Kapitola 1

Východiská práce

V tejto kapitole sa čitateľ oboznámi zo základnými princípami logiky prvého rádu ktoré sú nevyhnutné na pochopenie práce, zistí aké prostriedky a technológie boli použité na vývoj aplikácie a ich krátky opis, a na záver je krátka analýza existujúcich prác alebo aplikácií, ktoré ma určitým spôsobom inšpirovali.

1.1 Logika prvého rádu

Logika prvého rádu je formálny systém používaný v matematike a informatike. Od jednoduchej výrokovej logiky sa odlišuje tým, že okrem logických spojok a že pridáva predikáty a kvantifikátory.

1.1.1 Jazyk

Jazyk $\mathcal{L}$ v prvorádovej logike je formálny jazyk ktorý definuje množinu konkrétnych symbolov, ktoré definujú syntax a sémantiku logiky.

Jazyk stredoškolskej výrokovej logiky definuje symboly *premenných* (napr. x, y), *logických spojok* ($\wedge, \vee, \rightarrow, \neg$) a *kvantifikátorov* ($\forall, \exists$). Tieto symboly nie sú najpohodlnejšie na zapisovanie vzťahov alebo vlastností objektov.

Logika prvého rádu zavádza do jazyka *symbol rovnosti* $\doteq$ a tzv. mimologické symboly. Medzi mimologické symboly patria *symboly konštánt* $\mathcal{C}_\mathcal{L}$, *funkčné symboly* $\mathcal{F}_\mathcal{L}$ a *predikátové symboly* $\mathcal{P}_\mathcal{L}$. Všetkým predikátovým a funkčným symbolom je priradená *arita*. Arita je kladné prirodzené číslo a predstavuje počet argumentov symbolu. Arita sa zapisuje ako horný pravý index pri názve symbolu. Napríklad $\texttt{nenávidí}^2$ je predikátový symbol s aritou 2.

Symbol konštanty z množiny $\mathcal{C}_\mathcal{L}$ jazyka $\mathcal{L}$ predstavuje konkrétny objekt ktorý sa nedá zameniť alebo konkrétnu hodnotu. Dá sa povedať, že sú podobné ako vlastné mená v prirodzenom jazyku alebo konštanty v programovacom jazyku. Napríklad symbol konštanty `Dom123` predstavuje konkrétny dom s číslom 123 v nejakej obci a nedá sa zameniť zo žiadnym iným. Príklad definovania množiny konštánt v jazyku je v 1.1.

$$\mathcal{C}_\mathcal{L} = \{\texttt{Dom123}, \texttt{Frantisek}, \texttt{Velkost}\} \tag{1.1}$$

Funkčný symbol z množiny $\mathcal{F}_\mathcal{L}$ jazyka $\mathcal{L}$ predstavuje jednoznačne určený vzťah.Napríklad funkčný symbol `cena(Produkt123)` predstavuje cenu produktu 123 v nejakom obchode. Produkt bežne máva iba jednu cenu, takže `Produkt123` má nejakú *jednoznačne* určenú cenu. Príklad definovania množiny funkčných symbolov v jazyku je v 1.2.

$$\mathcal{F}_\mathcal{L} = \{\texttt{cena}^1, \texttt{-}^2\} \tag{1.2}$$

Predikátový symbol z množiny $\mathcal{P}_\mathcal{L}$ jazyka $\mathcal{L}$ predstavuje určitú vlastnosť alebo vzťahy. Príklad definovania množiny predikátových symbolov v jazyku je v 1.3.

$$\mathcal{P}_\mathcal{L} = \{\texttt{nenávidí}^2, \texttt{chlapec}^1\} \tag{1.3}$$

V jazyku sú ďalej definové *symboly indivíduových premenných* z nejakej nekonečnej spočítateľnej množiny $\mathcal{V}_\mathcal{L}$. Príklad definovania množiny symbolov premenných v jazyku je v 1.4.

$$\mathcal{V}_\mathcal{L} = \{x, y, z\} \tag{1.4}$$

Množiny $\mathcal{V}_\mathcal{L}, \mathcal{P}_\mathcal{L}, \mathcal{F}_\mathcal{L}, \mathcal{C}_\mathcal{L}$ sú navzájom disjunktné.

1.1.2 Syntax

Keď už je jasné, čo je to jazyk logiky, teraz opíšem, aké sú pravidlá a obmedzenia syntaxe v logike prvého rádu.

1.1.2.1 Termy a formuly

Termy jazyka $\mathcal{L}$ predstavujú konkrétne (pomenované symbolmi konštánt) alebo nekonkrétne (pomenované symbolmi premenných) objekty, alebo pomenované pomocou funkčných symbolov. Príklady termov:

- nekonkrétne - `x`, `y`, `z`
- konkrétne - `Frantisek`, `Dom123`, `Velkost`
- vzťahy - `cena(x)`, `cena(Dom123)`, `-(x,y)`

Do množiny termov $\mathcal{T}_\mathcal{L}$ je každý symbol premennej a konštanty jazyka $\mathcal{L}$. Funkčný symbol patrí medzi termy iba v prípade, ak všetky jeho argumenty sú tiež termami.

Množina *formúl* jazyka $\mathcal{L}$ je rekurzívne definovaná nasledovne:

- ak sú α a β formuly, tak aj $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta)$ sú formuly
- ak je α formula, tak aj $\neg\alpha$ je formula
- všetky atomické formuly
- ak x je indivíduová premenná a α je formula, tak aj $\forall x\alpha$ a $\exists x\alpha$ sú formulami

1.1.3 Sémantika

1.1.3.1 Štruktúra

Hodnota formuly alebo termu jazyka $\mathcal{L}$ určuje *štruktúra*. Štruktúrou pre jazyk $\mathcal{L}$ je dvojica $\mathcal{M} = (M, i)$, kde

- M je *doména* štruktúry $\mathcal{M}$
  - je to množina symbolov, ktoré sa nenachádzajú v jazyku
- i je *interpretačná funkcia* štruktúry $\mathcal{M}$
  -

1.2 Analýza použitých technológií

V tejto sekcii sa čitateľ oboznámi s technológiami ktoré budú použité pri tvorbe aplikácie.

1.2.1 Javascript

Javascript je objektovo orientovaný, interpretovaný programovací jazyk využívaný hlavne vo webových aplikáciách. Väčšinou sa vykonáva na strane klienta (webový prehliadač) a reaguje na rôzne udalosti od užívateľa (napr. kliknutie myšou na nejaký element, zmenšenie okna a podobne). Vďaka tomuto je možné vytvárať dynamické, interaktívne webové aplikácie.

Javascript patrí medzi jazyky ktoré sú prototype-based. Je to spôsob, akým prebieha dedenie objektov. Každý objekt obsahuje atribút ktorý referuje na objekt ktorý ho vytvoril, resp. objekt od ktorého je zdedený. Takýmto spôsobom objekty vytvárajú reťaz kde na začiatku je objekt ktorý nemá žiadneho predka.

1.2.2 HTML

HTML je značkovací jazyk využívaný na tvorbu webových stránok a webových aplikácií. Definuje tagy, pomocou ktorých sa formátuje text, pridávajú obrázky alebo vytvárajú hypertextové linky na iné stránky. Webový server posiela HTML súbory ako odpoveď na požiadavku do webového prehliadača, ktorý dokáže tieto značky správne vyrenderovať.

V čase vývoja tejto aplikácie bol štandard HTML5, takže všetky HTML súbory aplikácie sú validné s touto verziou.

1.2.3 CSS

CSS je štýlovací jazyk na štýlovanie HTML elementov.

1.2.4 Bootstrap

Bootstrap je CSS a Javascript framework na štýlovanie webovej stránky. Má (okrem iného) predvytvorené mechanizmy ako rozložiť elementy na stránke pre rôzne rozlíšenia prehliadača - tzv. grid.

1.2.5 React

React je Javascript knižnica vytvorená spoločnosťou Facebook, Instagram a komunitou vývojárov. Originálnym autorom je vývojár v spoločnosti Facebook, Jordan Walke. Prvá verzia React-u vyšla v marci v roku 2013.

Hlavným cieľom tejto knižnice je efektívna tvorba používateľských rozhraní. React je navrhnutý tak, aby ľubovoľná zmena v nejakom komponente zabezpečila, aby sa efektívne aktualizovali len tie komponenty, ktorých sa táto zmena týka.

V tejto kapitole sa budem snažiť v krátkosti opísať architektúru React-u na príklade, kde budem vytvárať rozhranie na pridávanie textových reťazcov do zoznamu a ich mazanie. Ukážky kódov sú zjednosušené, aby boli ľahšie čitateľné.

1.2.5.1 Elementy a komponenty

Na úvod je potrebné vysvetliť čo je to element a komponent a aké sú medzi nimi rozdiely.

Element je prvok, ktorý chceme vidieť na stránke. Je to objektová reprezentácia DOM objektu. Každý element má definovaný typ, rôzne atribúty a zoznam potomkov. Potomkovia elementu sú tiež elementy, ktoré sú v ňom vnorené. Jednoduchý element sa v Reacte vytvára pomocou funkcie React.createElement ako je vidieť v Listing 1.

Listing 1.1: Vytvorenie jednoduchého elementu

```
const element = React.createElement (
    'button',
    {
        id: 'add-btn',
    }
    'Add'
);
```

Prvý argument funkcie je typ elementu. Môže to byť string alebo objekt druhého elementu. Druhý argument je objekt argumentov elementu. Tretí argument je zoznam potomkov elementu. Konštanta element bude vyzerať nasledovne

```
const element = {
    type: 'button',
    props: {
        id: 'add-btn',
        children: 'Add'
    }
}
```

Typ elementu je veľmi dôležitý argument. Ak je to textový reťazec, musí to byť jeden z DOM elementov (napr. button, p, div). Ak je to nejaký objekt, funkcia vráti objekt, ktorý vrátil tento typ elementu s atribútmi, ktoré sú v druhom argumente.

Základným rozdielom medzi elementom a komponentom je, že komponent príjima atribúty, a na základe nich vracia elementy.

Druhým a jednoduchším spôsobom vytvárania elementov je pomocou JSX, čo je syntaxové rozšírenie pre Javascript. Pomocou JSX sa dajú elementy definovať ako klasické

HTML tagy a vytvárať tak prehľadnejší a ľahšie pochopiteľnejší kód. Element ktorý sme vytvárali v Listing 1 sa dá prepísať do JSX syntaxe nasledovným spôsobom:

Komponenty si môžeme predstaviť ako skupinu elementov. Komponent

1.2.5.2 Stav komponentu

Stav komponentu je jednoduchý objekt, podľa ktorého React vie, či je potrebné komponent aktualizovať a vykresliť alebo nie. Do tohto objektu je možné ukladať dáta, ktorých zmena má vyvolať prekreslenie komponentu. V stave komponentu by mali byť iba tie údaje, ktoré sa nejakým spôsobom vykresľujú, alebo od nich závisí, ako sa komponent vykreslí.

Objekt stavu sa definuje v konštruktore komponentu.

Listing 1.2: Definovanie stavu v konštruktore komponentu

```
class List extends React.Component {
        constructor(props) {
                super(props);
                this.state = {
                        items: []
                };
        }
        ...
}
```

Listing 2 demonštruje prvotné definovanie stavu. Aktuálne je v stave iba prázdne pole, do ktorého sa budú ukladať textové reťazce ktoré zadá užívateľ. Akonáhle sa do tohto poľa pridá prvok, odstráni prvok alebo sa aktualizuje, komponent List sa automaticky prekreslí zavolaním funkcie render().

Priame priradenie objektu k premennej this.state je možné iba v konštruktore. Mimo konštruktora sa stav definuje pomocou funkcie this.setState(). Táto funkcia dostáva ako argument zmenený objekt stavu. Listing 3 ukazuje, ako sa dá napríklad pridať prvok do poľa items a tak aktualizovať stav komponentu.

Listing 1.3: Aktualizovanie stavu komponentu

```
/*
*       @param String item
*/
```

```
addItem(item) {
        newItems = this.state.items.push(item);
        this.setState({
                items: newItems
        });
}
```

Zmeniť stav sa samozrejme dá aj priamym priradením, ale nenastane prekreslenie komponentu. Funkcia this.setState() zabezpečí, aby sa komponent prekreslil, teda aby sa vykonala funkcia render().

1.2.5.3 Životný cyklus komponentu

Komponent môže byť v jednej z troch fáz - vo fáze vytvárania, aktualizovania alebo vymazávania.

Komponent sa vytvára funkciou ReactDOM.render(). Táto funkcia dostane referenciu na objekt komponentu a DOM element, do ktorého sa má komponent vyrenderovať. Táto funkcia zaradí komponent do fázy vytvárania. Fáza vytvárania prebieha v nasledovných krokoch:

- getDefaultProps() - vráti predvolené vlastnosti komponentu. Môžu to byť vlastnosti ktoré potrebujeme mať vyplnené v prípade, ak neboli dodané z vonka
- getInitialState() - definuje prvotný stav komponentu
- componentWillMount() - v tomto kroku sa môžu vykonať akcie, ktoré chceme mať vykonané pred vykreslením komponentu
- reder() - vykreslenie komponentu
- componentDidMount() - nastane tesne po vykreslení komponentu

Výsledkom fázy vytvárania je, že komponent je vykreslený v DOM a dá sa s ním manipulovať.

Ak je už komponent vytvorený, do fázy aktualizácie môže prejsť zavolaním setProps() alebo setState() alebo forceUpdate(). Fáza aktualizácie prebieha v týchto krokoch:

- componentWillReceiveProps(nextProps) - komponent dostane nové vlastnosti volaním funkcie setProps()

- shouldComponentUpdate(nextProps, nextState) - v tomto kroku sa rozhoduje, či je potrebná aktualizácia. Porovnávajú sa aktuálne vlastnosti a stav s tými novými. Ak sa detekuje zmena, vráti true a pokračuje do ďalšiho kroku, inak false a fáza skončí.
- render() - prekreslenie komponentu
- componentDidUpdate() - nastane tesne po prekreslení komponentu

Vytvorený objekt môže tiež prejsť do fázy vymazávania. Táto fáza nastane zavolaním funkcie ReactDOM.unmountComponentAtNode(). Táto fáza má iba jeden krok componentWillUnmount() a v nej sa odporúča uvoľniť všetky zdroje (napr. časovač, súbory) ktoré komponent používal. Po skončení tejto fázy je komponent vymazaný a dá sa znova vytvoriť funkciou ReactDOM.render().

1.2.5.4 Udalosti

Najdôležitejšou súčasťou React-u je reagovanie na rôzne udalosti od užívateľa. Každý element môže mať definované, akú funkciu vykonať pri vyvolaní nejakej udalosti.

Komponentu je možné definovať aj vlastné udalosti. Napríklad ak chceme, aby sa vykonala nejaká funkcia pri zmene stavu nejakého komponentu, predáme mu ju cez props.