Aktuality studijního informačního centra

Aplikace umožní uživateli procházet aktuality z webu studijního informačního centra (dostupné z: https://lib.czu.cz/cs/r-8995-aktuality-sic).

Informace o knihovně

Aplikace poskytne informace o půjčování a vracení knih.

4.1.2 Nefunkční požadavky

Platformy

Aplikace bude dostupná na obou hlavních mobilních platformách – Android a iOS. Minimální verze operačního systému Android bude Android 5.0 Lollipop. Pro iOS bude aplikace dostupná od verze 10.

Stabilita a rychlost

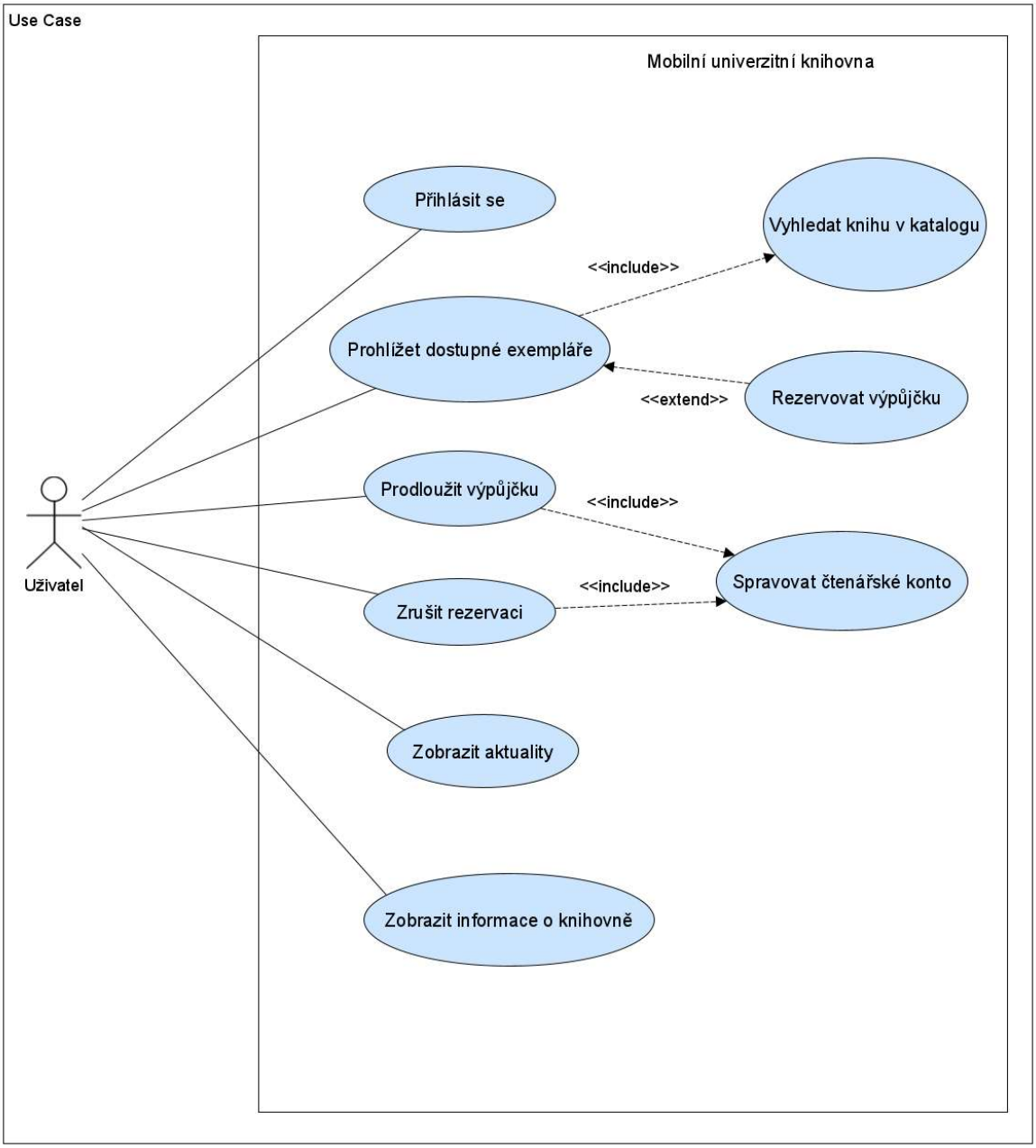
Aplikace by měla být stabilní a reagovat v přípustném čase.

Použitelnost

Používání aplikace by mělo být jednoduché a intuitivní.

4.2 Use Case diagram

Funkční požadavky byly vizualizovány pomocí Use Case diagramu, který popisuje chování mobilní aplikace z uživatelského pohledu. Aplikaci bude používat pouze jeden typ uživatele. Uživatel bude student registrovaný v knihovně univerzity, který bude moci provádět akce zobrazené v diagramu.



Obrázek 13: Use Case diagram, zdroj: Autor

4.3 Scénáře užití

4.3.1 Přihlásit se

Cíl: Přihlásit se do aplikace
Aktér: Uživatel
Vstupní podmínky: Uživatel má spuštěnou aplikaci
Výstupní podmínky: Uživatel je přihlášen do aplikace

Základní scénář:		
Krok	**Role**	**Akce**
1	Systém	Zobrazí formulář s přihlašovacím jménem a heslem
2	Aktér	Zadá své univerzitní přihlašovací údaje
3	Systém	Přihlásí uživatele, přesměruje ho na domovskou obrazovku a umožní navigaci v aplikaci
Alternativní scénář:		
Krok	**Role**	**Akce**
3.1	Systém	Pokud jsou zadané přihlašovací údaje špatné upozorní uživatele chybovou hláškou, uživatel zůstane na obrazovce s přihlášením

4.3.2 Spravovat čtenářské konto

Cíl: Spravovat čtenářské konto přihlášeného uživatele
Aktér: Uživatel
Vstupní podmínky: Uživatel je přihlášen v aplikaci
Výstupní podmínky: Je zobrazeno čtenářské konto přihlášeného uživatele

Základní scénář:		
Krok	**Role**	**Akce**
1	Aktér	Stiskne ve spodní navigaci tlačítko "Domů".
2	Systém	Načte ze serveru údaje o uživateli. Zobrazí tlačítka "Rezervace", "Výpůjčky" a "Historie výpůjček".
Alternativní scénář:		
Krok	**Role**	**Akce**
1.1	Systém	Pokud bude mít uživatel pokuty za nevrácené knihy, zobrazí textové upozornění.
1.2	Systém	Pokud bude mít uživatel výpůjčky s datem vrácením kratším než týden, zobrazí textové upozornění.

4.3.3 **Vyhledat knihu v katalogu**

Cíl: Vyhledat knihu na základě zadaných parametrů vyhledávání
Aktér: Uživatel
Vstupní podmínky: Uživatel je přihlášen v aplikaci a nachází se na obrazovce Katalog
Výstupní podmínky: Jsou zobrazeny výsledky vyhledávání

Základní scénář:		
Krok	**Role**	**Akce**
1	Systém	Zobrazí textové pole pro hledaný výraz a přepínač umožňující vyhledávání podle: názvu, autora, signatury
2	Aktér	Zvolí kritérium vyhledávání a zadá hledaný výraz
3	Systém	Zobrazí nalezené knihy a informace o jejich autorovi s rokem vydání
4	Aktér	Klikne na knihu
5	Systém	Zobrazí detail knihy
Alternativní scénář:		
Krok	**Role**	**Akce**
3.1	Systém	Pokud nebude nalezena žádná kniha, zobrazí textové upozornění

4.3.4 **Prohlížet dostupné exempláře**

Cíl: Prohlížet dostupné exempláře
Aktér: Uživatel
Vstupní podmínky: Uživatel je přihlášen v aplikaci a nachází se na obrazovce Katalog
Výstupní podmínky: Je zobrazen přehled o dostupnosti exemplářů v knihovně

Základní scénář:		
Krok	**Role**	**Akce**
1	Aktér	Přejde na detail požadované knihy, <<include>> (Vyhledat knihu v katalogu)
2	Aktér	Stiskne tlačítko "Exempláře"
3	Systém	Zobrazí exempláře (knihovní jednotky) knihy s hodnotami: typ jednotky, sbírka, signatura, čárový kód a dostupnost

4.3.5 **Rezervovat výpůjčku**

Cíl: Rezervovat výpůjčku
Aktér: Uživatel
Vstupní podmínky: Uživatel je přihlášen v aplikaci a prohlíží si exempláře
Výstupní podmínky: Uživatel vytvoří rezervaci výpůjčky

Základní scénář:		
Krok	**Role**	**Akce**
3.1	Systém	Pokud se bude jednat o exemplář, který je vypůjčený, ale lze jej rezervovat v systému Aleph, bude zobrazeno tlačítko „Rezervovat“
3.2	Aktér	Stiskne tlačítko "Rezervovat"
3.3	Systém	Provede rezervaci výpůjčky a zobrazí uživateli okno s hláškou „Byl jste zařazen/a do fronty na vypůjčení této publikace. Jakmile na Vás přijde řada, budete informován/a e-mailem a budete mít 6 pracovních dnů na vyzvednutí rezervované publikace. Pokud si nestihnete rezervovanou publikaci vyzvednout nebo zrušit do 6 pracovních dnů, bude rezervace zrušena a budete penalizován/a pokutou 10 Kč."

Scénáře pro následující případy užití vyplývají z drátěných modelů v další kapitole:

- prodloužit výpůjčku.
- zrušit rezervaci.
- zobrazit aktuality.
- zobrazit informace o knihovně.

4.4 Návrh uživatelského rozhrání

Na základě analýzy byl proveden návrh uživatelského rozhraní pomocí drátěných modelů (wireframes). K jejich tvorbě byl použit open-source program draw.io od společnosti JGraph. Draw.io poskytuje širokou knihovnu prvků pro návrh uživatelských rozhraní a diagramů.

4.4.1 Přihlášení a domovská obrazovka

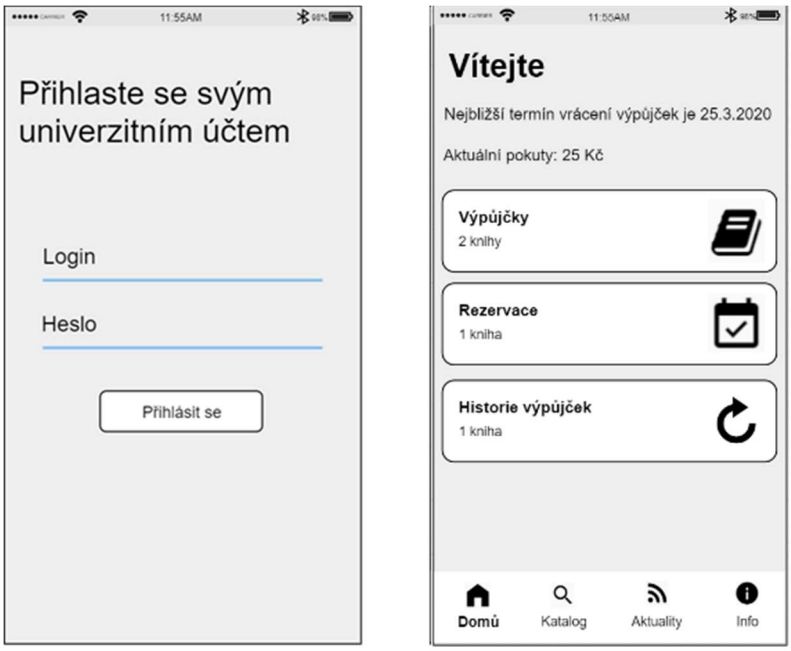
Nejprve se uživatel přihlásí pod svými univerzitními přihlašovacími údaji. Po přihlášení bude uživateli umožněna navigace v celé aplikaci pomocí tlačítek ve spodní navigaci. Tato navigace je součástí všech obrazovek a umožňuje navigaci na ostatní obrazovky stisknutím tlačítek:

- Domů – zobrazí domovskou obrazovku.
- Katalog – zobrazí katalog knih.
- Aktuality – zobrazí seznam s aktualitami.
- Info – zobrazí obrazovku s informacemi o knihovně.

Obrázek 14: Spodní navigace, zdroj: autor

Defaultně je zobrazena domovská obrazovka s informacemi o stavu čtenářského účtu. Pomocí tlačítek na domovské obrazovce je možný přechod na obrazovky:

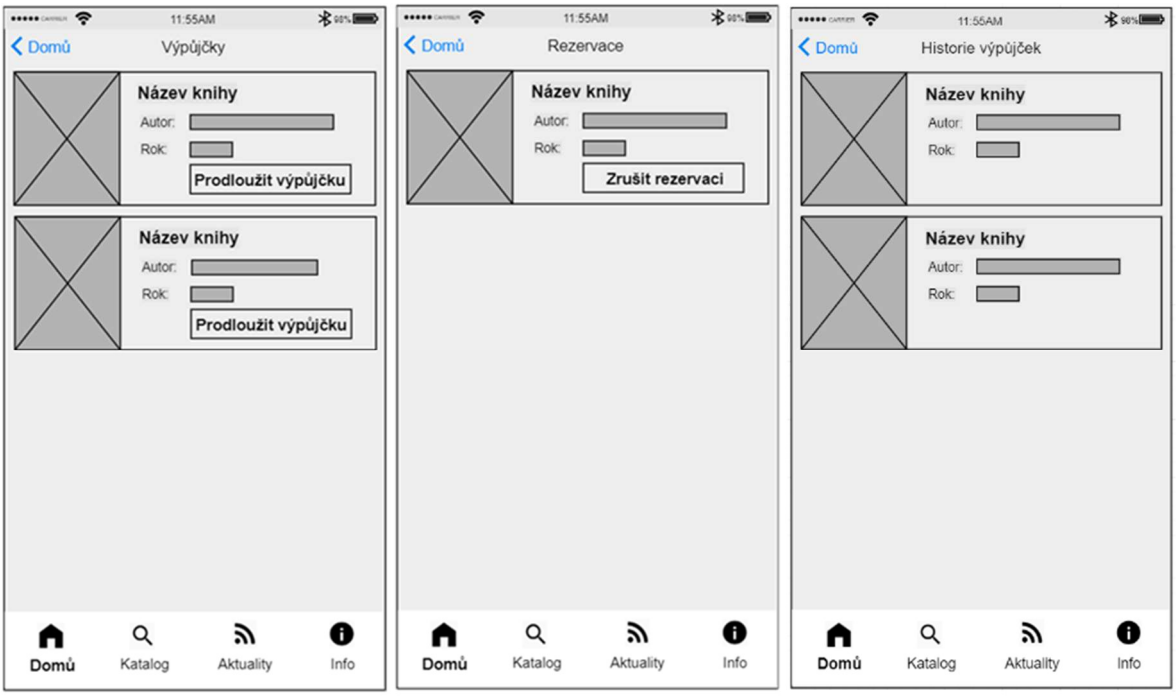
- Výpůjčky.
- Rezervace.
- Historie výpůjček.



Obrázek 15: Přihlašovací a domovská obrazovka, zdroj: autor

4.4.2 **Seznam výpůjček, rezervace a historie výpůjček**

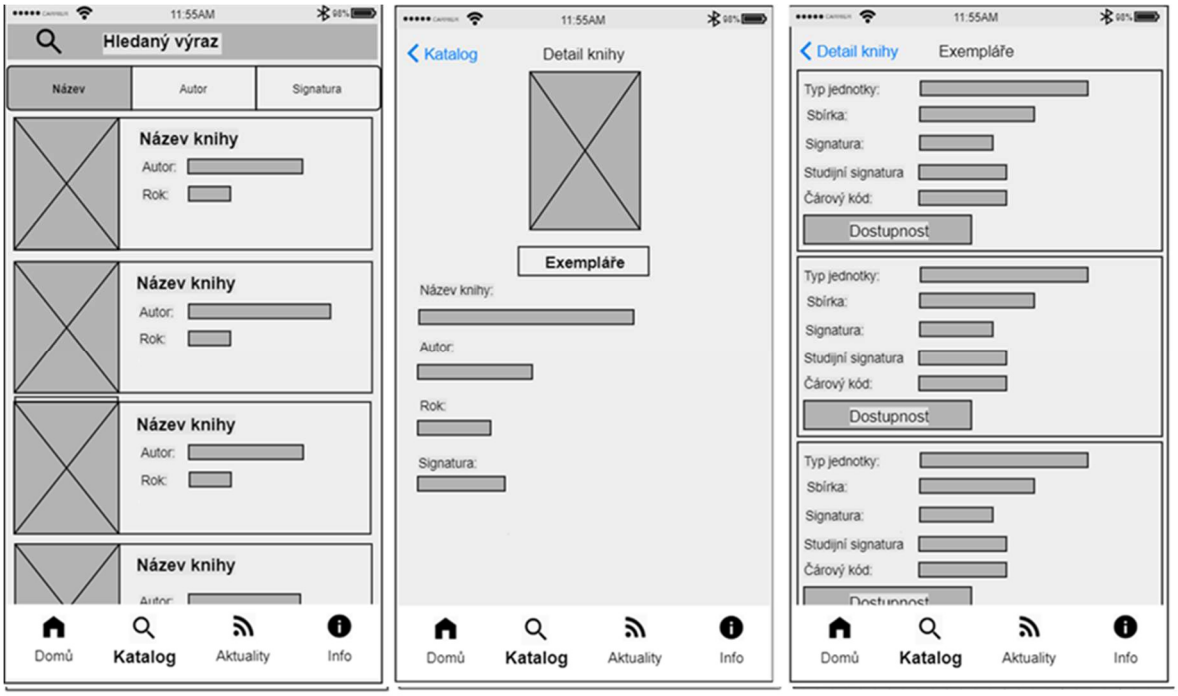
Na seznamu výpůjček může uživatel prodloužit výpůjčku pomocí tlačítka „prodloužit výpůjčku“. V seznamu rezervací lze rezervaci zrušit stisknutím tlačítka „zrušit rezervaci“.



Obrázek 16: Obrazovky seznam výpůjček, rezervace a historie výpůjček, zdroj: autor

4.4.3 Katalog knih

Do katalogu knih se uživatel dostane přes spodní navigaci. Do vyhledávacího pole zadá hledaný výraz a vybere si podle jakého kritéria chce vyhledávat. Pokud klikne na nalezenou knihu, zobrazí se její detail. Na detailu jsou zobrazeny podrobnější informace o knize a tlačítko exempláře. Pomocí tohoto tlačítka se uživatel přepne na obrazovku se seznamem exemplářů pro danou knihu. Tato obrazovka slouží především k zobrazení informací o dostupnosti jednotlivých exemplářů v univerzitní knihovně.

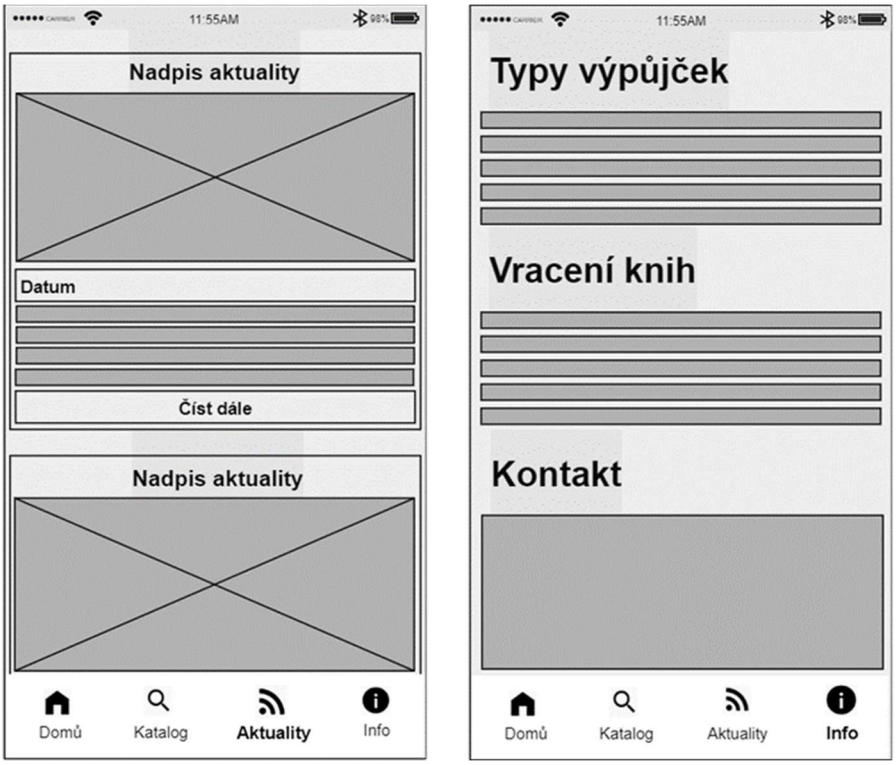


Obrázek 17:Obrazovky: Katalog knih, detail knihy a exempláře, zdroj: autor

4.4.4 Aktuality a informace

Obrazovka se seznamem aktualit zobrazí u každé aktuality její nadpis, obrázek, datum, krátký popis a tlačítko číst více. Po stisknutí tohoto tlačítka dojde k otevření webového prohlížeče se stránkou dané aktuality.

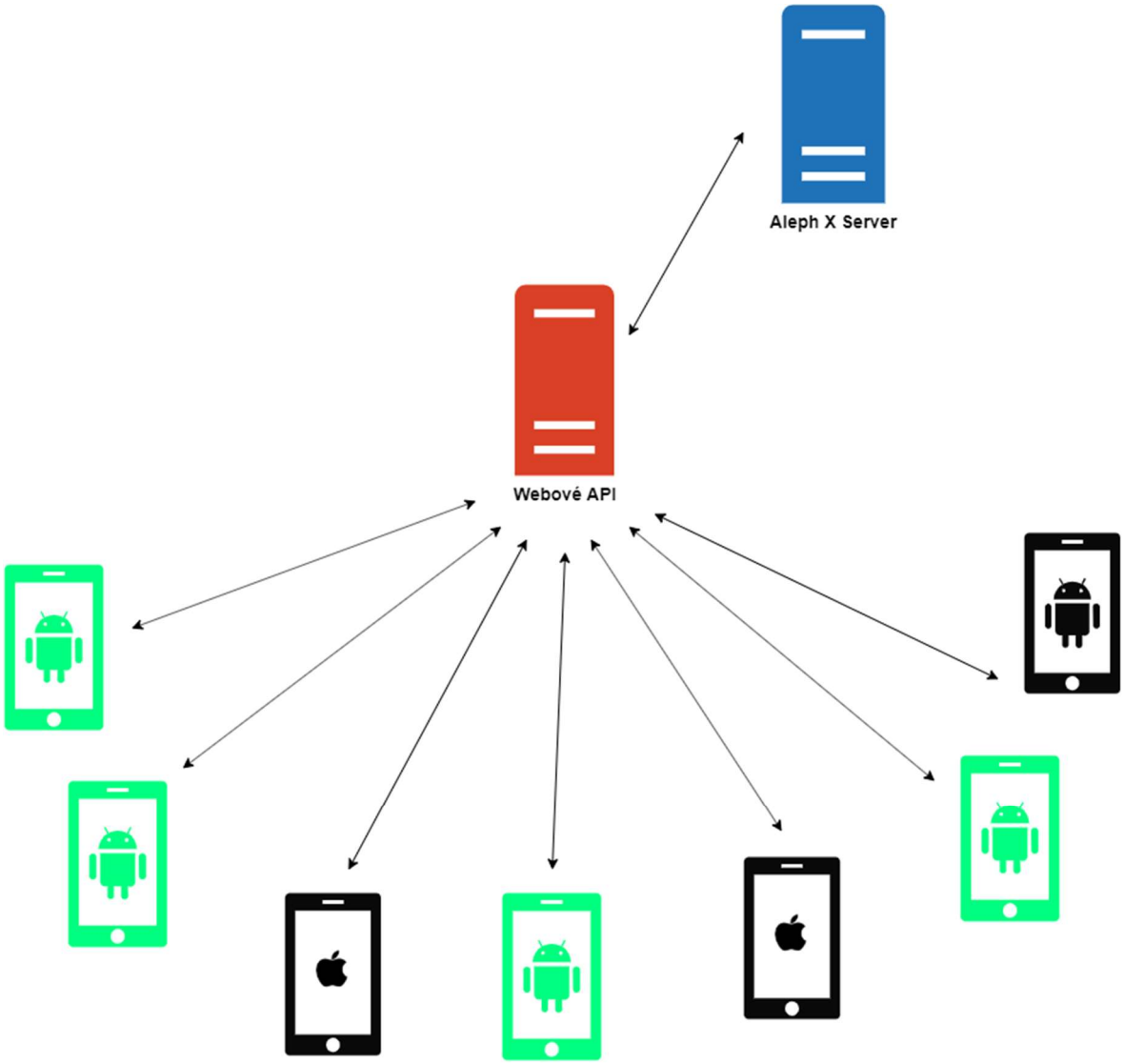
Obrazovka s informacemi informuje uživatele o typech výpůjček a podmínkách vrácení knih. Dále je ve spodní části obrazovky zobrazena mapa s lokací knihovny.



Obrázek 18: Obrazovky: Aktuality a Informace, zdroj: autor

4.5 Návrh infrastruktury

Před samotnou implementací mobilní aplikace bylo potřeba navrhnout, jak bude mobilní aplikace komunikovat s knihovním systémem Aleph. Knihovní systém Aleph poskytuje API pomocí komponenty označované X Server. Instance Alephu na ČZU povoluje přístup k API pouze povoleným IP adresám s omezeným počtem session. Z tohoto důvodu není možné volat API Alephu přímo z mobilních zařízení. K vyřešení tohoto problému byla navržena serverová aplikace, která poběží na serveru se schváleným přístupem k API Alephu. Tato aplikace složí jako prostředník mezi mobilní aplikací a knihovním systémem Aleph. Jejím hlavním úkolem bude transformovat XML odpovědi z Aleph X Serveru do čitelnější podoby ve formátu JSON. Komunikace mezi mobilní a serverovou aplikací bude probíhat pomocí webového API. Samotná implementace byla rozdělena do dvou částí, serverové a klientské.



Obrázek 19: Diagram infrastruktury, zdroj: autor

4.6 Implementace serverové aplikace

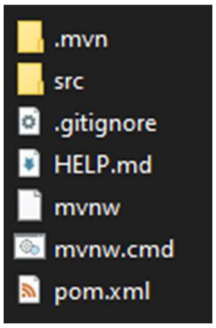
Serverová část byla zaměřena na tvorbu webového API, se kterými bude následně komunikovat mobilní aplikace v klientské části.

4.6.1 Použité nástroje

K vývoji bylo použito vývojové prostředí IntelliJ IDEA od firmy JetBrains. Ke kompilaci a spuštění projektu byl nainstalován Software Development Kit pro programovací jazyk Java 8. Dotazy na webové API byly prováděny programem Postman.

4.6.2 Vytvoření projektu

Projekt aplikace byl vytvořen pomocí webové aplikace dostupné na adrese https://start.spring.io/. Tato aplikace dává na výběr verzi Javy a frameworku Spring včetně jeho modulů. Byla vybrána Java ve verzi 8 a framework Spring ve verzi 2.1.0. Následně byl stažen archiv ve formátu zip s vygenerovaným projektem. Součástí rozbaleného archivu je i nástroj Maven, který slouží k buildu celého projektu.



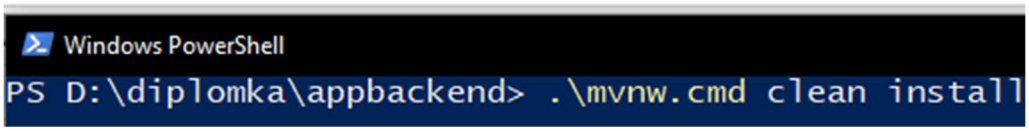
Obrázek 20: Obsah kořenové složky projektu, zdroj: autor

XML soubor pom.xml ve složce s projektem obsahuje informace o používaných knihovnách a popisuje, jak se má projekt sestavit. Soubor mvnw.cmd umožňuje spustit build projektu. Složka src obsahuje zdrojové soubory v programovacím jazyku Java.

4.6.3 Build a spuštění serverové aplikace

Předpokladem k buildu aplikace je nainstalování Software Development Kitu (SDK) pro Javu a následné nastavení systémové proměnné JAVA_HOME s cestou, kam bylo SDK nainstalováno.

K buildu aplikace pak stačí pomocí příkazové řádky spustit přiložený program mvnw.cmd s argumenty „clean install“:



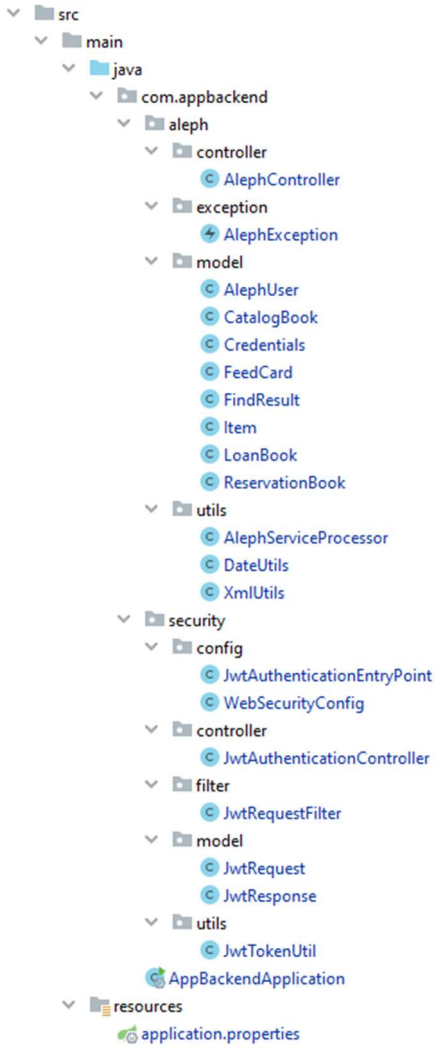
Obrázek 21: příkaz k buildu projektu, zdroj: autor

Po úspěšném buildu dojde k vytvoření složky target, ve které se bude nacházet soubor s příponou jar. Tento soubor se spouští programem java.exe s argumentem „-jar“:

Obrázek 22: Spuštění serverové aplikace

4.6.4 Struktura projektu

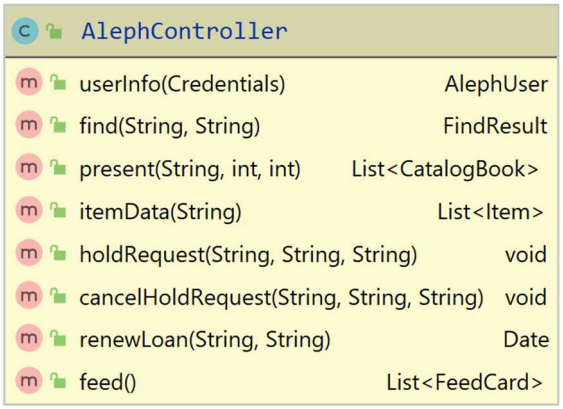
Složka src obsahuje Java balíček com.appbackend, který se skládá ze dvou podbalíčků: aleph a security. Balíček aleph obsahuje zdrojové soubory týkající webového API pro komunikaci se systémem Aleph. V balíčku security jsou zdrojové soubory týkající se zabezpečení aplikace. Soubor application.properties ve složce resources obsahuje nastavení aplikace.



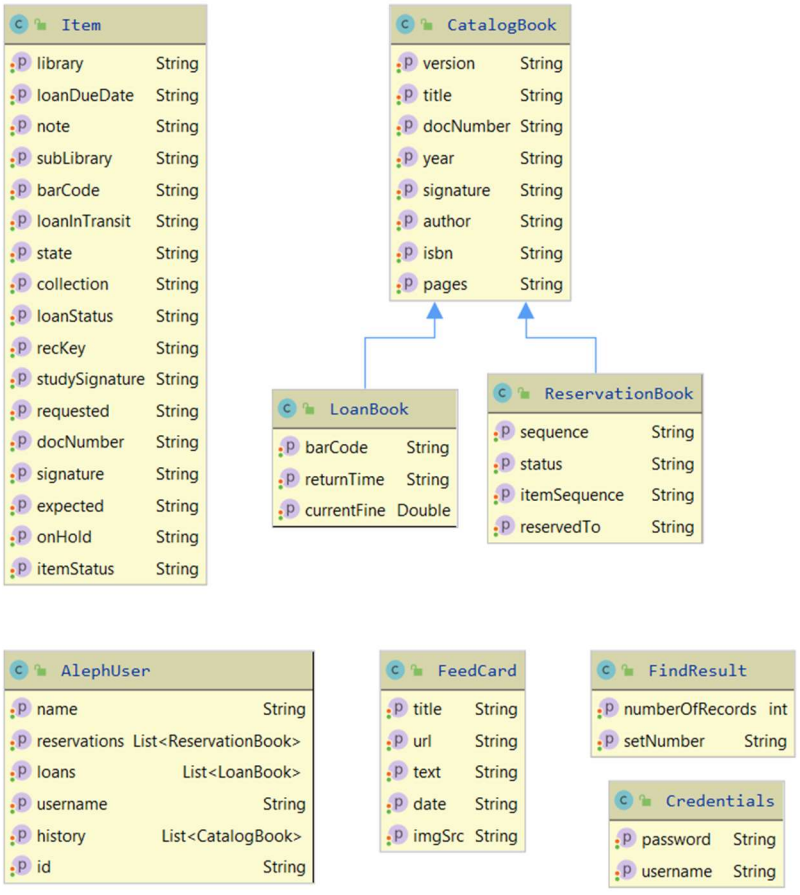
Obrázek 23: Struktura projektu, zdroj: autor

4.6.5 Tvorba webového API – AlephController

Webové API bylo vytvořeno pomocí programovacího jazyka Java a frameworku Spring. Rozhrání API bylo vytvořeno Spring anotací Controller nad třídou AlephController. Metody rozhrání byly definovány anotací RequestMapping. Každá metoda prezentuje webovou službu. Název metody určuje koncové URL, pod kterým je možné vzdáleně provolat webovou službu HTTP dotazem.



Obrázek 24: Metody webového API, zdroj: autor



Obrázek 25: Model webových služeb, zdroj: autor

4.6.6 Komunikace se systémem Aleph – AlephServiceProcessor

Téměř každá metoda ve třídě AlephController komunikuje se systémem Aleph prostřednictvím komponenty Aleph X-Server. Tuto komunikaci obstarává třída AlephServiceProcessor a její metoda processAlephRequest(). Argumentem této metody je textový řetězec s parametry pro systém Aleph. Metoda načte z nastavení aplikace definovanou URL pro server Alephu (včetně přihlašovacích údajů) a přidá k ní parametry. Následně provede dotaz na Aleph server a vrátí transformovanou odpověď z formátu XML v instanci třídy Document. Tato instance je následně zpracována pomocí XPathu a statických metod ve třídě XmlUtils dle potřeb dané metody.

Součástí každé odpovědi ze systému Aleph je identifikátor session, ten je možné v dalších dotazech použít místo přihlašovacích údajů. Session je uložena do statické proměnné s názvem CACHED_SESSION_ID. Dále je uložena informace s datem vytvoření session do statické proměnné SESSION_CACHED_TIME, pomocí její hodnoty je kontrolováno obnovování CACHED_SESSION_ID každých 15 minut.

Pokud bude odpověď ze systému Aleph obsahovat error, dojde k vyhození výjimky AlephException, která vyvolá vrácení stavového kódu 404 v HTTP odpovědi.

4.6.7 Přihlášení a načtení uživatelských dat – userInfo()

Pro přihlášení a načtení uživatelských dat ze systému Aleph byla vystavena metoda userInfo(). Tuto metodu je možné provolat HTTP dotazem metodou POST. Tělo dotazu musí obsahovat přihlašovací údaje do systému Aleph ve formátu JSON. Přihlašovací údaje jsou následně přeposlány do webové služby bor-info systému Aleph. Služba Alephu vrátí XML, ze kterého jsou následně načteny jednotlivé vlastnosti pro instanci třídy AlephUser. Výstupem metody je JSON obsahující informace o výpůjčkách, rezervacích a dříve vypůjčených knihách. Pokud systém Aleph vrátí chybu, bude vyhozena výjimka a výstupem bude JSON obsahující chybovou hlášku a stavový kód 404.

Ukázka volání metody userInfo pomocí HTTP dotazu a metody POST:

```
POST /aleph/userInfo HTTP/1.1
Host: localhost:9090
Content-Type: application/json

{
      "username":"xilep001",
      "password":"-------"
}
```

Odpověď při úspěšném načtení uživatelských údajů:


```
{
    "username": "xilep001",
    "id": "ID64185",
    "name": "Ilek Petr",
    "loans": [],
    "reservations": [
        {
            "docNumber": "000062761",
            "title": "Ekonometrické modelování",
            "author": "Hančlová, Jana,",
            "year": "2012",
            "isbn": "978-80-7431-088-1 (brož.)",
            "signature": "Z 25943",
            "pages": "",
            "version": null,
            "itemSequence": "000010",
            "sequence": "0001",
            "status": "Waiting in queue",
            "reservedTo": "02/04/2021"
        }
    ],
    "history": []
}
```


Odpověď při neúspěšném načtení uživatelských údajů:


```
{
    "timestamp": "2020-04-01T17:18:58.426+0000",
    "status": 404,
    "error": "Not Found",
    "message": "Error retrieving Patron System Key",
    "path": "/aleph/userInfo"
}
```


4.6.8 Vyhledávání knih v katalogu – find()

Vyhledávání knih v katalogu Aleph zajišťují dvě metody. První s názvem find() na základě hledaného výrazu a filtru vrací metadata o nalezených záznamech. Vyhledávání je možné filtrovat podle hodnot parametru filter:

- title – vyhledávání podle názvu.
- author – vyhledávání podle autora.
- signature – vyhledávání podle signatury.

Návratovou hodnotou metody je instance třídy FindResult, která obsahuje:

- Identifikátor – setNumber.
- počet nalezených záznamů – numberOfRecords.

V případě nenalezení knih bude výstupem služby JSON obsahující chybovou hlášku a stavový kód 404.

Tuto metodu je možné provolat HTTP dotazem s metodou GET:

```
GET /aleph/find?query=matematika&filter=title HTTP/1.1
Host: localhost:9090
```

Odpověď při úspěšném nalezení knih:

```
{
    "setNumber": "000676",
    "numberOfRecords": 154
}
```

Odpověď při neúspěšném nalezení knih:

```
{
    "timestamp": "2020-04-01T20:09:20.729+0000",
    "status": 404,
    "error": "Not Found",
    "message": "empty set",
    "path": "/aleph/find"
}
```

Se získaným identifikátorem setNumber je možné provolat metodou GET druhou vystavenou metodu s názvem present(). Vstupními parametry jsou **setNumber** a číselné hodnoty **from** a **loadBy**. Tyto hodnoty specifikují, od jakého záznamu a po kolika záznamech mají být vrácené údaje o nalezených knihách. Výstupem je tedy pole s instancemi třídy CatalogBook ve formátu JSON.

Ukázka volání metody HTTP dotazem metodou GET:

```
GET /aleph/present?setNumber=000676&from=1&loadBy=10 HTTP/1.1
Host: localhost:9090
```

Odpověď při úspěšném nalezení knih:

```
[
    {
        "docNumber": "74058",
        "title": "Nová infinitní matematika.",
        "author": "Petr Vopěnka",
        "year": "2016",
        "isbn": "9788024632216",
        "signature": null,
        "pages": "1",
        "version": ""
    },
    ...
]
```

4.6.9 Načtení exemplářů knihy – itemData()

Načtení jednotlivých exemplářů knihy pomocí vstupního parametru docNumber zastřešuje metoda itemData().

Tuto službu je možné provolat HTTP dotazem metodou GET:

```
GET /aleph/itemData?docNumber=74056 HTTP/1.1
Host: localhost:9090
```

Odpověď obsahuje pole instancí třídy Item ve formátu JSON:


```
[
    {
        "docNumber": "74056",
        "recKey": "000074056000010",
        "barCode": "EZ000001142",
        "signature": "",
        "studySignature": "",
        "subLibrary": "ČZU",
        "collection": "Základní knihovna SIC",
        "itemStatus": "Online",
        "note": "3 users",
        "library": "CZU50",
        "onHold": "N",
        "requested": "N",
        "expected": "N",
        "loanStatus": "",
        "loanInTransit": "",
        "loanDueDate": null,
        "state": "K vypůjčení online"
    }
]
```


4.6.10 Rezervace výpůjčky – holdRequest()

Rezervaci momentálně vypůjčeného exempláře lze provést provoláním metody holdRequest() s parametry docNumber, itemBarCode a borId.

Ukázka volání HTTP dotazu metodou GET:

```
GET /aleph/holdRequest?docNumber=62761&itemBarCode=13/1273&bor_id=ID64185
HTTP/1.1
Host: localhost:9090
```

Při úspěšném provedení rezervace je vrácen stavový kód 200, v opačném případě je vrácen stavový kód 404 s chybovou hláškou.

4.6.11 Zrušení rezervace výpůjčky – cancelHoldRequest()

Ke zrušení rezervace výpůjčky prostřednictvím metody cancelHoldRequest() jsou potřeba údaje o rezervaci itemSequence a sequence, které vrací webová metoda userInfo(). Dále je potřeba také docNumber.

Ukázka volání HTTP dotazu metodou GET:

```
GET
/aleph/cancelHoldRequest?docNumber=000062761&itemSequence=000010&sequence=0001
HTTP/1.1
Host: localhost:9090
```

Při úspěšném zrušení rezervace je vrácen stavový kód 200, v opačném případě je vrácen stavový kód 404 s chybovou hláškou.

4.6.12 Prodloužení výpůjčky – renewLoan()

Metoda pro prodloužení výpůjčky renewLoan vyžaduje itemBarCode výpůjčky a id uživatele borId. Při úspěšném prodloužení výpůjčky je vráceno datum prodloužení ve formátu JSON, pokud nastane chyba, aplikace vrátí stavový kód 404.

Ukázka volání HTTP dotazu metodou GET:

```
GET /aleph/renewLoan?itemBarCode=13/1274&borId=ID64185 HTTP/1.1
Host: localhost:9090
```

Vrácené datum ve formátu JSON při úspěšném prodloužení výpůjčky:

```
"2019-12-02T23:00:00.000+0000"
```

4.6.13 Zobrazení aktualit – feed()

Data pro zobrazení aktualit zajišťuje metoda feed. Tato metoda přebírá metadata o aktualitách z webu univerzitní knihovny (https://lib.czu.cz/cs/r-8995-aktuality-sic) a transformuje je do pole instancí třídy FeedCard ve formátu JSON.

Ukázka volání HTTP dotazu metodou GET:

```
GET /aleph/feed HTTP/1.1
Host: localhost:9090
```

Vrácená metadata o aktualitách:

```
[
    {
      "title": "Ztráty a nálezy",
      "date": "20.1.2020",
      "text": "Nemůžeš něco najít? Nezoufej!",
      "imgSrc": "https://lib.czu.cz/cache/content-files/82325-552x304.jpg",
      "url": "https://lib.czu.cz/cs/r-8995-aktuality-sic/ztraty-a-nalezy.html"
    },
    ...
]
```

4.6.14 Zabezpečení webového API – JwtAuthenticationController

Všechny metody webového API definované ve třídě AlephController jsou zabezpečeny JSON Web Tokeny (standard RFC 7519) pomocí frameworku Spring Security. Každý dotaz do aplikace tak musí v hlavičce obsahovat JSON Web Token, který je verifikován v metodě doFilterInternal() ve třídě JwtRequestFilter.

K získání tokenu byla vystavena veřejná metoda authenticate() ve třídě JwtAuthenticationController. Parametrem této metody jsou přihlašovací údaje ve formátu JSON. Přihlašovací údaje jsou následně přeposlány do webové služby bor-auth systému Aleph k autentizaci.

Pokud je autentizace úspěšná aplikace vrátí vygenerovaný token, v opačném případě vrátí stavový kód 401. Generování tokenu obstarává třída JwtTokenUtil a její metoda doGenerateToken(), která na základě uživatelského jména pomocí knihovny JWT vygeneruje token. Token je vytvořen algoritmem HS512 s definovaným klíčem a expirací v nastavení aplikace.

Ukázka volání metody authenticate() pomocí HTTP dotazu a metody POST:

```
POST /authenticate HTTP/1.1
Host: localhost:9090
Content-Type: application/json
Content-Type: application/json

{
      "username":"xilep001",
      "password":"-------"
}
```

Odpověď při úspěšné autentizaci:

```
{
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ4aWxlcDAwMSIsImV4cCI6MTU4NTc2MzkyNCwiaWF0IjoxNTg1NzQ1OTI0fQ.gufRbOCcG_DFwjEvRJ2xPQigCs0duAnV7oQOtDJBBXMaWg2Wscs3XmSJtGSknsh1qBKuZDq5zLlSa9yBKOSAzA"
}
```


Odpověď při neúspěšné autentizaci:


```
{
    "timestamp": "2020-04-01T14:43:58.843+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "Unauthorized",
    "path": "/authenticate"
}
```


4.6.15 Konfigurace

Aplikaci je možné konfigurovat editací souboru application.properties pomocí následujících proměnných:

- server.port - určuje port na kterém bude aplikace dostupná.
- jwt.secret – klíč pro šifrování JSON Web Tokenu (JWT).
- jwt.expirationHours – doba za jak dlouho token vydrží (v hodinách).
- aleph.url – adresa knihovního systému Aleph.
- aleph.and.separator – určuje jaký oddělovač bude použit mezi parametry při volání webových služeb Aleph.

```
server.port=9090

jwt.secret=eBfsBXfN6rQAMs
jwt.expirationHours=5

aleph.url=https://aleph.czu.cz
aleph.and.separator=&
```

4.7 Implementace mobilní aplikace

Tato kapitola popisuje jednotlivé kroky implementace mobilní aplikace.

4.7.1 Použité nástroje

K vývoji bylo použito vývojové prostředí IntelliJ IDEA od firmy JetBrains. Mobilní aplikace byla naprogramována v programovacím jazyku TypeScript s použitím frameworku React Native. Projekt dále používá nástroj Node.js, správce balíčků Node Package Manager (NPM) a nástroj Expo.

4.7.2 Vytvoření projektu

Nejprve byla stažena a nainstalována platforma Node.js. Součástí Node.js je i správce balíčků NPM. Pomocí správce NPM byl přes příkazovou řádku nainstalován nástroj Expo:

```
npm install -g expo-cli
```

Obrázek 26: Instalace nástroje Expo pomocí NPM, zdroj: autor

Následně byl založen projekt s názvem „sic-app“ příkazem „expo-init“ pro nástroj Expo. Při založení projektu byla vybrána prázdná šablona pro programovací jazyk TypeScript:

```
PS C:\Users\petri\Desktop\expo> expo init SicApp
┌─────────────────────────────────────────────────────────────────────────────┐
│                                                                             │
│   There is a new version of expo-cli available (3.17.15).                   │
│                                                                             │
│   You are currently using expo-cli 3.11.7                                   │
│                                                                             │
│   Install expo-cli globally using the package manager of your choice; for example latest version   │
│                                                                             │
└─────────────────────────────────────────────────────────────────────────────┘
? Choose a template:
  ----- Managed workflow -----
  blank                 a minimal app as clean as an empty canvas
> blank (TypeScript)    same as blank but with TypeScript configuration
  tabs                  several example screens and tabs using react-navigation
  ----- Bare workflow -----
  minimal               bare and minimal, just the essentials to get you started
  minimal (TypeScript)  same as minimal but with TypeScript configuration
```

Obrázek 27: Založení projektu nástrojem Expo, zdroj: autor

4.7.3 Build a spuštění mobilní aplikace

Nejprve byl vytvořen uživatelský účet na stránkách: https://expo.io/. Pod tímto účtem je nutné se poté přihlásit do nástroje Expo příkazem „expo login“.

```
expo login -u [username] -p [password]
```

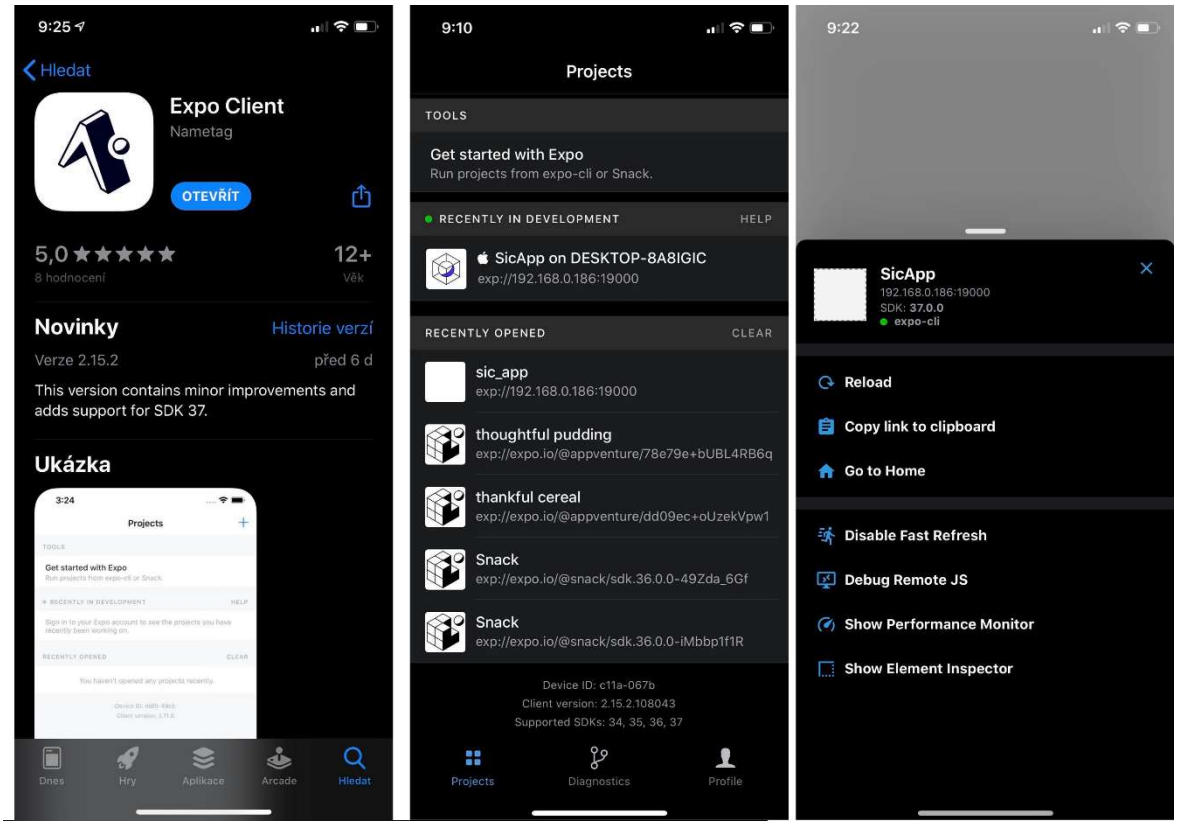
Obrázek 28: Přihlášení do nástroje Expo, zdroj: autor

Build aplikace se provádí příkazem „npm start“, který vyvolá spuštění nástroje Expo v na adrese: http://localhost:19002/.

```
npm start
```

Obrázek 29: Build aplikace, zdroj: autor

Ke spuštění aplikace v režimu ladění bylo potřeba stáhnout do mobilního zařízení aplikaci Expo Client přes Obchod Play nebo App Store a přihlásit se do ní pod svým účtem. Po přihlášení se objeví v seznamu projektů odkaz, který vyvolá spuštění aplikace. Po spuštění je možné zatřesením zařízení otevřít nabídku s možnostmi ladění.



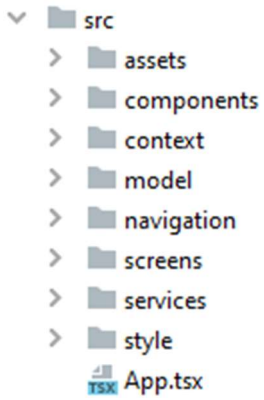
Obrázek 30: Mobilní aplikace pro nástroj Expo, zdroj: autor

4.7.4 Struktura projektu

Nejdůležitější položky v kořenové složce projektu:

- package.json – soubor popisuje závislosti projektu a jeho sestavení.
- app.json – soubor s nastavením aplikace: název aplikace, logo atd..
- node_modules – složka s nainstalovanými závislostmi.

- src – složka se zdrojovými kódy v programovacím jazyku TypeScript.



Obrázek 31: Obsah složky src, zdroj: autor

Obsah složky src:

- assets – složka s ikonami a fonty.
- components – složka se zdrojovým kódem komponent, které jsou univerzální.
- context – složka pro soubory definující React Context.
- model – složka s definicí tříd s modelem pro webové API.
- navigation – složka pro třídy vykonávající navigaci v aplikaci.
- screens – složka s logikou jednotlivých obrazovek aplikace.
- services – složka pro aplikační logiku.
- style – složka pro globální styly komponent.
- App.tsx – soubor se zdrojový kódem, který se vykoná při spuštění aplikace.

4.7.5 Implementace navigace v aplikaci – App.tsx

V souboru App.tsx byla deklarována funkcionální komponenta s názvem App. Tato komponenta byla nastavena jako vstupní bod do aplikace funkcí registerRootComponent().

Pomocí knihovny React Navigation a funkce createBottomTabNavigator() byla vytvořena konstanta Tab. Obsahem této konstanty jsou komponenty Navigator a Screen přístupné přes tečkovou notaci. Použitím těchto komponent byla složena spodní navigace:

Obrázek 32: Výsledná navigace, zdroj: autor

V Komponentě App byl definován State s proměnnými userSigned a isLoading, které reprezentují, zda je uživatel přihlášen, nebo zda probíhá načítání aplikace. Jejich hodnoty jsou měněny funkcí reducer() na základě definovaných akcí:

- LOADING_FINISHED.
- SIGN_IN.
- USER_ALREADY_SIGNED.
- SIGN_OUT.

Tyto akce mění zobrazení jednotlivých obrazovek v aplikaci.

- Akce LOADING_FINISHED indikuje, že proběhlo úspěšné načtení aplikace a může tedy být zobrazena přihlašovací obrazovka.
- Akce SIGN_IN nebo USER_ALREADY_SIGNED říkají, že uživatel byl přihlášen a má se zobrazit domovská obrazovka.
- Akce SIGN_OUT reprezentuje odhlášení uživatele a provede zobrazení přihlašovací obrazovky.

V komponentě App byl definován React Context na základě rozhrání AuthContextInterface. Tento kontext je dostupný v celé aplikace a umožňuje uživatele přihlásit nebo odhlásit metodami signIn() nebo signOut().

4.7.6 **Přihlašovací obrazovka – LoginScreen.tsx**

Při tvorbě přihlašovací obrazovky bylo nejprve provedeno vytvoření textových polí pro přihlašovací jméno a heslo komponentami Input. Tyto textové pole mají definovaný State použitím React Hooku useState(). Tlačítko pro přihlášení vykreslené komponentou Button reaguje na událost onPress provoláním funkce submit(). Spuštěním této funkce jsou hodnoty ze State textových polí poslány do webového API vykonáním metody getUserInfo(). Pokud server nevrátí error, dojde k přihlášení uživatele metodou signIn() předanou pomocí React Contextu. V metodě signIn() je na pozadí ukládán serializovaný objekt třídy User a JWT token z webového API v zašifrované podobě do paměti zařízení.

4.7.7 **Volání webového API – AlephService.ts**

Logika pro volání webového API byla vytvořena ve třídě AlephService. V konstantě API_URL byla nastavena adresa, kde běží serverová aplikace. Každé volání webového API je provedeno metodou processApiCall(), která obstarává přidávání uloženého JWT tokenu

do hlavičky každého volání. Pokud je JWT token prošlý, provolá znovu metodu web API k získání nového a následně jej uloží do zašifrované paměti v zařízení. Ostatní metody třídy AlephService korespondují s názvy metod webového API, jejich úkolem je předávání dat z webového API do prezentační vrstvy.

4.7.8 Domovská obrazovka – HomeScreen.tsx

K získání informací o stavu čtenářského konta bylo potřeba nejprve provolat webové API statickou metodou AlephService.getUserInfo(). Její návratový objekt třídy User byl následně uložen do State funkcionální komponenty HomeScreen. Odkazy na obrazovky výpůjčky, rezervace a historie výpůjček byly vytvořeny komponentou TouchableHighlight. Stylování bylo provedeno Flexboxem.

4.7.9 Karta knihy – BookListItem.tsx

Protože z drátěného modelu vyplývalo časté zobrazování karty s knihou, byla vytvořena univerzální komponenta BookListItem. Do Props této komponenty je možné předat údaje o knize a funkci onPress(), která se vykoná po kliknutí na kartu knihy. Načítání obrázku knihy bylo provedeno provoláním webové služby obalkyknih.cz, která na základě ISBN knihy vrátí adresu s obrázkem.

4.7.10 Obrazovky výpůjčky, rezervace a historie výpůjček

Obrazovky výpůjčky, rezervace a historie výpůjček byly vytvořeny jako funkcionální komponenty v souborech:

- LoansScreen.tsx.
- ReservationsScreen.tsx.
- HistoryScreen.tsx.

Seznam knih v těchto obrazovkách byl vytvořen pomocí komponenty pro tvorbu seznamů FlatList. Do parametru renderItem této komponenty byla předána funkce vykreslující komponentu BookListItem. Tlačítka „Rezervovat“ a „Zrušit rezervaci“ byly do komponenty BookListItem předány pomocí kompozice. Po jejich stisknutí dojde k provolání příslušného webového API statickými metodami ve třídě AlephService.

4.7.11 **Katalog knih – AlephCatalogScreen.tsx**

Nejprve bylo provedeno vytvoření vyhledávacího pole a přepínač filtrů. K tomu byly použity komponenty Input a ButtonGroup. Těmto komponentám byl přiřazen State, při jehož změně dojte k aktualizaci seznamu knih na základě provolání metody find() vzdáleného API.

Metoda find() vrací objekt FindResult s počtem nalezených knih (numberOfRecords) a identifikátorem výsledku daného hledání (setNumber). Tento objekt je uložen do třídní proměnné findResult. Následně jsou dotaženy údaje o jednotlivých knihách voláním metody present(). Tato metoda byla parametrizována tak, aby vracela výsledky po dvanácti knihách.

Seznam knih byl vytvořen stejně jako v předchozí kapitole komponentami Flatlist a BookListItem. V komponentě FlatList byla přidaná reakce na událost onEndReached voláním metody loadMoreBooks(). Tato metoda je zavolána při dosažení konce seznamu knih v případě, že nejsou načtené všechny knihy. Uvnitř této metody dojde znovu k volání metody present(), která vrátí údaje pro dalších dvanáct knih. Tímto způsobem jsou načteny všechny knihy dokud není zobrazený počet knih roven hodnotě proměnné numberOfRecords.

4.7.12 **Detail knihy a exempláře – BookDetailScreen.tsx**

Detail knihy zastřešuje funkcionální komponenta BookDetailScreen, do Props jsou předány údaje o knize, které jsou následně zobrazeny. Načítání obrázku knihy bylo implementováno stejně jako na kartě knihy.

4.7.13 **Exempláře – BookItemsScreen.tsx**

Při prvním vykreslení komponenty BookItemsScreen je zobrazena indikace načítání komponentou ActivityIndicator. Data pro zobrazení exemplářů knihy jsou načítána na základě hodnoty docNumber. K vyvolání tohoto načítání byl použit hook useEffect(), ve kterém dojde k asynchronnímu provolání funkce loadItems(). Tato funkce načte data z webového API metodou getItemData() a následně aktualizuje State komponenty, což způsobí zobrazení exemplářů v komponentě FlatList a schování indikace načítání. Pokud má exemplář reprezentovaný instancí třídy Item neprázdný atribut loanDueDate, bude vykresleno tlačítko „Rezervovat“. Interakce s uživatelem po stisknutí tlačítka byla implementována prostřednictvím statické metody Alert.alert().

4.7.14 **Aktuality – FeedScreen.tsx**

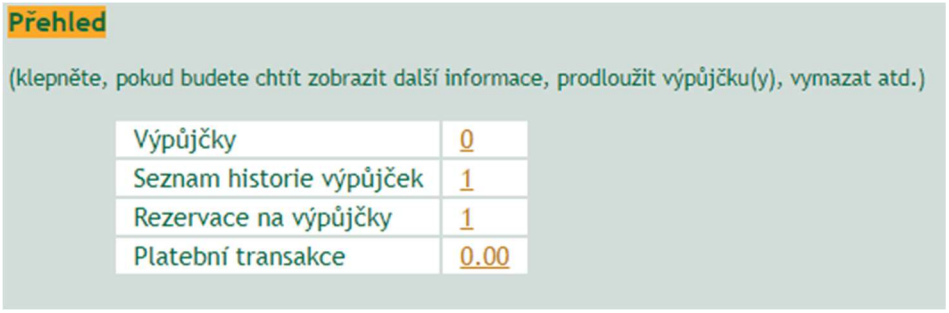
Načítání aktualit v komponentě FeedScreen bylo provedeno podobnou logikou jako u komponenty BookItemsScreen. Otevírání internetového prohlížeče po stisknutí tlačítka „Číst dále“ bylo definováno ve funkci openInBrowser(). Která na základě URL aktuality zobrazí okno prohlížeče statickou metodou WebBrowser.openBrowserAsync().

4.7.15 **Informace o knihovně - InfoScreen.tsx**

Tato komponenta zobrazuje statický text s informacemi o knihovně pomocí komponent RobotoText. Jednotlivé sekce byly odděleny použitím komponenty Divider. Mapa s lokací univerzitní knihovny je zobrazena komponentou MapView na základě GPS souřadnic předaných v Props.

4.8 Testování

Nejprve bylo provedeno bylo provedeno testování serverové části v aplikaci Postman. Jednotlivé metody webového API byly provolávány a jejich výstup byl porovnán s výstupem webového rozhrání katalogu univerzitní knihovny na adrese: https://aleph.czu.cz/.



Obrázek 33: Čtenářské konto v katalogu Aleph – web, zdroj: autor

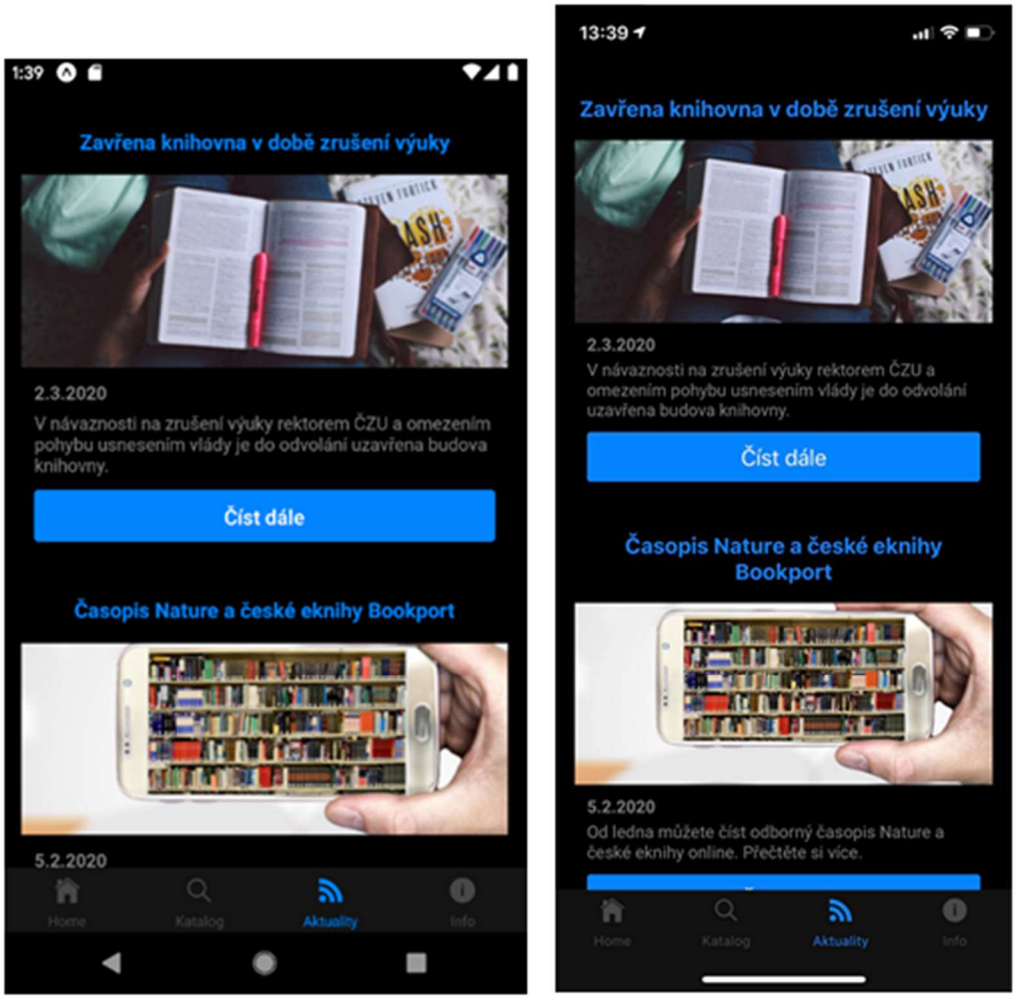
Při testování občas docházelo k chybě na straně Aleph X-serveru, kdy při odeslání správných uživatelských údajů vracel chybu: „Error retrieving Patron System Key“. Po vyskočení této chyby se nebylo možné ani přihlásit do čtenářského konta na webu univerzitní knihovny. Tento jev byl opraven ukládáním sessionId z Aleph X-serveru do dočasné paměti aplikace.

Poslední chyba, která byla nalezena, je vkládání různých oddělovačů do atributů třídy reprezentující knihu. Tato chyba byla opravena nahrazováním znaků z množiny {\;,,:} prázdným znakem.

```
{
    "username": "xilep001",
    "id": "ID64185",
    "name": "Ilek Petr",
    "loans": [],
    "reservations": [
        {
            "docNumber": "000070035",
            "title": "Kapesní fyzika pro inženýry ",
            "author": "Blahovec, Jiří,",
            "year": "2016",
            "isbn": "978-80-213-2675-0 (brožováno)",
            "signature": "Z 28080/1",
            "pages": "",
            "version": null,
            "itemSequence": "000020",
            "sequence": "0004",
            "status": "Waiting in queue",
            "reservedTo": "05/04/2021"
        }
    ],
    "history": [
        {
            "docNumber": "",
            "title": "Statistické metody I ",
            "author": "Svatošová, Libuše",
            "year": "2007",
            "isbn": "978-80-213-1672-0 (brož.) : Kč 75,00",
            "signature": "S 2625/251",
            "pages": "",
            "version": null
        }
    ]
}
```


Obrázek 34: Výstup z provolání webového API, zdroj: autor

Nakonec bylo provedeno testování průchodu mobilní aplikací na zařízeních Nokia 6.1 a Apple iPhone Xs. Na obou zařízeních se aplikace zobrazovala a chovala korektně.



Obrázek 35: Zobrazení aktualit, vlevo Nokia 6.1, vpravo iPhone Xs, zdroj: autor

5 Závěr

V teoretické části práce byla přiblížena problematika vývoje mobilních aplikací. Velká pozornost byla věnována multiplatformnímu vývoji aplikací pomocí technologie React Native. Byly představeny klíčové koncepty této technologie včetně praktických ukázek. Konec teoretické části byl věnován technologii Spring a univerzitnímu knihovnímu systému Aleph.

V praktické části byl na základě analýzy požadavků v kapitole 4.1 byl proveden návrh aplikace pomocí drátěných modelů (wireframů). Před samotnou implementací aplikace byl proveden návrh infrastruktury, ve kterém byla navržena serverová aplikace sloužící jako prostředník mezi mobilními zařízeními a knihovním systémem Aleph.

K implementaci serverové aplikace byl použit programovací jazyk Java a framework Spring. Tyto technologie se osvědčily svou robustností a kvalitní dokumentací. Výsledkem je aplikace publikující webové API, které bylo zabezpečeno standardem RFC 7519.

Implementace mobilní aplikace byla provedena použitím JavaScriptového frameworku React Native. Aplikace byla stylizována do tmavého tématu, které je aktuálně velmi populární pro úsporu baterie mobilních zařízení. Obrazovky finální aplikace jsou obsaženy v přílohách. Zkušenosti s technologií React Native nabývaly velmi pozitivního dojmu, až na občasné pády aplikace bez vypsání chybové hlášky.

Výstupem je aplikace pro platformy Android a iOS, která si klade za cíl usnadnit uživatelům knihovny přístup ke čtenářskému účtu, katalogu, aktualitám a informacím knihovny.

5.1 Budoucí vývoj

Aplikace je nyní připravena k otestování více uživateli, k tomu je potřeba zajistit server s veřejnou IP adresou, na kterém poběží serverová část aplikace. Tato IP adresa bude nastavena v projektu s mobilní aplikací. Následně bude možné vydat testovací verzi aplikace do služeb Google Play a TestFlight pomocí nástroje Expo.

Náměty na novou funkcionalitu:

- Přidání světlého vzhledu aplikace, který bude možné přepínat.
- Lokalizace aplikace do anglického jazyka.
- Odesílání push notifikací o stavu výpůjček a rezervací.

6 Seznam použitých zdrojů

[1] SCHÖN, Otakar. Před deseti lety začala revoluce. iPhone změnil svět mobilů, teď ale sám potřebuje změnu. *Aktuálně.cz* [online]. Praha: Economia a.s., 2017 [cit. 2020-03-02]. Dostupné z: https://zpravy.aktualne.cz/ekonomika/pred-deseti-lety-zacala-revoluce-iphone-zmenil-tvar-mobilu-t/r~6058e2445d7511e7954a002590604f2e/

[2] CLEMENT, J. Number of apps available in leading app stores as of 4th quarter 2019. *Statista* [online]. New York, 2020 [cit. 2020-1-22]. Dostupné z: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[3] Mobile Application (Mobile App): Definition - What does Mobile Application (Mobile App) mean?. *Techtopedia* [online]. 2018 [cit. 2020-01-02]. Dostupné z: https://www.techopedia.com/definition/2953/mobile-application-mobile-app

[4] GLOAG, David. Mobile App: Definition, Development & Management. *Study.com* [online]. Mountain View, c2003-2020 [cit. 2020-01-02]. Dostupné z: https://study.com/academy/lesson/mobile-app-definition-development-management.html

[5] FLORA, Harleen, Xiaofeng WANG a Swati CHANDE. An Investigation on the Characteristics of Mobile Applications: A Survey Study. *International Journal of Information Technology and Computer Science* [online]. 2014, **6**(11), 21-27 [cit. 2020-01-22]. DOI: 10.5815/ijitcs.2014.11.03. ISSN 20749007. Dostupné z: http://www.mecs-press.org/ijitcs/ijitcs-v6-n11/v6n11-3.html

[6] GREGORY, Samuel. What is the Difference Between Web Apps, Native Apps, Hybrid Apps and Progressive Web Apps?. *Hacker Noon* [online]. 2019 [cit. 2020-22-01]. Dostupné z: https://hackernoon.com/what-is-the-difference-between-web-apps-native-apps-hybrid-apps-and-progressive-web-apps-py19n2gdi

[7] COWART, Jim. What is a Hybrid Mobile App?. *Telerik* [online]. Bedford, 2012 [cit. 2020-23-01]. Dostupné z: https://www.telerik.com/blogs/what-is-a-hybrid-mobile-app-

[8] Hybrid mobile app. *PCMag* [online]. c1996-2020 [cit. 2020-23-01]. Dostupné z: https://www.pcmag.com/encyclopedia/term/hybrid-mobile-app

[9] Introduction to progressive web apps. *MDN web docs* [online]. Mozilla, 2019 [cit. 2020-25-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction

[10] WINDMILL, Eric. *Exploring Cross-Platform Development with Flutter, React Native, and Xamarin*. New York: O'Reilly Media, 2019. ISBN 9781617296789.

[11] ZAMMETTI, Frank. *Practical React Native: Build Two Full Projects and One Full Game using React Native*. Pottstown: Apress, 2018. ISBN 978-1-4842-3939-1.

[12] EISENMAN, Bonnie. *Learning React Native: Building native mobile apps with JavaScript*. Second Edition. New York: O'Reilly Media, 2017. ISBN 978-1-491-98914-2.

[13] What is Web API?. *Tutorials Teacher* [online]. Tutorials Teacher, 2020 [cit. 2020-03-25]. Dostupné z: https://www.tutorialsteacher.com/webapi/what-is-web-api

[14] HAVERBEKE, Marijn. *Eloquent JavaScript: a modern introduction to programming*. Third edition. San Francisco: No Starch Press, 2019. ISBN 978-1-59327-950-9.

[15] FAIN, Yakov a Anton MOISEEV. *TypeScript Quickly*. 2nd. New York: Manning Publications Company, 2020. ISBN 9781617295942.

[16] JavaScript. *MDN web docs* [online]. Mountain View: Mozilla, 2020 [cit. 2020-02-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[17] History of Ecma. *History of Ecma* [online]. Geneva: Ecma International [cit. 2020-02-04]. Dostupné z: https://www.ecma-international.org/memento/history.htm

[18] NANCE, Jared. TypeScript vs. JavaScript Migrate. *Stackify* [online]. [cit. 2019-11-25]. Dostupné z: https://stackify.com/typescript-vs-javascript-migrate/

[19] DOM: Document Object Model. *Tvorba webu* [online]. Praha: Adaptic [cit. 2020-03-09]. Dostupné z: https://www.tvorba-webu.cz/dom/

[20] EYGI, Cem. What is "Props" and how to use it in React?. *ITNEXT* [online]. Medium, 2019 [cit. 2020-02-04]. Dostupné z: https://itnext.io/what-is-props-and-how-to-use-it-in-react-da307f500da0

[21] Component State in React Native. *GeeksForGeeks: A computer science portal for geeks* [online]. GeeksForGeeks, 2018 [cit. 2020-02-05]. Dostupné z: https://www.geeksforgeeks.org/component-state-react-native/

[22] React.Component: The Component Lifecycle. *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-02-05]. Dostupné z: https://reactjs.org/docs/react-component.html

[23] Style. *React Native: A framework for building native apps using React* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-02-10]. Dostupné z: https://reactnative.dev/docs/style

[24] Composition vs Inheritance. *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-02-06]. Dostupné z: https://reactjs.org/docs/composition-vs-inheritance.html

[25] Refs and the DOM. *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-03-10]. Dostupné z: https://reactjs.org/docs/refs-and-the-dom.html

[26] ZIROLL, Bob. Learn React Context in 5 Minutes: A Beginner's Tutorial. *FreeCodeCamp* [online]. San Francisco: Free Code Camp, 2019 [cit. 2020-03-10]. Dostupné z: https://www.freecodecamp.org/news/react-context-in-5-minutes/

[27] Context. *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-03-10]. Dostupné z: https://reactjs.org/docs/context.html

[28] Introducing Hooks. *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-02-15]. Dostupné z: https://reactjs.org/docs/hooks-intro.html

[29] BASRA, Manisha. React Hooks: useState (using the state hook). *Hacker Noon* [online]. Eagle: Hacker Noon, 2019 [cit. 2020-03-10]. Dostupné z: https://hackernoon.com/react-hooks-usestate-using-the-state-hook-89ec55b84f8c

[30] Using the State Hook. *React: A JavaScript library for building user interfaces* [online]. Menlo Park: Facebook Inc., 2020 [cit. 2020-03-10]. Dostupné z: https://reactjs.org/docs/hooks-state.html

[31] KŘÍŽ, Pavel. React Hooks, které potřebujete znát. *Zdroják: O tvorbě webových stránek a aplikací* [online]. 2019 [cit. 2020-03-10]. Dostupné z: https://www.zdrojak.cz/clanky/react-hooks-ktere-potrebujete-znat/

[32] A Comparison Between Spring and Spring Boot. *Baeldung* [online]. Baeldung, 2019 [cit. 2020-03-25]. Dostupné z: https://www.baeldung.com/spring-vs-spring-boot

[33] Spring: IoC Containers. *Tutorials Point* [online]. Hyderabad, 2020 [cit. 2020-03-29]. Dostupné z: https://www.tutorialspoint.com/spring/spring_ioc_containers.htm

[34] PANKAJ, . Spring @Component. *JournalDev: Java, Java EE, Android tutorials* [online]. 2018 [cit. 2020-03-29]. Dostupné z: https://www.journaldev.com/21429/spring-component

[35] PICHLÍK, Roman. Automatický "sběr" objektů daného typu pomocí Springu. *Dagblog* [online]. Praha, 2020 [cit. 2020-03-29]. Dostupné z: https://dagblog.cz/automatick%C3%BD-sb%C4%9Br-objekt%C5%AF-dan%C3%A9ho-typu-pomoc%C3%AD-springu-b569e3f94337

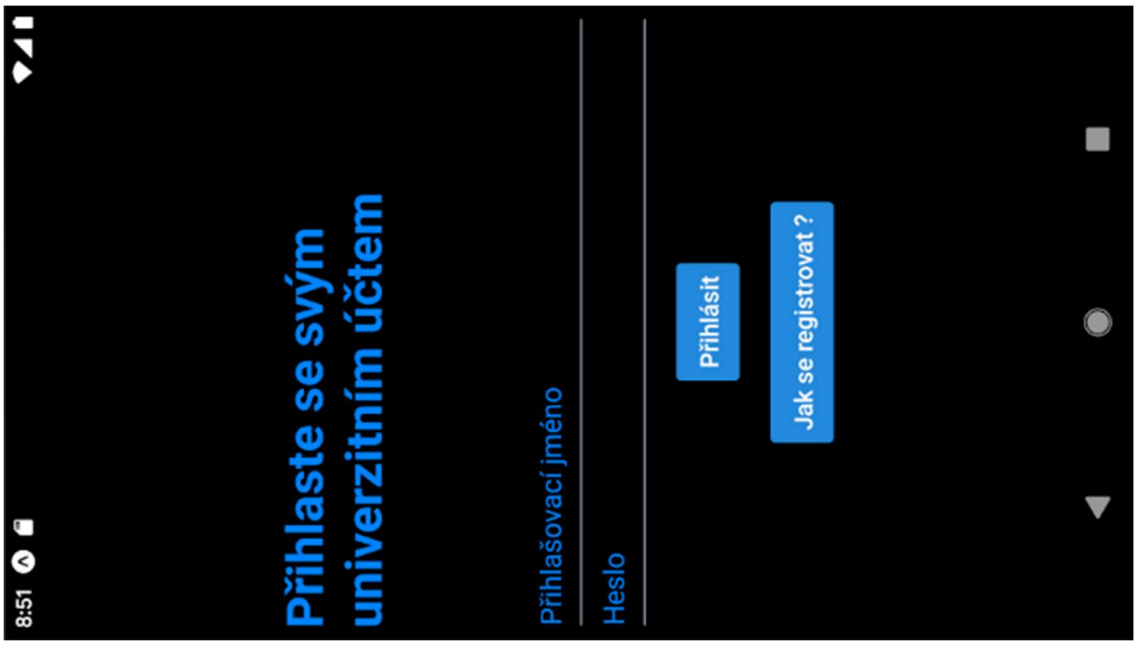
[36] PARASCHIV, Eugen. Spring RequestMapping. *Baeldung* [online]. Baeldung, 2020 [cit. 2020-03-29]. Dostupné z: https://www.baeldung.com/spring-requestmapping

[37] VĚNCESLAVA, Zellerová. Integrovaný knihovnický systém ALEPH. *WikiKnihovna: Knihovníci sobě* [online]. 2012 [cit. 2020-03-30]. Dostupné z: http://wiki.knihovna.cz/index.php/ALEPH

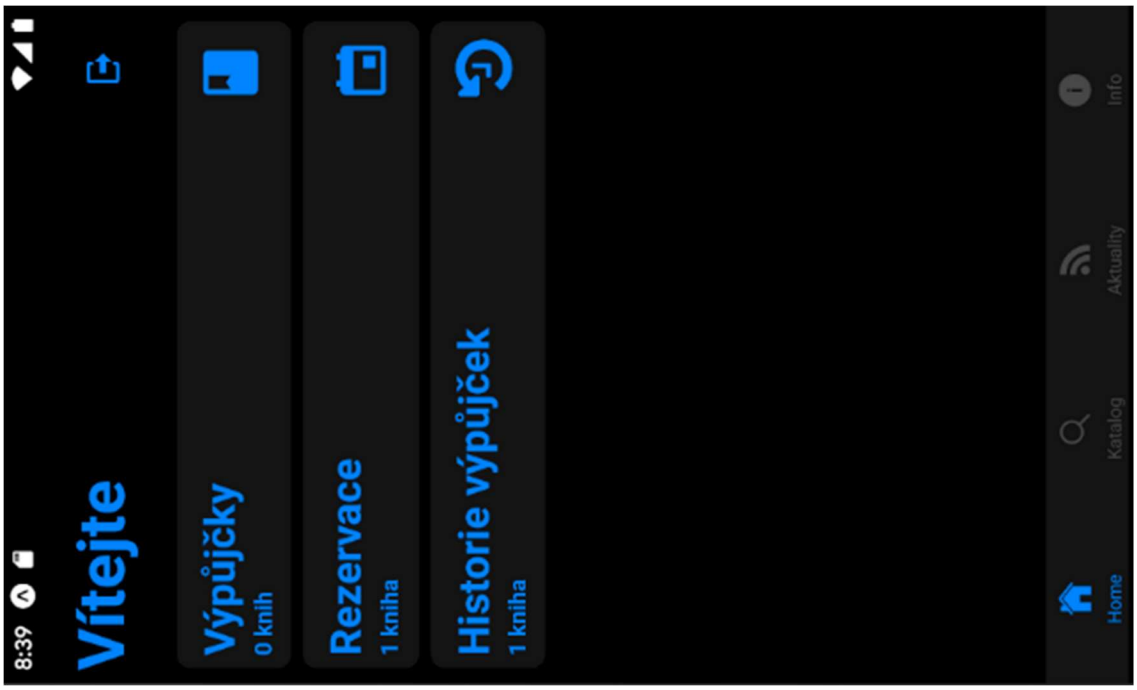
[38] Introduction to Aleph X-Services. *Ex Libris Developer Network* [online]. [cit. 2020-03-30]. Dostupné z: https://developers.exlibrisgroup.com/aleph/apis/Aleph-X-Services/introduction-to-aleph-x-services

7 Přílohy

Příloha 1 - Přihlašovací obrazovka



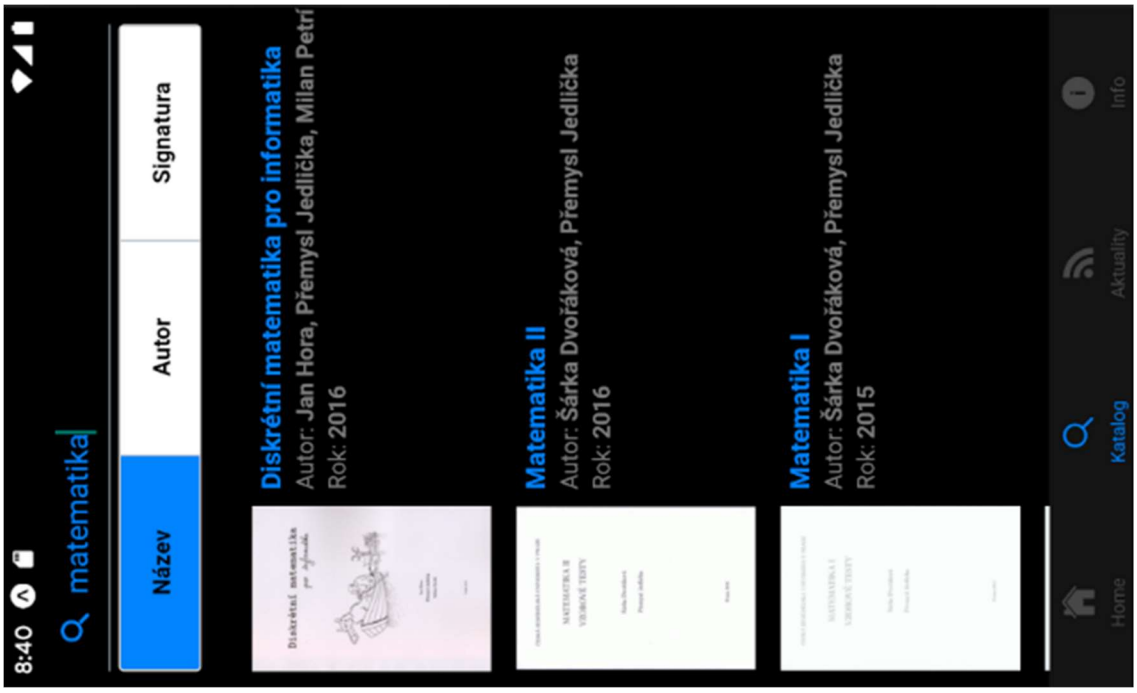
Příloha 2 - Domovská obrazovka



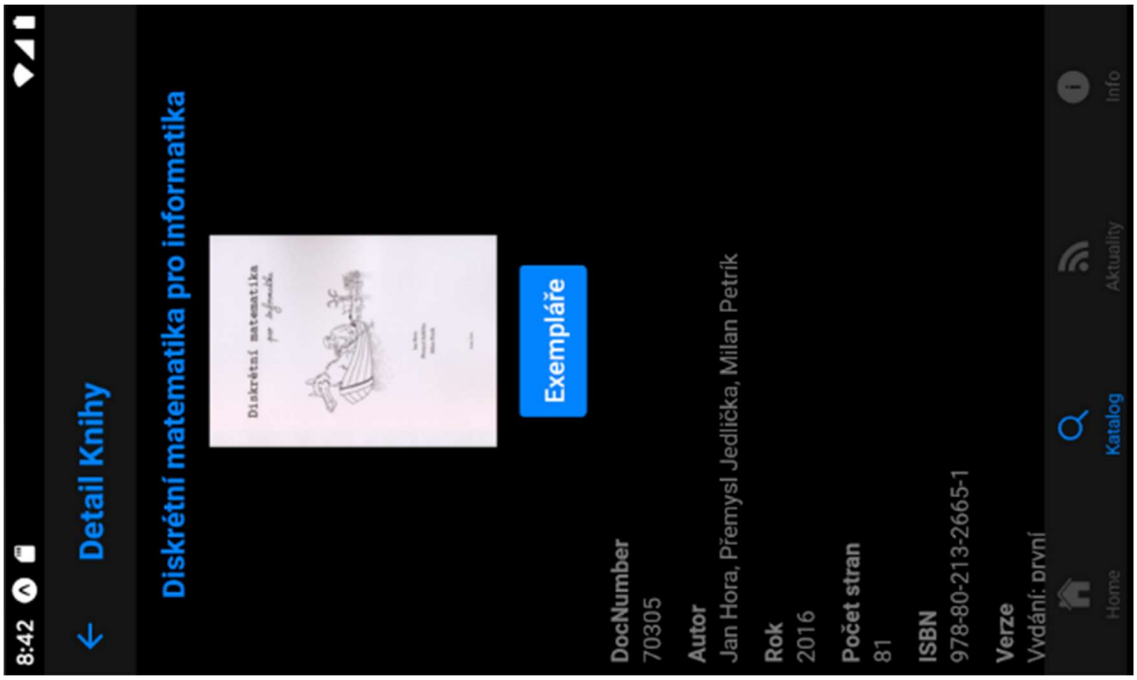
Příloha 3 - Obrazovka Rezervace

Příloha 4 - Obrazovka Historie výpůjček

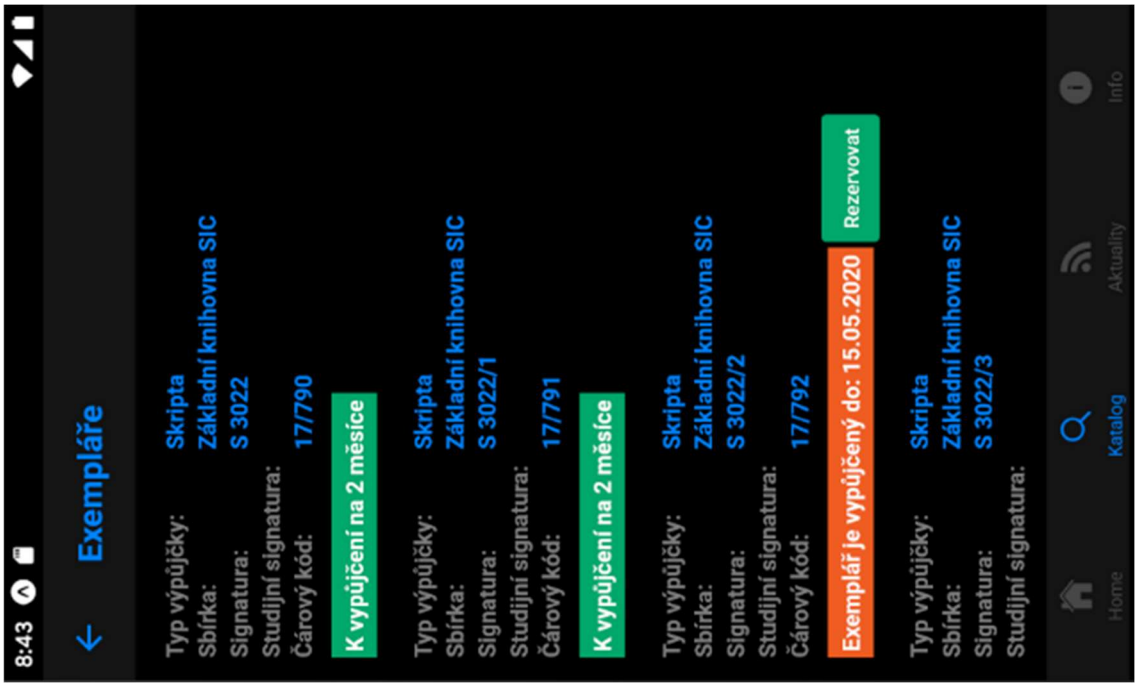
Příloha 5 - Obrazovka Katalog



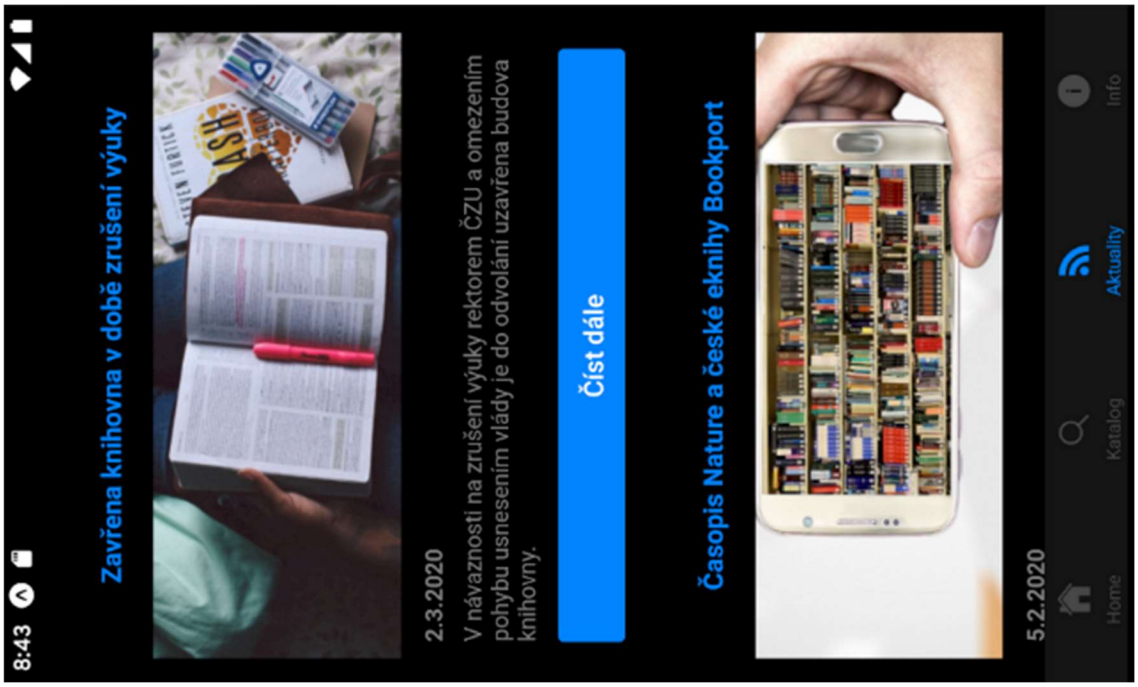
Příloha 6 - Obrazovka Detail knihy



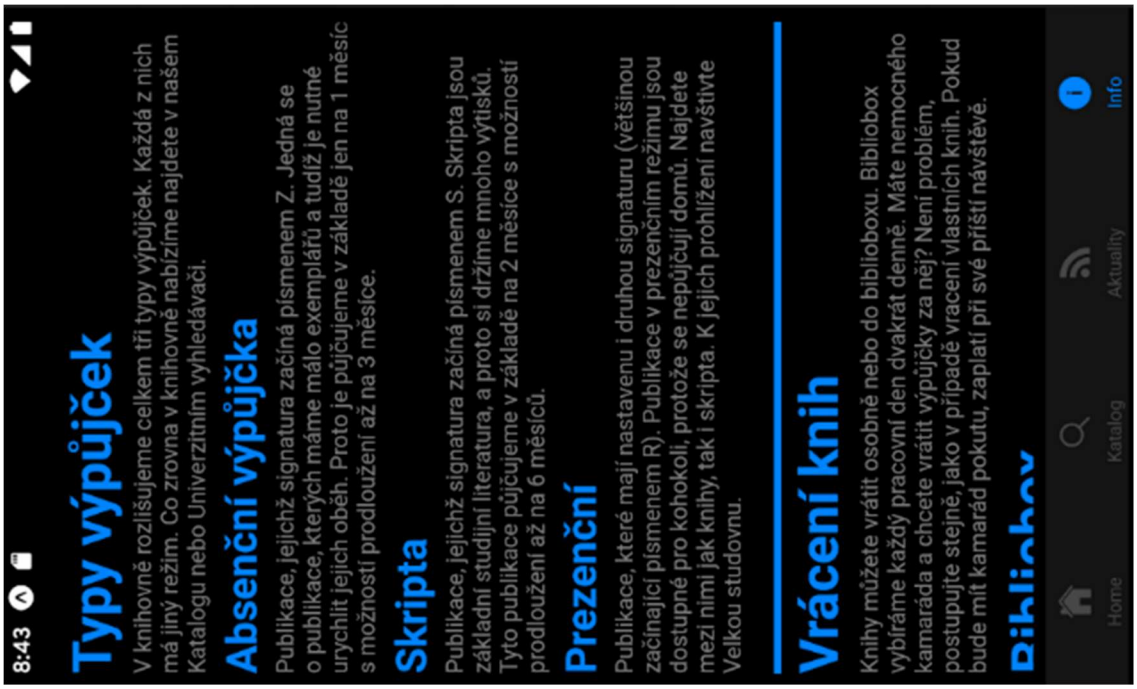
Příloha 7 - Obrazovka Exempláře



Příloha 8 - Obrazovka Aktuality



Příloha 9 - Obrazovka Info



Příloha 10 - CD se zdrojovými kódy aplikace