Další témata

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMIWEBA Webové aplikace

Další témata

- Doposud představné je postačující pro tvorbu různorodých (architektura, požadavky) i velmi komplexní aplikací.
- Pro zajímavost si stručně představíme ještě několik technologií, které umožňují využít to co jsme se v tomto kurzu naučili pro tvorbu desktopových a mobilních aplikací.

Nativní aplikace

- Položme si následující otázku. Je pro uživatele přívětivější používat webovou aplikaci (nebo stránku) skrze webový prohlížeč nebo použít nativní aplikaci která zprostředkovává stejnou funkcionalitu.
- Mnohé studie prokazují, že v případě mobilních zařízení je nativní aplikace pro uživatele výrazně přívětivější cesta (výrazně lepší UX). Nativní aplikace je nainstalovaná, má na ploše ikonu, funguje v režimu offline a respektuje UI zásady daného operačního systému.
- Z výše uvedených důvodů je tedy velmi zajímavé kromě webové aplikace vyvinout i její mobilní verzi.
- Samozřejmě vyvinout nativní aplikaci k existující webové aplikaci znamená aplikaci vytvořit (od začátku) pomocí jiných technologií. To může být velmi obtížné, ale jen touto cestou je v současnosti možné dosáhnout na z pohledu UX nejlepší řešení.
- Zjevnou nevýhodou je platformová závislost. Například aplikace pro Android by se měla chovat jinak než aplikace pro iOS (pokud ma repsektovat zvyklosti daného systému).

Multiplatformní aplikace

- Jako alternativa k nativnímu vývoji se nabízí vyvinout multiplatformní (někdy také hybridní) aplikaci.
- Ty jsou založeny na poměrně jednoduchém principu. Aplikace je tvořena nativní skořápkou (pouze minimální nativní aplikace) obsahující komponentu webview, která zpřístupňuje jádro webového prohlížeče.
- Skutečná aplikace je webová aplikace, která běží právě v této komponentě. Tento vývoj je poměrně populární jelikož umožňuje vyvinout snadno multiplatformní aplikaci, přičemž je sdílen kód na různých platformách. Nevýhodou je horší UI a UX (byť je v těchto oblastech v poslední době značný pokrok).
- Aby se mohla webová aplikace běžící uvnitř webview komponenty chovat jako nativní aplikace, musí mít možnost ovládat hardware daného zařízení. Například zpřístupnit fotoaparát či datové uložiště.
- K tomu účelu se využívají frameworky pro vývoj multiplatformních aplikací jako například Apache Cordova nebo PhoneGap.
- Tento přístup je postupně je postupně utlačován technologiemi typu React native. Ty poskytují API pro přístup k systémovým službám a usnadňují distribuci nativní skořápky (například ExpoGo).

Progresivní webové aplikace

- Díky neustále se rozrůstajícím možnostem webových prohlížečů a webových technologí je možné na místo výše uvedených frameworků použít jako interface mezi hardware právě webový prohlížeč. Díky tomu je možné z běžné webové stránky/aplikace vytvořit mobilní nebo desktopovou aplikaci. Aplikace tohoto typu se označují jako progresivní webové aplikace (PWA).
- PWA aplikace jsou nezávislé na OS, responzivní, umožňují práci offline, instalovatelné, bezpečné (lze je provozovat pouze na HTTPS) a dostupné skrze URL (není je možné distribuovat skrze aplikační obchody).
- Každá progresivní webová aplikace obsahuje manifest soubor, který udržuje základní informace o aplikaci. Offline modu je zajištěno pomocí JavaScriptu, přesněji řečeno servis workera, který slouží jako proxy middleware.
- Ukázka PWA aplikace je k dispozici na https://github.com/martin-trnecka/pwa.
- Vyzkoušet je možné aplikaci na adrese https://martin-trnecka.github.io/pwa/.

HTML API

- Zpřístupnění hardware je možné skrze HTML API.
- Například IndexedDB API umožňuje v prohlížeči provozovat relační databázi a ukládat perzistentně data.
- Analogicky je možné využit Storage API, které jsme již představili. Fotoaparát je možné zpřístupnit pomocí MediaDevices rozhraní.

Electron

- Elektron je technologie umožňující vytvářet multiplatformní desktopové aplikace. Princip je analogický jako v případě multiplatformních mobilních aplikací.
- Elektron poskytuje API, které zpřístupňuje části operačního systému webové aplikaci, které běží v node.js. Renderování UI je zajištěno pomocí webview komponenty.
- Nejprve si vytvoříme kostru aplikace. Se základní konfigurací nám pomůže npm init.

```
mkdir my-electron-app
cd my-electron-app
npm init
npm install electron # pouze pokud není naistalován
```

Electron

- npm init vytvoří soubor package.json. Ten si mírně upravíme.

```
{
  "name": "my-electron-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "electron ."
  },
  "author": "Me",
  "devDependencies": {
    "electron": "^21.3.1"
  }
}
```

- Podoba se může lišit dle požadavků. Hlavní je scripts start, který nám spustí aplikaci.

Electron

- Vytvoříme si html stránku, které bude realizovat okno aplikace.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'" />
    <meta http-equiv="X-Content-Security-Policy" content="default-src 'self'; script-src 'self'"
/>
    <title>Hello from Electron!</title>
  </head>
  <body>
    <h1>Hello from Electron!</h1>
    <p>👋</p>
  </body>
</html>
```

Electron

- Následně vytvoříme skript, který vyrenderuje okno aplikace.

```
const { app, BrowserWindow } = require('electron');

// vytvoření okna
const createWindow = () => {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
  });

  win.loadFile('index.html'); // načtení obsahu okna
};

// ovládání a a manipulace s oknem
app.whenReady().then(() => {
  createWindow();

  // aplikace na MacOS mohou běžet i bez okna, v takovémto případě
  // znovuspuštění aplikace pouze renderuje nové okno
  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow();
    }
  });
});

// zavření okna na Windows a Linux, na MacOS (darwin) se nepoužívá
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});
```

Electron

- Aplikaci spustíme příkazem níže.

```
npm run start
```

- Hlavní instance electron aplikace běží v omezeném node.js, který má přístup k operačnímu systému. Z bezpečnostních důvodů je renderování UI mimo node.js (stará se o něj jádro prohlížeče). K dispozici je celá řada Electron modulů (API), které zabezpečují různé (bežné) funkce a UI prvky, například Notification, Menu, net a mnoho dalších.

Electron

- Propojení všech částí elektron aplikace realizuj preoload skript. Kód uložený v souboru preload.js následuje.

```
const { contextBridge } = require('electron')

contextBridge.exposeInMainWorld('data', {
  node: () => process.versions.node,
  chrome: () => process.versions.chrome,
  electron: () => process.versions.electron,
  team: Array('Jack', 'Samantha', 'Daniel', 'Teal\'c'),
});
```

Electron

- Použití preload skriptu uvedeme v metodě createWindow(), kterou jsme ukázali dříve.

```
// vytvoření okna
const createWindow = () => {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      // potřebné kvůli správným adresám na různých OS
      preload: path.join(__dirname, 'preload.js'),
    },
  });

  win.loadFile('index.html'); // načtení obsahu okna
};
```

Electron

- Přidáme skript pro úpravu renderování do index.html a přidáme potřebné elementy (team a information).

```
const information = document.getElementById('info');
information.innerText = `This app is using Chrome (v${data.chrome()}), Node.js (v${data.node()}), and Electron
(v${data.electron()})`;

const team = document.getElementById('team');
data.team.forEach((t) => team.innerHTML += `${t}`);
```

Electron

- Propojení mezi aplikací a oknem je třeba realizovat pomocí IPC (jedná se o samostané procesy). Nejprve rozšíříme preload.js.

- const { contextBridge, ipcRenderer } = require('electron')

- contextBridge.exposeInMainWorld('data', {
- node: () => process.versions.node,
- chrome: () => process.versions.chrome,
- electron: () => process.versions.electron,
- team: Array('Jack', 'Samantha', 'Daniel', 'Teal\'c'),
- ping: () => ipcRenderer.invoke('ping'), // IPC
- });

Electron

- Přidáme obsluhu v okně.

```
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path')

// vytvoření okna
const createWindow = () => {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
    },
  });
```

Electron

- Do souboru index.html přidáme tlačítko a do souboru render.js jeho obsluhu.

```
// obsluha tlačítka
const button = document.getElementById('button');

button.addEventListener('click', async () => {
  const response = await window.data.ping();
  information.innerHTML = response;
});
```