Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Nástroj pro analýzu responzivního chování webové stránky

2024

Bc. Lukáš Kubík

Vedoucí práce:
RNDr. Martin Trnečka, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

Bibliografické údaje

Autor:	Bc. Lukáš Kubík
Název práce:	Nástroj pro analýzu responzivního chování webové stránky
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Aplikovaná informatika, Specializace: Vývoj software
Vedoucí práce:	RNDr. Martin Trnečka, Ph.D.
Počet stran:	61
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Bc. Lukáš Kubík
Title:	A tool for a responsive web page behavior analysis
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Applied Computer Science, Specialization: Software Development
Supervisor:	RNDr. Martin Trnečka, Ph.D.
Page count:	61
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Tato diplomová práce popisuje implementaci nástroje umožňujícího jednoduchou analýzu responzivního chování a přístupnosti webové stránky. Důraz byl kladen především na snadné automatické použití. Nástroj například generuje souhrnný report, dle zadaných parametrů. Součástí práce je i stručný přehled a srovnání existujících řešení.

Synopsis

This Master Thesis includes the implementation of a tool that allows a simple analysis of the responsive behaviour and accessibility of a website. The primary concern was the easy automatic use. For example, the tool generates a summary report, depending on specified parameters. The thesis also includes a general overview and comparison of existing solutions.

Klíčová slova: responzivní design; responzivní obrázky; fluidní mřížky; viewport; breakpoint; media query

Keywords: responsive design; responsive images; fluid grids, viewport; breakpoint; media query

Děkuji RNDr. Martinu Trnečkovi, Ph.D. za odborné vedení, konzultace, cenné rady při implementaci nástroje a pomoc se zpracováním této diplomové práce.

Odevzdáním tohoto textu jeho autor|ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval|a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

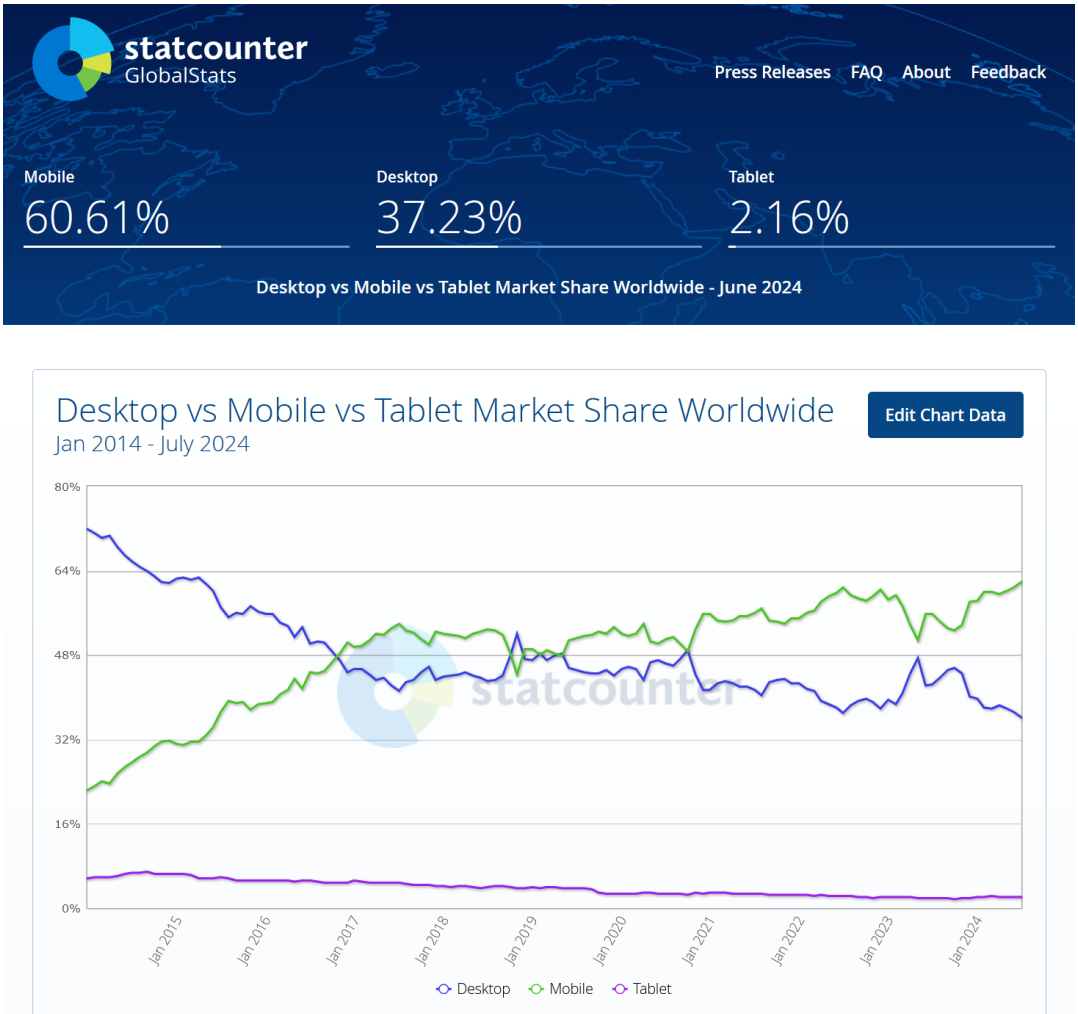
Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Tato diplomová práce je zaměřena na analýzu responzivního chování webových stránek. V dnešní době většina z nás považuje responzivní web design za samozřejmost. Když navštívíme webovou stránku očekáváme, že bude fungovat a vypadat dobře na všech našich zařízeních bez ohledu na to, jakou velikost má jeho displej. Z bližšího pohledu je pro přizpůsobení stránky podstatná hlavně jeho šířka. Nutno zmínit, že pro nás potřebná hodnota šířky není přímo rozlišení displeje, ale určuje ji šířka viewportu daného zařízení (blíže podsekce 6.4).
V době vzniku webových stránek (počátkem devadesátých let) vývojáři nemuseli věnovat tolik času přizpůsobení webové stránky různým velikostem obrazovky (tedy responzivnímu designu). Tehdy měla většina zařízení přistupujících na stránku monitory s rozlišením 640×480, 800×600 nebo 1024×768 pixelů. Dnešní situace je však zcela odlišná.



Obrázek 1: Statcounter globální statistiky

Podívejme se na obrázek 1, kde můžeme vidět statistiky od webové stránky Statcounter zaměřující se na analýzu celosvětové návštěvnosti webu. Podle těchto statistik v roce 2017 převýšil přístup k webovým stránkám z mobilních zařízení počet přístupů ze stolních počítačů (desktop). K červenci 2024 tvoří mobilní zařízení více jak 60 %, druhý desktop má o více než třetinu méně (37,23 %) a tablety jsou stále spíše okrajová záležitost (2,16 %).
S rostoucím počtem lidí, kteří přistupují k internetu z mobilních zařízení, už nám nemůže stačit mít statický design webových stránek, který vypadá dobře pouze na monitoru počítače. V současnosti máme mnoho různých typů zařízení, od malých displejů mobilních telefonů, přes tablety a laptopy, až po velké monitory osobních počítačů. Hodnota šířky viewportů dnešních zařízení je tak velice rozdílná, a proto je nutné, aby byl obsah naší stránky responzivní. Jakým způsobem (technikou) je dnes možné dosáhnout responzivních webových stránek se blíže zabývá sekce 2 za tímto úvodem.
Další sekce 3 a 4 obsahují popis všech technologií a přídavných balíčků, které byly použity při implementaci přiloženého nástroje (součást této práce). V sekci 5 s technickou (programátorskou) dokumentací se popisuje, jak je zmíněný nástroj rozdělen do jednotlivých souborů a složek. Jsou zde podrobně popsány konkrétní funkce nástroje, a také několik ukázek zdrojového kódu v programovacím jazyku JavaScript (včetně použitých knihoven). Sekce 6 (uživatelská dokumentace) popisuje co nástroj obsahuje a jak s ním pracovat. Na stránce `Breakpoints` jsou především podrobně rozebrány výsledky analyzované stránky. Na stránce `Viewports` se jedná o možnosti manipulace s viewporty. Nachází se tady také několik obrázků různých částí nástroje, aby se uživatel lépe zorientoval v popisovaných položkách. Následující sekce 7 obsahuje tabulkové srovnání obou částí nástroje s podobně zaměřenými existujícími nástroji. V sekci 8 je uvedeno pár návrhů k možnosti rozšíření do budoucna. Za touto části se nachází ještě závěrečné shrnutí (s popisem průběhu) vývoje nástroje a celé práce.

2 Responzivní design

Responzivní design je o vytváření webových stránek přizpůsobujících se danému zobrazovacímu zařízení. Tedy od malých displejů mobilních telefonů přes tablety, až po velké monitory osobních počítačů. Responzivní stránka se totiž automaticky přizpůsobí pro různé velikosti displeje, přesněji viewportu. Bližší popis viewportu je v podsekci 6.4, hodnoty viewportů jsou pak v tabulce 2.
Přizpůsobováním webové stránky se zabýval už v roce 2004 Cameron Adams. Ten popsal metodu použití JavaScriptu [1] k výměně různých kaskádových stylů [2] v závislosti na velikosti okna prohlížeče. Adamsova technika tehdy pro vývojáře představovala trochu práce navíc, umožňovala však určitou kontrolu nad rozložením stránek a fungovala jako raná verze breakpointů předtím, než se začaly používat. Autorem pojmu „responzivní design" se stal Ethen Marcotte, když ho v roce 2010 popsal formou článku pro internetový časopis A List Apart [3]. Marcotte označil za klíčové pro vytvoření responzivního webu 3 komponenty: responzivní obrázky (responsive images), fluidní mřížky (fluid grids) a media queries.

2.1 Responzivní obrázky

Responzivní obrázky se automaticky přizpůsobí velikosti viewportu. Pokud chceme obrázek s automatickou změnou velikosti (zvětšení/zmenšení), pak stačí pro obrázek nastavit CSS atribut `width` na `100%` a atribut `height` na `auto`. Jednoduchý příklad, jak vytvořit responzivní obrázky je na zdrojovém kódu 1.

```
1 <!-- adding HTML -->
2 <img src="nature.jpg" alt="Nature" class="responsive">
3
4 /* adding CSS */
5 .responsive{
6   width: 100%;
7   height: auto;
8 }
```

Zdrojový kód 1: HTML, CSS responzivní obrázky

2.2 Fluidní mřížky

Fluidní layout je nezbytnou součástí moderního responzivního designu. Fluidní mřížky mají sloupce s dynamicky se měnící hodnotou šířky (typicky v procentech a zlomcích). Tento přístup automaticky zvětšuje, nebo zmenšuje různé kontejnery (se sloupci) na základě velikosti viewportu zobrazovacího zařízení.

Rozšířený způsob, jak vytvořit fluidní mřížky je použít systém Flexbox (Flexible Box Layout) [4] nebo CSS grid systém [5]. Pokud budeme chtít pomocí Flexboxu vytvořit například dvousloupcové rozvržení vhodné pro většinu velikostí viewportů, stačí do stránky přidat zdrojový kód 2.

```
1  <!-- adding HTML -->
2  <div class="container">
3    <div class="column-left">left column</div>
4    <div class="column-right">right column</div>
5  </div>
6
7  /* adding CSS */
8  .container {
9    display: flex;
10 }
11 .column-left {
12   flex: 50%;
13 }
14 .column-right {
15   flex: 50%;
16 }
```

Zdrojový kód 2: HTML, CSS fluidní mřížky Flexbox

Když ale budeme potřebovat automaticky změnit dvousloupcové rozložení na jeden sloupec, tedy vhodné pro malá zařízení jako jsou mobilní telefony, pak už musíme použít fluidní mřížky v kombinaci s media queries [6] (podsekce 2.3). Flexbox má také mnoho dalších unikátních vlastností a je to složitější téma, na které v této práci není dostatek prostoru.
Vytvořit fluidní mřížku od začátku nemusí být vždy snadný úkol. Dobrý nápad je proto na její tvorbu použít nějaký existující CSS framework. Mezi doporučované pro tuto úlohu patří například Bootstrap [7]. Jedná se o jeden z nejoblíbenějších frameworků známých například pro svoji univerzální kompatibilitu s běžně používanými preprocesory, pluginy a další. Jiným vhodným kandidátem je Skeleton [8]. Jde o „odlehčený“ (boilerplate) framework a je vhodný pro začátečníky. Přestože ve srovnání s ostatními systémy může nabízet méně funkcí, pro menší projekty dostačuje.
Máme také možnost vytvořit si mřížku (grid) [5] vlastní. Pokud budeme chtít například stejnou mřížku jako obsahuje knihovna Bootstrap, tak nám to zabere přibližně stejný počet řádků kódu, jako když si vytvoříme vlastní. Příklad základního použití CSS Grid Layout k vytvoření mřížky se dvěma sloupci a třemi řádky můžeme vidět na zdrojovém kódu 3.

```
<!-- adding HTML -->
<div class="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>

/* adding CSS */
.container {
  display: grid;
  grid-template-columns: auto auto;
}
```

Zdrojový kód 3: HTML, CSS fluidní mřížky CSS Grid Layout

2.3 Media queries a breakpointy

Media queries [6] začínají klíčovým slovem (pravidlem) `@media` a následně je třeba zadat typ zařízení, pro který je chceme použít. Může to být obrazovka `screen`, tisk `print` nebo zařízení schopné převádět text na řeč `speech` (pro čtečku obrazovky). Obrazovka `screen` bude zvolena u čehokoliv s displejem například monitor počítače nebo telefon a tisk `print` se používá, když chceme vytisknout stránku. Je možné vybrat i všechny typy dohromady, existuje k tomu klíčové slovo `all`.
Media queries podporují tři standartní logické operátory: `and`, `or` a `not` stejně, jako v mnoha programovacích jazycích. Uvnitř složených závorek je CSS kód k vykonání a uvnitř kulatých závorek jsou atributy s požadovanými hodnotami. Ty lze použít například v závislosti na orientaci prohlížeče (`portrait` či `landscape`), nejčastěji se ale používají atributy `min-width` a `max-width` a jejich použitím vznikají *breakpointy*. Breakpoint je jednoduše řečeno hodnota šířky (`width`) viewportu, při které je aplikován media query (zde breakpoint) a přitom se vyhodnotí část CSS kódu ve složených závorkách. Pomocí breakpointů, tak můžeme měnit skutečný počet sloupců ve fluidní mřížce v závislosti na velikosti viewportu. Media queries můžeme použít libovolný počet.

V tabulce 1 můžeme vidět nejrozšířenější hodnoty breakpointů. Příklad použití konkrétní hodnoty atributů `min-width` a `max-width` najdeme v podsekci 6.3.2.

Tabulka 1: Hodnoty breakpointů

atribut	**hodnota**	**typ zařízení**
min-width	576px	mobil
min-width	768px	iPad, tablet
min-width	992px	laptop
min-width	1200px	desktop
min-width	1400px	desktop, TV
max-width	575.99px	mobil (na výšku)
max-width	767.99px	mobil (na šířku)
max-width	991.99px	iPad, tablet
max-width	1199.99px	desktop
max-width	1399.99px	desktop, TV

2.4 Mobile First

Jak bylo zmíněno v úvodu, mobilní zařízení dnes dominují v přístupu k webovým stránkám a bez přizpůsobení našich stránek pro mobily se tedy neobejdeme. Proto je vhodné se při vývoji stránek první zaměřit na přizpůsobení pro mobil a až následně pro další typy zařízení. Tato metoda (postup) se nazývá „Mobile First" a my si zde uvedeme pár výhod jejího použití.

- Rychlost načítání
  Pokud budeme načítat pouze to, co je v mobilním zobrazení nezbytně nutné, tak se naše stránky budou načítat rychleji. Například můžeme mít dekorativní prvky nebo postranní panel, který na větší obrazovce funguje (a vypadá) dobře, ale na mobilu může být vynechán. Mobilní uživatelé jsou obvykle ochotni čekat pouze jednotky sekund (asi tři) na načtení obsahu stránky, než ji opustí a my přijdeme o návštěvníky.

- Řešení navigace
  Protože umístit navigační nabídku na malý displej mobilu je obtížný úkol, donutí nás to řešit tento navigační problém předem a ušetříme čas s jeho úpravou v budoucnu.

Závěrem této sekce je dobré explicitně zmínit, že responzivní design není o vývoji mobilní stránky (aplikace). Pokud vezmeme celý obsah stránky a vložíme ho do jednoho sloupce (jako to může stačit na mobilu), tak nám to nepomůže. Takové řešení by nevypadalo (ani by se neovládalo) dobře. Uvedeme si krátký příklad. Každý známe videoserver Youtube. Dovedli bychom si představit, že by na našem monitoru nebo televizi byl všechen obsah libovolného kanálu/kategorie jen v jednom sloupci a ostatní sloupce by byly pryč? To je přece nesmysl, takto by tam bylo příliš mnoho nevyužitého místa.
Aby byla tato sekce úplná musíme zmínit také adaptivní design. Adaptivní design řeší přizpůsobení tak, že se vytvoří stránka pro každé zařízení zvlášť. Musíme tedy vytvořit více verzí stránek, kdy je každá vhodná přímo pro daný typ zařízení. Na druhou stranu u responzivního design vytváříme jenom jednu stránku, a ta se pomocí různých technik (breakpointy a další) sama přizpůsobuje podle velikosti viewportu zařízení. Pro přizpůsobení webové stránky je tedy hlavně z pohledu udržitelnosti nejlepší použít responzivní design (tedy mít jen jednu stránku).

3 Použité technologie

3.1 HTML

HTML (Hypertext Markup Language) je značkovací jazyk popisující strukturu obsahu webové stránky. Jde o nejzákladnější webovou technologii pocházející z roku 1991 (první standard, Tim Berners-Lee). Vývoj pozdějších verzí probíhal pod organizací W3C (World Wide Web Consortium). V současnosti se používá specifikace HTML Living Standard [9]. HTML popisuje sémantiku a skládá se z řady jednotlivých elementů. Element definuje značka (tag), nejčastěji se používají značky dvě (počáteční a koncová). Například tag `<h1>` je pro nadpis, `<p>` se používá na odstavce, `<a>` je pro odkaz a tak dále. Značka (počáteční) může obsahovat atributy, které jsou tvořené příslušným názvem a hodnotou. Tyto elementy lze také pojmenovávat pomocí atributů `id` a `class`. Platí, že HTML kód by měl být validní. Ověřit to můžeme pomocí validátoru. Jeden najdeme na adrese: `https://validator.w3.org`.

3.2 CSS

CSS (Cascading Style Sheets, také kaskádové styly) [2] jsou jazyk používaný ke stylování HTML dokumentu, tedy popisující, jak by se měly prvky (elementy) zobrazovat. HTML elementy sice mají svoji výchozí vizualizaci, ale pokud ji chceme změnit, tak to uděláme právě pomocí CSS. Tento jazyk byl navržen organizací W3C v roce 1996 (CSS1).
CSS se skládá z pravidel. Každé pravidlo obsahuje selektor, například `body` pro tělo stránky nebo `.nazevTridy` pro vlastní nadefinovanou třídu. Za selektorem potom následuje blok deklarací. Deklarace má identifikátor a hodnotu vlastnosti, například atribut `width` pro šířku nebo `color` pro barvu. Hodnoty atributů mohou být v pixelech, procentech a řadě dalších typů. V přiloženém nástroji jsou kaskádové styly využívány v kombinaci s preprocesorem Sass [10] a frameworkem Tailwind [11].

3.3 Tailwind

Tento CSS framework byl v oblasti webových technologií jednou z největších novinek roku 2021 a v současnosti je jedním z nepopulárnějších UI (uživatelské rozhraní) frameworků. Jedná se o přizpůsobitelný nízko úrovňový CSS framework, který poskytuje způsob, jak dosáhnout pěkného uživatelského rozhraní bez nutnosti psát vlastní CSS. S Tailwindem [11] totiž není třeba psát názvy pro třídy (`class`) nebo pro `id`, aby jsme k nim následně mohli v samostatném souboru dopisovat pravidla. Celý trik spočívá v možnosti psát zkratky jednotlivých předdefinovaných CSS pravidel přímo v atributu `class`, a tím snížíme řádky kódu v souboru s CSS.

```
1 <!-- adding HTML and CSS -->
2 <h1 class="font-bold text-blue-800 text-2xl"> Hello </h1>
```

Zdrojový kód 4: HTML, CSS framework Tailwind

Například v ukázce zdrojového kódu 4 říkáme, že chceme v elementu `h1` mít silnější tmavě modrý text o velikosti fontu `1.5rem` (24 pixelů). Přitom nemusíme využívat jen přednastavené styly, ale snadno si lze dopsat vlastní a ty používat stejně jednoduše jako ty předdefinované. V přiloženém nástroji jde o nejčastější způsob používání (zápisu) CSS. Například vlastní šířky rámečku pro viewporty stačilo dopsat v jednom konfiguračním souboru na pár řádcích. Tailwind je tvořen plně atomickým CSS (CSS založené na jednoúčelových třídách), podobně i konkurenční Bootstrap[7] v reakci na Tailwind přidává atomické třídy.

3.4 JavaScript

Programovací jazyk JavaScript [1] vyvinul Brendan Eich v květnu 1995. Původně se jmenoval Mocha, ale známým se stal až jako JavaScript. Postupným přidáváním nových funkcí a změnami v jazyce prošel významným vývojem a je dnes nezbytný pro vytváření dynamických, interaktivních webových stránek a webových aplikací. S pomocí nástrojů (technologií) jako je Node.js, se tento jazyk rozšířil za hranice webového vývoje a začalo se s ním programovat na straně serveru. Dnes je to jeden z nejrozšířenějších programovacích jazyků s velkou aktivní komunitou vývojářů, která neustále podporuje jeho růst a rozvoj.
JavaScript [12] rozlišuje velikost písmen a příkazy jazyka jsou doporučeny ukončovat středníkem (kvůli přehlednosti). Jde o dynamicky typovaný programovací jazyk, takže se při definici proměnných neuvádí jejich typ. Proměnné musí být označeny jedinečnými názvy, protože některá klíčová slova jsou vyhrazená a musí začínat písmenem. Funkce (někdy metoda) je v JavaScriptu blok kódu určený k provádění konkrétního úkolu a je třeba ji spustit (zavolat), jako v ostatních programovacích jazycích. Většina zdrojových kódů obsažených v sekci technické dokumentace je napsaná v jazyce JavaScript (s použitím JavaScriptových knihoven React a jQuery).

3.5 jQuery

jQuery [13] je rychlá JavaScriptová knihovna fungující ve velkém množství prohlížečů. Výrazně zjednodušuje programování v JavaScriptu, jako například manipulaci s HTML dokumenty, zpracování událostí, animace a Ajax [14]. Tato knihovna byla vydána v roce 2006 a jde o svobodný otevřený software. V přiloženém nástroji se často používá k získávání kontejnerů a jiných elementů pro jejich následné nastavování různých hodnot. Například změna viditelnosti, barvy, přiřazení textu, animace operací s viewporty (na zdrojovém kódu 5) a další.

```
1  $(function () {
2      var div = $(".device" + props.idx)
3
4      // origin values
5      div.animate({ height: $(".device" + props.idx).height(),
           opacity: '0.4' }, "fast");
6      div.animate({ width: $(".device" + props.idx).width(),
           opacity: '0.6' }, "fast");
7
8      // new values
9      div.animate({ height: temp, opacity: '0.6' }, "fast");
10     div.animate({ width: $(".device" + props.idx).height(),
           opacity: '1' }, "fast");
11 });
```

Zdrojový kód 5: JS animace rotace pomocí knihovny jQuery

3.6 AJAX

AJAX (Asynchronous JavaScript and XML) [14] jsou technologie zaměřené na vývoj interaktivních webových aplikací, bez nutnosti jejich znovu načítání za pomoci asynchronního zpracování. V nástroji je AJAX využit především v kombinaci s knihovnou jQuery v podobě metody `jQuery ajax()`. Tato metoda slouží k provádění asynchronních HTTP požadavků. Pomocí této metody se odešle požadavek s url adresou (linkem) souborů s css pravidly na analyzované stránce. V případě úspěchu proběhne funkce, která ze získaných dat (odpověď serveru) v přednastaveném textovém formátu kontroluje přítomnost breakpointů a dalších položek.

3.7 JSON

JSON (JavaScript Object Notation) [15] je „sebepopisující" snadno pochopitelný textový formát pro ukládání a přenos dat. Rozšířenou alternativou k tomuto jazyku je například jazyk XML (Extensible Markup Language). JSON v přiloženém nástroji slouží k ukládání záznamů o operacích s viewporty (pro krokování) a uchování jejich základních dat. Jednoduchou úpravou jednoho souboru (s koncovkou `.json`) můžeme kdykoliv snadno přidat, nebo odebrat další viewporty. Příklad uložení dvou objektů (viewportů) se čtyřmi atributy ve formátu JSON můžeme vidět na zdrojovém kódu 6.

```
1 [ { "name": "Apple iPhone 15 Pro Max", "type": "mobile",
2     "width": "430", "height": "932" },
3   { "name": "Apple iPad Pro 2021", "type": "tablet",
4     "width": "1024", "height": "1366" } ]
```

Zdrojový kód 6: formát JSON

3.8 React

React (také React.js) [16][17] je v současné době jednou z nejrozšířenějších open-source knihoven JavaScriptu. Pochází z roku 2013 a používá se pro vytváření uživatelských rozhraní na základě komponent. Další příklady takovýchto frameworků jsou Vue, nebo Angular. React spravuje společnost Meta (dříve Facebook) a komunita vývojářů (Code of Conduct, Stack Overflow a další). React řeší neefektivitu s jakou JavaScript pracuje s DOM (Document Object Model) [18]. Bez něho by jakákoliv změna na stránce představovala renderování celého DOM. Další problém jazyka JavaScript je jeho „nízkoúrovňovost", a proto je složitější zapsat i jinak jednoduché úkony s DOM.
V principu jde na webovou aplikaci ve frameworku React nahlížet, jako na sadu UI komponent. Základem je metoda `ReactDOM.createRoot()`, která vytváří `React root DOM` element. Vytvářené elementy je možné následně zobrazit pomocí metody `render()`. Stěžejním pojmem je React komponenta, která zapouzdřuje část webové aplikace/stránky a vytváří React elementy. Z pohledu programování jsou komponenty funkce akceptující vlastnosti (`props`) a vracející výstup na nich závislý. Tyto komponenty začínají velkým písmenem a používají se ke stejnému účelu, jako funkce v JavaScriptu. Fungují přitom izolovaně a s pomocí klíčového `return` vracejí HTML elementy.
Komponenty si také mohou udržovat vlastní stav. K tomu slouží React Hooks (od verze 16). Ty umožňují komponentám přistupovat ke stavu aplikace a dalším funkcím. React Hooks můžeme volat pouze z komponent na nejvyšší úrovni a nemohou být podmíněné. Pro jejich použití se první musí naimportovat pomocí `import` například `import { useState } from "react";` Mezi nejčastěji používané funkce React Hooks patří:

- React `useState` Hook, jenž umožňuje obecně sledovat stav vlastnosti.
- React `useEffect` Hook pro operace jako je načítání dat a aktualizace DOM.
- React `useContext` Hook rozšiřující React o možnost řídit stav globálně, tedy abychom měli přístup ke stavu i z jiných komponent.
- React `useRef` Hook zachovávající hodnoty mezi renderováním.

3.9 Node.js

Node.js [19] je multiplatformní běhové prostředí JavaScriptu na enginu V8 z Google Chrome. Existují i další prohlížeče s vlastním enginem pro JavaScript. Například Firefox má SpiderMonkey a Safari používá JavaScriptCore (někdy nazývaný Nitro). Tato open-source technologie nám poskytuje běhové prostředí pro spouštění JavaScriptu mimo webový prohlížeč. Programy pro Node.js jsou psané v jazyce JavaScript. Ne vše z ECMAScript je v Node.js podporováno. JavaScript je všeobecně považován za interpretovaný jazyk, ale moderní JavaScriptové enginy již JavaScript pouze neinterpretují, ale také ho kompilují (od roku 2009). Engine V8 JavaScript interně kompiluje metodou kompilace JIT (just-in-time), která urychluje jeho provádění. Hlavní výhoda této technologie spočívá ve spojení webového a aplikačního serveru. Tím je možné dosáhnout velkého výkonu. Typicky se technologie Node.js využívá pro tvorbu (škálovatelného) webového back-endu.

3.10 ScreenCapture API

ScreenCapture API (Application Programming Interface) je technologie umožňující nahrávat videozáznamy libovolného okna na obrazovce uživatele. Společně s MediaStream Image Capture API [20] je možné z těchto videozáznamů vytvořit i snímek obrazovky. Právě pro tento účel, tedy „zachytit“ snímek obrazovky s pracovní plochou obsahující viewporty, je tato technologie využita na stránce `Viewports`.

4 NPM, přídavné balíčky a doplňky

Tato kapitola nejdříve popisuje správce balíčků NPM. S jeho pomocí bylo do přiloženého nástroje doinstalováno několik balíčků. Za částí s NPM se nachází jejich popis. Tyto balíčky umožňují v nástroji například vygenerovat report do uživatelem vybraného formátu (včetně nastavení míry komprese) nebo pořizovat snímky obrazovky. Za balíčky je ještě krátký popis doplňků pro prohlížeč.

4.1 NPM

NPM (Node.js package manager) [21] je v JavaScriptu napsaný správce balíčků pro Node.js z roku 2010. Jeho instalace probíhá automaticky společně s instalací Node.js. NPM obsahuje klienta příkazové řádky nazvaného npm a veřejnou online sbírku balíčků npm Registry, určenou především pro Node.js. Jde o alternativu pro balíčkovací systém Yarn, z roku 2016 od společností Facebook. S pomocí NPM byly do přiloženého nástroje doinstalovány všechny v této práci zmíněné balíčky. Samotné přidání balíčku do projektu je velice jednoduché. Uveďme si příklad příkazu k instalaci balíčku Sass: `npm install sass`

4.2 Sass

Jde o CSS preprocesor, to jsou obecně nástroje generující CSS (transpilace). Preprocesory nám navíc přináší funkce, proměnné, matematické výrazy, možnost vnořovat definice a další věci. Sass (Syntactically Awesome StyleSheets) [10] je jeden z nejznámějších. Další používané preprocesory jsou LESS a Stylus. V současnosti je ideální volbou použít Sass se syntaxí SCSS (soubory s koncovkou `.scss`), a právě s ní se používá i v přiloženém nástroji. Přidání Sass přesněji distribuce Dart Sass zkompilované do čistého JavaScriptu, je v nástroji formou balíčku doinstalovaného přes NPM [21].

4.3 jsPDF

jsPDF [22] je balíček pro generování PDF souborů v jazyce JavaScript. Tato HTML5 knihovna se používá na stránce `Breakpoints` při vytváření reportu do formátu PDF. Výhodou je, že při generování umožňuje nastavit míru komprese výsledného dokumentu nebo jeho orientaci.

4.4 html-to-image

html-to-image [23] je balíček umožňující generovat obrázky z DOM pomocí HTML5 canvas a SVG. Jeho použití je na stránce `Breakpoints` pro vygenerování reportu s výsledky analyzované stránky do různých formátů. Konkrétně jde o BLOP (Binary Large Object) společně s PNG, SVG a JPEG.

4.5 react-confirm-alert

Balíček react-confirm-alert [24] slouží k zobrazení okna s potvrzením, před následným provedením operace. V nástroji je využit na stránce `Viewports`, kde zobrazuje upozornění před odstraněním viewportů z pracovní oblasti.

4.6 syncfusion/ej2-base

Součástí komerční knihovny syncfusion/ej2-base [25] jsou především knihovny Syncfusion `Draggable` a `Droppable`, které uživatelům umožňují přetáhnout jakýkoli prvek DOM. Tento balíček je využit na stránce `Viewports`, kde jde s jeho pomocí provést operaci „drag and drop". Tedy „uchopit" komponentu (viewport) a poté ji přesunout (například myší) na požadované místo a tam ji nakonec „upustit".

4.7 CORS Unblock a Moesif Origin

Doplňky (rozšíření) CORS Unblock [26] i Moesif Origin poskytují každému požadavku v prohlížeči vlastní potřebné hlavičky. To nám umožňuje odesílat požadavky napříč doménami přímo z prohlížeče a zabránit tak chybám typu Cross Origin (CORS [27] Cross-Origin Resource Sharing). Uživatel může rozšíření aktivovat/deaktivovat jednoduše tlačítkem v panelu nástrojů.

5 Technická dokumentace

Soubory (komponenty) tvořící aplikaci jsou uloženy ve složce „components“ a „pages“. V této technické (programátorské) části budou postupně probrány všechny důležité soubory v obou zmíněných složkách. První je vždy popis souboru a pokud obsahuje podstatné funkce, pak následuje popis těchto funkcí. Popis funkcí je podán jednoduše, aby se dalo rychle pochopit, kdy (nebo odkud) se funkce volá, jak probíhá výpočet uvnitř a co případně následuje po jejím vykonání.

5.1 Složka components

5.1.1 AvailabilityTable.js, ValueTable.js a TableOther.js

Všechny zmíněné soubory obsahují komponenty pro tabulky s výsledky analyzované stránky. Jediná zde obsažená funkce je v `TableOther.js` a slouží pro otevření nové záložky, ve které se načte stránka `validator.w3.orgs` s předanou adresou analyzované stránky. Protože zbylé komponenty neobsahují žádné funkce, ale jen HTML elementy a kaskádové styly (hlavně Tailwind [11]), tak je nebudeme dále rozvádět.

5.1.2 Viewport.js

Tato komponenta slouží jako kontejner pro oddělené zobrazení a načtení stránky do jednotlivých viewportů. Předává se přitom několik argumentů přes `React Props`. Ty jsou předávány komponentám prostřednictvím atributů HTML podobně, jako jsou předávány argumenty pomocí funkcí v JavaScriptu. Nejčastěji je využit atribut `idx` (`props.idx`), který slouží k identifikaci více jednotlivých částí (elementů) této komponenty. Je to hlavně proto, že se na pracovní ploše může vytvořit více různých viewportů a u každého chceme měnit hodnoty jednotlivě, jinak by se hodnoty měnily jen u prvně vytvořeného. Toto se vyřeší právě pomocí dynamicky vytvářených elementů, pokaždé s jiným `id` či `className` (název třídy v Reactu).
Druhý předaný atribut je `src` (`props.src`), ten slouží pro předání uživatelem zadané url adresy, pro její následné načtení do HTML elementu `iframe` (inline frame). Poslední atribut `border` (`props.border`) je zde přítomen pro nastavení vlastní v Tailwindu nadefinované šířky rámečku pro viewporty. Podle typu zvoleného zařízení jsou tři možnosti: mobil, tablet a laptop. Zdrojový kód komponenty `Viewport` je vidět na kódu 7.

```
1  const Viewport = (props) => {
2
3    return (
4      <div className={"device" + props.idx}>
5        <iframe id={"iframe" + props.idx} title="viewportFrame"
6          className={"shadow-xl " + props.border + " border-black
               rounded-2xl"}
7          sandbox="allow-same-origin allow-scripts allow-forms
               allow-popups allow-modals"
8          loading="lazy" src={props.src}>
9        </iframe >
10     </div>
11   )
12 }
13 export default Viewport;
```

Zdrojový kód 7: HTML, CSS, JS komponenta Viewport

5.1.3 Toolbar.js

`Toolbar` je komponenta tvořící panel nástrojů u jednotlivých viewportů. Podobně jako v komponentě `Viewport`, je tady přes `props` opakovaně využíván předaný parametr `idx`. Opět jde o odlišení elementů, aby bylo možné nastavovat v každém panelu nástrojů hodnoty zvlášť. Tato komponenta zahrnuje několik funkcí pro vykonávání operací s viewporty.

Funkce komponenty **`Toolbar`**

- `rotateDevice()`

Funkce mění orientaci viewportu v pracovní oblasti z `portrait` (na výšku) na `landscape` (na šířku) a naopak. Tuto funkci spouští kliknutí na tlačítko „s obrázkem rotace“ v panelu nástrojů. Interně se spustí pomocí události (event) `onClick`. Následně ve funkci probíhá kontrola, jestli skončila poslední operace rotace, aby nedošlo k opakovanému spouštění funkce. Toto je řešeno s pomocí Hook `useState` a časovače, který na chvíli zablokuje použití tlačítka pro rotaci v panelu nástrojů. Poté jsou do formátu JSON [15] uloženy staré parametry viewportu, kvůli možnosti udělat krok zpět (nebo vpřed). Dále dochází k samotné rotaci, jak je zobrazeno na zdrojovém kódu 5 a ta se navíc s pomocí knihovny jQuery zanimuje. Po dokončení rotace se v panelu nástrojů už jen nastaví nové („prohozené“) hodnoty šířky a výšky viewportu.

- `setCustomWidth()`

Jedná se o funkci pro změnu šířky viewportu. Spouští ji každá změna příslušného zadávacího pole (element `input`, atribut `width`) v panelu nástrojů. Interně se vyvolá pomocí události `onChange`. Uvnitř se první do formátu JSON uloží staré hodnoty viewportu (pro krokování). Poté probíhá kontrola zadaného čísla na vstupu, k tomu se volá funkce `isNaN()`. Zároveň se ještě kontroluje, jestli je v rozmezí `200–2000`. Pokud je podmínka splněna, dochází k výměně hodnot za nové a s pomocí jQuery se tato operace rychle zanimuje. Nakonec se v panelu nástrojů nastaví nová hodnota šířky viewportu (v `inputu` zůstane posledně zadaná).

- `setCustomHeight()`

Funkce provádí změnu výšky viewportu. U této funkce probíhá všechno naprosto stejně, jako u funkce `setCustomWidth()`, pouze se místo změny šířky viewportu mění jeho výška (atribut `height`).

- `removeViewportById()`

Pomocí této funkce dojde k odstranění daného viewportu na základě jeho unikátního id (parametr `props.idx`). Vyvolá se pomocí události `onClick` po kliknutí na tlačítko „s křížkem" v panelu nástrojů u příslušného viewportu. Před spuštěním samotné funkce se ještě uživateli zobrazí potvrzovací zpráva. K tomu slouží doinstalovaný balíček `react-confirm-alert` [24]. Po potvrzení se uvnitř funkce první uloží hodnoty viewportu (pro krokování) a následně dojde k výběru kontejneru se souvisejícími komponentami `Viewport` a `Toolbar`. Celý tento kontejner (obsahující viewport s panelem nástrojů) je pak odstraněn pomocí předdefinované funkce `remove()`.

5.2 Složka pages

5.2.1 Viewports.js

Tato komponenta zahrnuje všechno co se nachází na stránce `Viewports`, hlavně jde o pracovní oblast v níž se pracuje s viewporty. Tvoří druhou hlavní stránku aplikace (po analyzační části na stránce `Breakpoints`), kde se vytváří a renderují dříve popisované komponenty `Viewport` a `Toolbar`. Je zde také zahrnuto několik funkcí.

Funkce komponenty (stránky) **`Viewports`**

- `checkURLParam()`

Funkce slouží pro kontrolu parametrů v url adrese, které byly předány při přesměrování uživatele ze stránky `Breakpoints` (pomocí tlačítek tabulky „Breakpoint availability"). Volání začíná v Hook `useEffect`, kde se po příchodu do komponenty (zde stránky) `Viewports` vždy kontroluje, zda jsou zadané parametry v url adrese. Pokud ano, pak se pokračuje uvnitř této funkce a z předaných parametrů se určí typ zařízení a url adresa. K určení typu se v cyklu projde obsah souboru `viewportsData.json` s daty o jednotlivých viewportech a vyhledá se první výskyt shodného typu. Poté se i se všemi ostatními atributy tato data předají jako vstup do funkce `createViewport()`. V této funkci se podle předaných parametrů vytvoří viewport (včetně panelu nástrojů) a vloží na pracovní plochu. Nakonec se zruší další průchody a funkce končí. Zdrojový kód funkce `checkURLParam()` můžeme vidět na zdrojovém kódu 8.

```
1  // By parameter in url checks if the search was executed and
       calls createViewport
2  function checkURLParam() {
3      const url = window.location.href; // Gets current URL
4      // Extracting URL parameters
5      const urlSearchParams = new URLSearchParams(new URL(url).
           searchParams);
6      let type = urlSearchParams.get('type')
7      try {
8          if (type != null) {
9              document.getElementById("url").value =
                   urlSearchParams.get('url');
10             let find = false;
11             viewportsData.forEach(data => {
12                 if (data.type == type && !find) {
13                     createViewport(data.name, data.type,
                           data.width, data.height, viewportIdx + 1,
                            false, 100, 20)
14                     find = true;
15                 }
16             });
17         }
18     } catch (error) {
19     }
20 }
```

Zdrojový kód 8: JS získání parametrů z URL adresy a průchod souboru

- `removeAllViewports()`

Funkce odstraní všechny viewporty z pracovní oblasti. Zavolá se pomocí události `onClick` po kliknutí na „reset“ v menu nad pracovní plochou. Před jejím spuštěním se vždy uživateli zobrazí varování s potvrzením, zda chce všechny viewporty odstranit. K tomu je využit balíček `react-confirm-alert` [24]. Po potvrzení dojde uvnitř funkce k výběru kontejneru, přesněji jde o sekci „sectionViewports“ obsahující všechny komponenty `Viewport` a `Toolbar` na pracovní ploše. Celý tento kontejner (se všemi viewporty a panely nástrojů) je následně vyprázdněn pomocí předdefinované funkce `empty()`. Na závěr dojde v cyklu s pomocí funkce `pop()` k odebrání všech záznamů o provedených krocích.

- `createViewport()`

`createViewport()` slouží k vytvoření viewportu, včetně panelu nástrojů a jejich vyrenderování na pracovní ploše. Jedná se o nejobsáhlejší funkci v této komponentě (na stránce `Viewports`). Nejčastěji je volána pomocí události `onChange`, při výběru viewportu v elementu `select` (dropdown - rozkliknutelný seznam položek). Ještě před vstupem do funkce je přitom ze souboru `viewportsData.json` s daty o viewportech zjištěno vše o daném viewportu a předáno jako vstupy do této funkce. Další možnost volání metody createViewport() je na základě parametrů v url, jak je zmíněno v popisu funkce `checkURLParam()`.
Uvnitř funkce nejprve dojde ke kontrole validní url adresy. Pokud je validní funkce pokračuje, jinak se zobrazí příslušné upozornění uživateli. Poté dochází k inkrementaci proměnné `idx` v Hook `useState`. To je nutné kvůli předání této hodnoty jako parametr v `props` do dynamicky vytvářených komponent `Toolbar` a `Viewport` (parametr byl již zmíněn v popisu obou komponent). Následně se předdefinovanou metodou `createElement()` vytvoří element `div` společný pro viewport i panel nástrojů, aby tyto dvě související části bylo kam umístit dohromady. Pak se vytvoří samotná komponenta `Toolbar` s předanými potřebnými parametry (viz popis `Toolbar.js`) a rovnou se vyrenderuje. Přitom se použijí v knihovně React předdefinované metody `createRoot()` a `render()`.
Potom probíhá stejný proces pro vytvoření a renderování komponenty `Viewport` (jen zobrazovací část bez panelu nástrojů). Přitom má tato komponenta předány trochu jiné parametry do `props` (viz soubor `Viewport.js`). Nakonec se obě nově vytvořené a již vyrenderované komponenty vloží do dříve vytvořeného společného kontejneru `div`, aby s nimi bylo možné například společně posouvat. To proběhne pomocí předdefinované metody `appendChild()`. Celá tato část je vidět na zdrojovém kódu 9.
Dále probíhá s pomocí jQuery získání elementů v právě vytvořeném panelu nástrojů a nastavují se jim výchozí hodnoty, dle vložených při vstupu do této funkce. Pak proběhne pomocí Hook `useContext` uložení poslední validní použité url adresy, aby se uchovala napříč aplikací a nemusela se znovu zadávat do vyhledávacího pole. Jako poslední se z doinstalovaného balíčku `@syncfusion/ej2-base` [25] vytvoří

nový konstruktor naimportované komponenty `Draggable` a je mu předán vytvořený kontejner obsahující panel nástrojů. S jeho pomocí lze s jednotlivými viewporty posouvat „drag and drop“ pomocí „uchopení“ za jejich panel nástrojů.

```
1  /* Dynamically creates and render the viewport and toolbar */
2
3  setViewportIdx(viewportIdx + 1)
4
5  // div for 1 viewport and 1 toolbar
6  const toolBarAndViewportDiv = document.createElement("div");
7
8  // creating a viewport toolbar
9  const newToolBarDiv = document.createElement("div");
10 const toolBar = ReactDOM.createRoot(newToolBarDiv);
11 const elementToolBar = <ToolBar name={name} type={type} idx={
       idParam} />;
12 toolBar.render(elementToolBar);
13
14 // add toolbar to the new created div
15 toolBarAndViewportDiv.appendChild(newToolBarDiv)
16
17 // create viewport
18 const newViewportDiv = document.createElement("div");
19 const viewport = ReactDOM.createRoot(newViewportDiv);
20
21 const elementViewport = <Viewport src={url} type={name} border={
       border} idx={idParam} />;
22 viewport.render(elementViewport);
23
24 // add viewport
25 toolBarAndViewportDiv.appendChild(newViewportDiv)
26
27 $(toolBarAndViewportDiv).prop('id', 'toolBarAndViewportDiv' +
       idParam);
28
29 // add toolbar & viewport in div into the section
30 let temp = document.getElementById('sectionViewports');
31 temp.appendChild(toolBarAndViewportDiv)
```

Zdrojový kód 9: JS vytvoření viewportu s panelem nástrojů

- `undoRedo()`

Funkce `undoRedo()` vrací viewport do stavu před posledně provedenou operací (například změna šířky) a naopak. S její pomocí můžeme také vrátit odstraněný viewport. Tato funkce je volaná pomocí události `onClick` kliknutím na tlačítka „undo", nebo „redo" v menu nad pracovní plochou. Uvnitř funkce se první určí kolik kroků zpět jsme vykonali. K tomu slouží proměnná `stepsBack` pro inkrementaci v Hook `useState`. Následně se pomocí vstupního parametru `type` určí, jestli jde o typ operace undo (zpět), nebo redo (vpřed) a případně dojde k inkrementaci proměnné `stepsBack`. Poté proběhne kontrola, zda nedošlo k přesahu kroků mimo vytvořené kroky. Tento přesah se ověří pomocí seznamu kroků uloženém ve formátu JSON [15] v proměnné `steps` s pomocí Hook `useState`.
Pokud je vše splněno, pak se pokračuje do příkazu `switch`, kde se zjistí, jaká byla poslední operace a u kterého to bylo viewportu. Na základě těchto informací se z uložených kroků v proměnné `steps` získá příslušný záznam a do pomocnýchproměnných se uloží hodnoty pro následnou operaci. Podle typu operace z tohoto záznamu (o kroku) se provede příslušná operace, tak jak je to popsáno u souvisejících funkcí v komponentě `Toolbar.js`. Zdrojový kód operací změny šířky a odstranění viewportu provedené ze záznamu je znázorněný na kódu 10.
V případě, že se jedná o záznam, kdy došlo k odstranění viewportu, pak se zavolá funkce `createViewport()`. Do této zavolané funkce se přitom ze záznamu předají všechny parametry o odstraněném viewportu a dojde k vytvoření nového se stejnými hodnotami jako měl původní. V tomto případě se ještě v cyklu projdou všechny záznamy kroků původního viewportu a přenastaví se u nich atribut `viewportId` za nové ID přiřazené nově vytvořenému viewportu. Pokud k operaci odstranění došlo až v budoucím kroku a uživatel se zrovna snaží o krok vpřed („redo"), pak se naopak viewport znovu odstraní.

```
case steps[actualStepID].action == "changeWidth":
    $(function () {
        let idx = steps[actualStepID].viewportId
        var div = $(".device" + idx)
        let width;
        if (type == -1) {
            width = steps[actualStepID].width
        } else {
            width = steps[actualStepID].newWidth
        }

        const value = document.querySelector("#valueWidth" + idx
            );

        div.animate({ width: div.width() }, "fast");
        div.animate({ width: width }, "fast");

        value.textContent = "width: " + width + "px";
        $('#width_input' + idx).prop("value", width);

        if (type == -1) {
            $("#selected-device" + idx).text(steps[actualStepID
                ].name);
        } else {
            $("#selected-device" + idx).text("customized");
        }
    });
    break;
case steps[actualStepID].action == "create" || steps[
    actualStepID].action == "delete":
    let operation = steps[actualStepID].action;

    if ((stepDirection != -1 && operation == "create") || (
        stepDirection == -1 && operation == "delete")) {
        createViewport(steps[actualStepID].name, steps[
            actualStepID].type, steps[actualStepID].width,
            steps[actualStepID].height, steps[actualStepID].
                viewportId, true, steps[actualStepID].top,
            steps[actualStepID].left);
        $("#removeAllViewports").css('opacity', 1);
    } else {
        $("#toolBarAndViewportDiv" + steps[actualStepID].
            viewportId).remove();
        if ($('[id^="toolBarAndViewportDiv"]').length == 0) {
            $("#removeAllViewports").css('opacity', 0.5);
        }
    }
    break;
```

Zdrojový kód 10: JS část operací s viewporty ze záznamu

- `addStep()`

Tato funkce slouží k ukládání záznamů o jednotlivých krocích (operacích) s viewporty. Základem je proměnná `steps` v Hook `useState`, kde se ve formátu JSON uchovávají záznamy o každém provedeném kroku. Tato metoda se volá automaticky při operacích s viewporty. Ve funkci se první ověří, zda se uživatel při provedení nové operace nepohybuje v minulosti. Pokud ano, pak dojde s pomocí předdefinované funkce `pop()` k odstranění všech záznamů (budoucích kroků), až po ten současný. Jde o to, že nová operace provedená při výskytu uživatele v minulosti, způsobila přepsaní dříve dostupných kroků vpřed (do budoucna).
Aktuální (poslední) provedená operace se stává vždy novým záznamem v seznamu kroků a současně prvním, ke kterému se lze vrátit pomocí volání funkce `undoRedo()`. Samotné uložení záznamu probíhá pomocí předdefinované funkce `push()`. Přitom jsou uloženy všechny potřebné parametry, předané z metody ve které proběhla poslední operace (například z `rotateDevice()`. Nakonec se ještě nastaví proměnná `stepsBack` v Hook `useState` na `0`, protože počet kroků zpět v ní uložený, musí být po nové operaci vynulován.

- `screenshot()`

Jedná se o metodu sloužící pro zachycení snímku obrazovky. Volá se pomocí události `onClick`, přes položku „screenshot“ v nabídce nad pracovní plochou. Pro potřebnou funkcionalitu se zde využívá technologie ScreenCapture API. Uvnitř této asynchronně probíhající funkce dojde nejdřív s pomocí Hook `useRef` k vytvoření a spuštění videozáznamu obrazovky. Proběhne zde také „schování“ vybraných věcí, které nechceme „vyfotit“.
Zmíněný videozáznam běží pouze `500 ms` a poté je ukončen. Těsně před jeho ukončením, se přitom pomocí MediaStream Image Capture API [20] vytvoří objekt `ImageCapture` a uloží se do něho jeden zachycený snímek obrazovky. K tomu je využita předdefinovaná funkce `grabFrame()`. Poté dojde k zavolání metody `handleDraw()` pro vyrenderování obrázku (z dat snímku) do předvoleného formátu PNG. Pro dočasné uložení snímku je přitom využit element `canvas` s pomocí Hook `useState` a Hook `useEffect`. Na závěr se nám hotový obrázek automaticky stáhne. Celý postup si můžeme prohlédnout na zdrojovém kódu 11.

```
const handleDraw = () => {
    let { current: ctx } = context;
    let { current: image } = imageRef;
    try {
        ctx.drawImage(image, 0, -140, window.innerWidth,
            window.innerHeight);
        setDataUrl(canvasRef.current.toDataURL("image/png"));
    } catch (e) {
        console.error(e);
    }
};

// Uses the ScreenCapture API to start a short video feed
const screenshot = async (displayMediaOptions) => {
    try {
    // If we have a reference to use (video) assign it
    videoRef.current.srcObject = await
        navigator.mediaDevices.getDisplayMedia(
        displayMediaOptions
    );
    // ImageCapture api requires a MediaStreamTrack
    const track = videoRef.current?.srcObject?.getVideoTracks()
        [0];
    if (track) {
        try {
            // Create a new ImageCapture
            let imageCapture = new ImageCapture(track);
            // Timeout used to make sure the prompt isn't caught
                in the image
            setTimeout(async () => {
                imageRef.current = await imageCapture.grabFrame
                    ();
                handleDraw();

                // Stop the video feed after
                videoRef.current?.srcObject
                    ?.getTracks()
                    ?.forEach((track) => track.stop());
            }, 500);
        } catch (e) { }
    }
    } catch (err) {
    }
};

useEffect(() => {
    context.current = canvasRef.current.getContext("2d");
}, []);
```

Zdrojový kód 11: JS snímek obrazovky pomocí ScreenCapture API

5.2.2 Breakpoints.js

Komponenta zahrnuje obsah na stránce `Breakpoints`. Probíhá tady analýza uživatelem zadané stránky zaměřená především na přítomnost breakpointů. Jde o první hlavní část aplikace. Vytváří a renderují se zde také dříve zmíněné komponenty tabulek. Následně si výsledky analýzy umístěné v tabulkách můžeme podle zvolených parametrů vygenerovat formou reportu. V komponentě `Breakpoints` se nachází několik funkcí.

Funkce komponenty (stránky) **`Breakpoints`**

- `createReport()`

Funkce `createReport()` slouží k vytvoření a stažení reportu s výsledky analyzované stránky, dle zvolených parametrů. Volá se pomocí události `onClick` po kliknutí na „download" pod menu reportu. Uvnitř funkce nejdříve dojde k vytvoření záhlaví reportu. V záhlaví je obsažen název nástroje společně s url adresou analyzované stránky a datum. Potom se zjistí první uživatelem zvolený parametr určující obsah. Ten je získán z proměnné `selectContent` v Hook `useState`. V příkazu `switch` se tak pomocí zmíněné proměnné určí vybraný kontejner s tabulkou (tabulkami) a zavolá se `setInputAndWidth()`. V této metodě se z vybraného kontejneru uloží jeho obsah do proměnné `input` pro další úpravy (zatím se nic negeneruje). Vznikajícímu reportu v proměnné `input` se zde ještě nastaví vhodná šířka a s pomocí funkce `prepend()` se přidá již vytvořené záhlaví reportu. Zpět v příkazu `switch` pak dle zvoleného obsahu může proběhnout ještě několik vizuálních úprav. Za příkazem se zjistí druhý uživatelem zvolený parametr určující formát reportu. Ten je získán z proměnné `selectFormat` opět v Hook `useState`. Potom se v dalším příkazu `switch`, podle této proměnné určí příslušná funkce pro vygenerování připraveného obsahu pro report.
Pokud jde o formát SVG, PNG, nebo JPEG, pak dojde k volání některé z funkcí `toSvg()`, `toBlob()`, případně `toJpeg()`. Všechny tyto funkce jsou naimportované z doinstalovaného balíčku `html-to-image` [23]. Při volání zmíněných funkcí se přitom vždy předává předem připravený obsah reportu v proměnné `input`. Po vygenerování reportu se zavolá `downloadReport()` a předají se jí vygenerovaná data. Uvnitř se pomocí předdefinované funkce `createElement()` vytvoří element <a>(hyperlink) a podle předaného formátu se mu nastaví název souboru. Nakonec pomocí metody `click()` dojde ke stažení do zařízení uživatele. Část funkce popsaná v tomto odstavci je na zdrojovém kódu 12. Pokud jde o formát PDF, tak první proběhne výše zmíněná funkce `toBlob()` pro vygenerování obsahu, jako binárních dat, a ta jsou po úpravě použita pro vstup do funkce `addImage()`. Funkce `addImage()` je naimportovaná z balíčku `jsPDF` [22] a slouží pro generování PDF souborů. První je nutné vytvořit nový konstruktor komponenty `jsPDF`, aby u něho šlo zavolat výše zmíněnou funkci. Nakonec se u konstruktoru pomocí funkce `save()` nastaví název výsledného souboru s reportem a ten se automaticky stáhne.

```
function downloadReport(format, data){
    var link = document.createElement('a');
    link.download = 'report.' + format;
    link.href = data;
    link.click();
}

switch (true) {
    case selectFormat == 1:
        htmlToImage.toSvg(input, { cacheBust: true, height:
            height, width: width })
            .then(function (dataUrl) {
                downloadReport("svg", dataUrl);
            });
        break;
    case selectFormat == 2:
        htmlToImage.toBlob(input, { cacheBust: true, height:
            height, width: width })
            .then(function (blob) {
                downloadReport("png", window.URL.createObjectURL
                    (blob));
            });
        break;
    // here is one more similar case hidden
    case selectFormat == 4: // PDF
        htmlToImage.toBlob(input, { cacheBust: true, height:
            height, width: width })
            .then(function (blob) {
                const doc = new jsPDF();
                var fileURL = window.URL.createObjectURL(blob);
                switch (true) {
                    case selectContent == 1:
                        doc.addImage(fileURL, 'PNG', 0, 5, 210,
                            290, '', 'FAST');
                        break;
                     case selectContent == 2:
                // here are some more similar cases hidden
                    default:
                    }
                doc.save("report.pdf"); // A4 format
            });
        break;
```

Zdrojový kód 12: JS část vygenerování a stažení reportu

- `startAnalysis()`

Funkce spouští analýzu uživatelem zadané stránky. Jde o hlavní a zároveň nejobsáhlejší funkci v komponentě (na stránce) `Breakpoints`. Volání proběhne pomocí události `onClick`, po zadání url adresy v search baru společně s kliknutím na „Find". Ještě před vstupem do funkce proběhne pomocná metoda `testCors()`, ověřující schopnost získat přístup k některým datům přístupným jen s pomocí CORS [27] a případně zobrazí upozornění.
Uvnitř funkce dojde první ke kontrole validity zadané url adresy. Pokud je validní, tak vše pokračuje, jinak se zobrazí upozornění uživateli. Poté dochází k nastavení proměnných pro uchování informací o dostupnosti breakpointů, vybraných technologií a dalších položek. Dále se pomocí metody `createHTMLDocument()`, podle zadané url adresy v searchbaru (element `input`) vytvoří nový HTML dokument.
Následně se připraví proměnná `ruleList` na získávaná CSS pravidla z právě vytvořeného dokumentu. Začíná zde běžet první (vnější) cyklus `for`, který prochází celý seznam pravidel (`Document.styleSheets`) vytvořeného dokumentu. Jednotlivé položky seznamu jsou průběžně ukládány do zmíněné proměnné `ruleList` k jejich analýze ve vnitřním cyklu.
Druhý (vnitřní) cyklus `for` už iterativně prochází jednotlivé položky seznamu pravidel (`styleSheets[i].cssRules`). Přitom některé obsahují jen `@import url()` s adresou k pravidlům, a proto se tento případ kontroluje. V případě, že se jedná o importovaná pravidla, zavolá se funkce `checkImport()` a předá se jí url adresa importovaných pravidel. Tato funkce vytvoří další dokument obsahující importovaná CSS pravidla a proběhne zde i volání funkcí pro jejich kontrolu. Tuto operaci vytvoření dokumentu s importovanými CSS pravidly můžeme vidět na zdrojovém kódu 13.
Zpět ve vnitřním cyklu potom pokračuje kontrola dostupnosti breakpointů přímo v dokumentu stránky. Přitom se vždy kontroluje, zda jde o pravidlo `@media` a zároveň za ním následuje atribut `max-width` nebo `min-width`. Dále se v těchto pravidlech s pomocí funkce `checkGridAndFlexbox()` zkontroluje přítomnost mřížkového (`grid`) systému a systému `flexbox`.

```
1  /* Create a new html document to be able to find his CSSRules */
2  let importDoc;
3  function createImportDocument(html, title) {
4      importDoc = document.implementation.createHTMLDocument(title
           )
5      importDoc.documentElement.innerHTML = html
6      return importDoc
7  }
8
9  // checking breakpoints in imported url documents
10 function checkImport(value) {
11     $.get(value, function (data) {
12         importDoc = createImportDocument(data, '
               testImportDocument');
13         let importCssData = importDoc.body.innerHTML;
14         // check the flexbox and grid
15         checkGridAndFlexbox(importCssData);
16         // if there is a rule @media
17         if (importCssData.includes("@media")) {
18             checkOtherBreakpoints(breakpoints, importCssData);
19             checkBPValue(breakpoints);
20         }
21     });
22 }
```

Zdrojový kód 13: JS vytvoření dokumentu s importovanými CSS pravidly

Poté co doběhnou oba dříve zmíněné cykly `for`, tak můžeme pokračovat do dalšího bloku kódu. V této části se v cyklu `for` volá funkce `checkLink()`, uvnitř které se s pomocí AJAX [14] prochází všechny (předané) linky dostupné na analyzované stránce. Postupně se zajde na každou url adresu a stejným způsobem jako výše se zkontroluje přítomnost breakpointů a systémů `grid` a `flexbox`. Tentokrát už to probíhá v nestrukturovaných textových datech a je třeba si u každé operace hlídat pozice v textu. K tomu slouží pomocná funkce `checkOtherBreakpoints()`, která hlídá pozici posledního breakpointu a opatrně se posouvá při hledání dalších. Pokud jsou nalezeny, tak se stejně jako v předchozím průchodu (viz odstavec výše) umístí k ostatním dříve nalezeným do sdíleného seznamu. Tato část je vidět na zdrojovém kódu 14.
Když už jsou všechna potřebná pravidla zjištěna (a uložena), proběhne extrakce hodnot breakpointů a jejich případný převod na hodnoty v pixelech. Tato část se uskuteční pomocí metody `checkBPValue()`. Přitom už jsou jednotlivé hodnoty rozlišovány a umístěny zvlášť, podle typu atributu, pro jejich následné zobrazení ve správném sloupci tabulky.

```
1  // checking breakpoints and flexbox and grid in .css files
2  function checkLink(linkUrl) {
3      $.ajax({
4          'url': linkUrl, // URL to send the request
5          'dataType': 'text', // server response data type
6          'success': function (resultData) {// if request succeeds
7
8              // check before data starts to be shorted
9              checkGridAndFlexbox(resultData);
10             // if there is a rule @media
11             if (resultData.includes("@media")) {
12                 checkOtherBreakpoints(breakpoints, resultData,
                       0);
13                 checkBPValue(breakpoints);
14             }
15         },
16         error: function () { // if request fails
17         }
18     });
19 }
```

Zdrojový kód 14: JS získání CSS pravidel z linku pomocí AJAX

Mimo práci s CSS pravidly provádí nástroj v dokumentu analyzované stránky, také kontrolu dostupnosti několika vybraných HTML elementů. Tato kontrola probíhá hned po doběhnutí posledního volání funkce `checkBPValue()`. V cyklu se prochází všechny v dokumentu dostupné HTML tagy a porovnává se u nich shoda atributu `tagName` s předvolenými elementy. Konkrétně jde o `AMP`, `IFRAME` a `FORM`. Tyto elementy jsou blíže popsány v části 6.3.3 „Other items" sekce uživatelské dokumentace.
Vytvoří se komponenty `AvailabilityTable`, `ValueTable` a `TableOther` obsahující tabulky, a ty se vyrenderují do připravené sekce. Přitom se z knihovny React použijí předdefinované metody `createRoot()` a `render()`. Následně dojde k odstranění duplicitních hodnot v roztříděných seznamech s breakpointy a zavolá se funkce `setBPValuesOnTable()`. Tato funkce postupně zobrazí předané hodnoty v příslušných buňkách tabulek. V této části se také použije funkce `markAll()`, s jejíž pomocí se v tabulce komponenty `TableOther` zobrazí i dostupnost ostatních položek. Nakonec se v Hook `useContext` uloží poslední validní použitá url, aby mohla zůstat zadaná napříč aplikací.

6 Uživatelská dokumentace

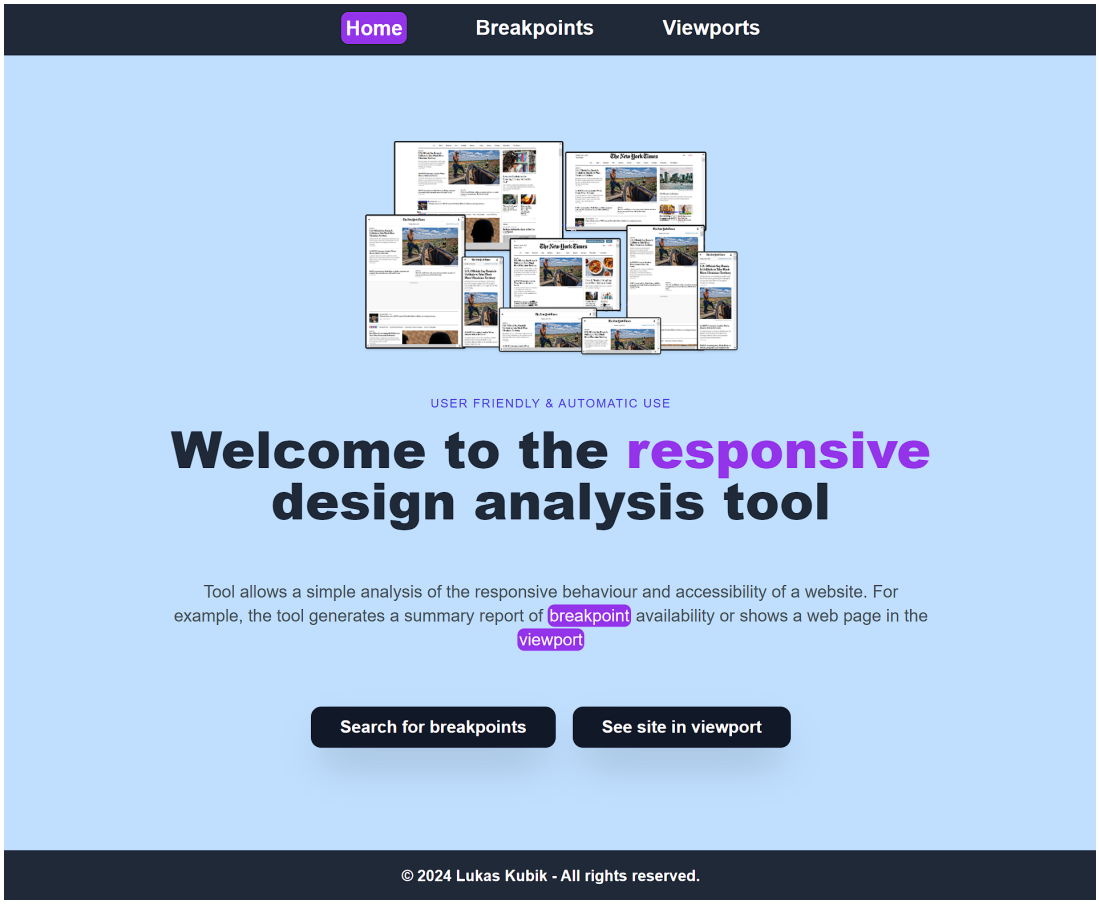
6.1 Instalace a spuštění nástroje

Ke spuštění nástroje je nutné nainstalovat technologii Node.js [19]. Jde o běhové prostředí spouštěné na straně serveru a potřebné pro používání JavaScriptových knihoven, jako je například React. Samotné spuštění nástroje pak probíhá pomocí příkazové řádky (CLI). Postup je popsán v následujících krocích.

1. Začneme tím, že si stáhneme instalátor Node.js na adrese `https://nodejs.org/en/download/prebuilt-installer` a nainstalujeme podle pokynů.
2. Po instalaci spustíme příkazovou řádku. Máme několik možností, například ve Windows 10/11 pomocí klávesy `Windows + R` spustit vyhledávání a zadat zkratku `cmd`. Také můžeme použít příkazovou řádku dostupnou v editorech, nebo IDE, jako jsou Visual Studio Code, WebStorm a Vim.
3. Uvnitř příkazové řádky se přesuneme do složky s naší aplikací. Pro přesuny můžeme využít příkaz `cd` (change directory).
4. Zadáme příkaz `npm i`. Tento příkaz zadáváme jenom před prvním spuštěním aplikace a slouží k nainstalování všech potřebných věcí využívaných v projektu. Tato instalace může chvíli trvat.
5. Jako poslední aplikaci (vždy) spustíme příkazem `npm start`, a ta se otevře v prohlížeči.

6.2 Domovská stránka

Po spuštění aplikace se otevře domovská stránka (na obrázku 2). Dále se pokračuje výběrem stránky v navigačním menu (horní okraj), nebo tlačítky pod úvodním titulkem. Pokud nebudeme počítat domovskou stránku, je nástroj rozdělený do dvou navzájem oddělených stránek. Každá z nich se přitom zaměřuje na jiný typ operací k ověření responzivity webové stránky. Jde o stránky nazvané `Breakpoints` a `Viewports` a v následující části textu budou jedna po druhé představeny z uživatelského pohledu.



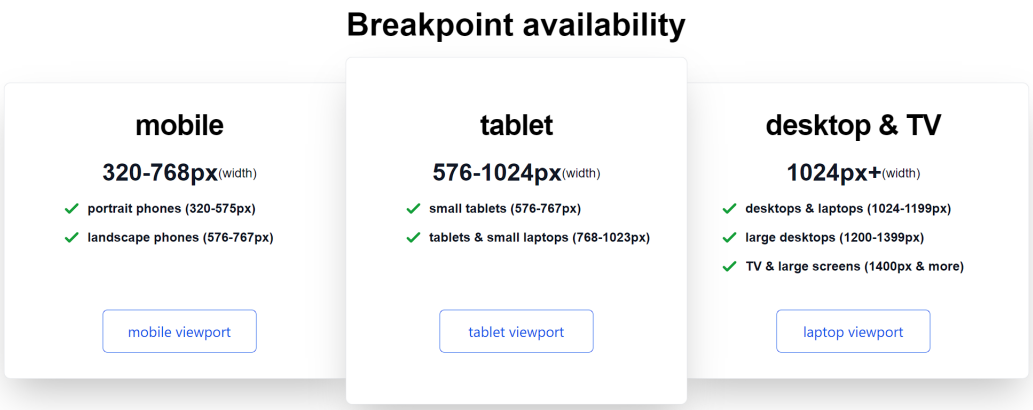
Obrázek 2: domovská stránka nástroje

6.3 Stránka Breakpoints

Stránka `Breakpoints`, jak již název napovídá, umožňuje na základě zadané webové adresy zjistit, zda jsou na ní dostupné breakpointy. Breakpointům už byla věnována část kapitoly 2.3. Po příchodu na stránku se od uživatele očekává zadání webové adresy a zahájení analýzy pomocí tlačítka „Find“. Po dokončení analýzy webové stránky nástrojem, se uživateli zobrazí menu reportu a několik tabulek obsahujících výsledky celé operace. Popis reportu teď přeskočíme, později se mu budeme věnovat v samostatné podsekci 6.3.4. Nejdříve se zaměříme na jednotlivé tabulky s výsledky.

6.3.1 Breakpoint availability

První část s výsledky nazvaná „Breakpoint availability“ obsahuje informace o dostupnosti breakpointů, vhodných pro různé typy zobrazovacích zařízení. Jako jediná je tvořena dvěma tabulkami.



Obrázek 3: první tabulka Breakpoint availability

První tabulka, jak můžeme vidět na obrázku 3, má kartové rozložení na tři díly (sloupce). Zobrazovací zařízení jsou v ní rozložena zleva doprava, od těch s nejmenší velikostí viewportu. Pokud políčko před typem zařízení obsahuje symbol „fajfky“, tak je na analyzované stránce dostupná hodnota minimálně jednoho breakpointu, vhodného pro zobrazení na tomto zařízení. Čím více je těchto symbolů zobrazeno, tím lepší je schopnost stránky přizpůsobit se různým velikostem viewportu. Stránka je tedy více responzivní. Naopak symbol „křížku“ vždy značí, že je breakpoint (hodnota breakpointu, technologie, nebo element) na analyzované stránce nedostupný. Tlačítka v dolní části tabulky nám umožňují přejít na stránku `Viewports`, kde se automaticky vytvoří daný typ Viewportu s načtenou analyzovanou stránkou.

Druhá tabulka (obrázek 4) z velké části kopíruje výsledky dostupných breakpointů v tabulce předchozí, poskytuje však přehlednější seznamové zobrazení. V její dolní části se navíc nachází zcela nové položky. Tyto položky určují, kde jsou na analyzované stránce lokalizovány breakpointy. Detailní popis průběhu lokalizace breakpointů se nachází v popisu funkce `startAnalysis()` (část 5.2.2 technické dokumentace). Testované možnosti výskytu jsou tři:

- „breakpoints in document“

Breakpointy umístěné přímo na stránce a dostupné pomocí přístupu přes dokument stránky.

- „breakpoints from file“

Velmi rozšířené připojení CSS pravidel pomocí samostatného souboru, který je jinak součástí projektu.

- „imported breakpoints“

Jde o možnost, kdy se pravidla nemusí v projektu vyskytovat a jsou naimportovaná pomocí url adresy.

Breakpoints by type of device	Availability
portrait phones (320-575px)	✓
landscape phones (576-767px)	✓
tablets & small laptops (768-1023px)	✓
desktops & laptops (1024-1199px)	✓
large desktops (1200-1399px)	✓
TV & large screens (1400px & more)	✓
Breakpoint locations	**Availability**
breakpoints in document	✗
breakpoints from file	✓
imported breakpoints	✗

Obrázek 4: druhá tabulka Breakpoint availability

6.3.2 Breakpoint values

Druhá část „Breakpoint values“ je tvořena tabulkou (na obrázku 5) s konkrétními hodnotami, všech dostupných breakpointů na analyzované stránce. Hodnoty breakpointů jsou zde rozděleny v řádcích po hodnotě 200 pixelů (sloupec „Range“) a dále dle jejich typu (atributu) do dvou sloupců.
Ve sloupci „`min-width`“ jsou hodnoty v pixelech (px) od kterých platí nějaké přizpůsobení obsahu pro zobrazovací zařízení (viewport). Jako příklad si uvedeme hodnotu 640 pixelů (`min-width: 640px`), pak je tento breakpoint (jeho hodnota) vhodná k zobrazení na zařízeních s šířkou viewportu od `640px` (kdy se aktivuje) maximálně do `1024px`. Větší šířky už by nemusely být vhodné pro zobrazený obsah a proto je potřeba přidat další breakpoint s vyšší hodnotou (jako je například zmíněných 1024 pixelů).
Sloupec „`max-width`“ naopak obsahuje hodnoty v pixelech, do kterých je aplikováno přizpůsobení obsahu stránky pro nějaké zařízení. Pro příklad můžeme opět použít hodnotu 640 pixelů (`max-width: 640px`). Takový breakpoint je vhodný k zobrazení na viewportech s šířkou maximálně `640px` (do kdy je aktivován). Jde tedy o menší zařízení jako jsou mobilní telefony. Pro lepší představu o rozšířených hodnotách breakpointů se můžeme podívat na tabulku 1.

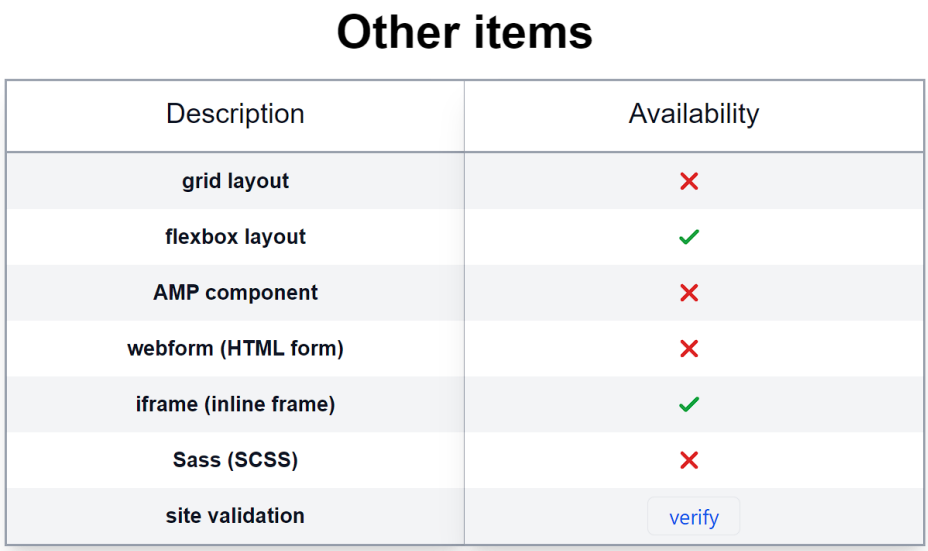
Breakpoint values

Range	min-width (values in pixels)	max-width (values in pixels)
≥300 ~ <500px	392	✕
≥500 ~ <700px	✕	600 656
≥700 ~ <900px	792 882	856
≥900 ~ <1100px	1000	999
≥1100 ~ <1300px	1294	✕
≥1300 ~ <1500px	✕	1311
≥1500px	2244 1720	✕
total	7	5

Obrázek 5: tabulka Breakpoint values

6.3.3 Other items

Třetí část pojmenovaná „Other items“ zahrnuje tabulku se všemi položkami, které přímo nesouvisí s breakpointy. Tabulku můžeme vidět na obrázku 6. Postupně si zde popíšeme jednotlivé analyzované položky.



Other items

Description	Availability
grid layout	✗
flexbox layout	✓
AMP component	✗
webform (HTML form)	✗
iframe (inline frame)	✓
Sass (SCSS)	✗
site validation	verify

Obrázek 6: tabulka Other items

- „grid layout“

CSS Grid layout [5] slouží pro tvorbu layoutu umístěného do mřížky. Je to tedy systém založený na mřížce s řádky a sloupci (2D rozvržení). Využívá se k usnadnění vytváření webových stránek. Oproti systému flexbox může být vhodnější pro komplexnější layouty.

- „flexbox layout“

Flexbox (Flexible Box Layout) [4] je zmíněný také v podsekci 2.2 o fluidních mřížkách. Nejčastěji se používá k zarovnávání a přesouvání obsahu (hlavně kontejnery). Oproti CSS Grid funguje lépe v jednom rozměru (jen s řádky, nebo sloupci).

- „AMP component“

AMP [28] byla původně zkratka „Accelerated Mobile Pages“, dnes se používá jen symbol blesku. Jedná se o technologii pro rychlé zobrazení stránky. Základní myšlenkou je „kešování“ (`AMP cache`). AMP framework má přibližně 100 komponent (pro obrázky a další), vše je pouze jednou. Přítomnost libovolné z těchto komponent (například `AMP-IMG`) určí dostupnost AMP na analyzované stránce.

- „webform (HTML form)“

Kontrola přítomnosti webového formuláře. Tyto formuláře se používají pro získávání dat od uživatele ze vstupů (textová políčka, zaškrtávací políčka a další). Uživatelský vstup je většinou následně odeslán na server ke zpracování.

- „iframe (inline frame)“

Iframe (inline frame) umožňuje vložit další dokument do aktuálního HTML dokumentu. Vzniklému rámu je možné nastavovat různé atributy, určující chování načteného obsahu uvnitř (například pro vyskakovací okna). Na stránce `Viewports` je využíván pro oddělené zobrazení načtené stránky v jednotlivých viewportech.

- „Sass (SCSS)“

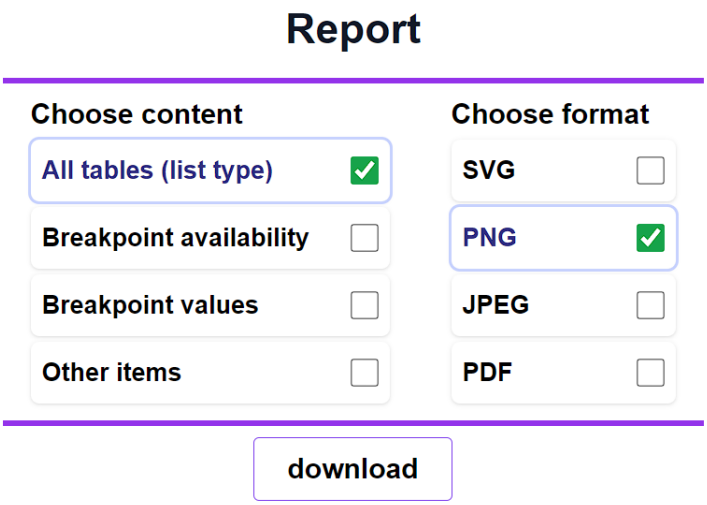
Testuje dostupnost preprocessoru Sass [10] společně se syntaxí `SCSS`. Konkrétně jde o testování přítomnosti připojených souborů s koncovkou `.scss`. Tato technologie má už svůj samostatný odstavec v podsekci 4.2. Možnosti preprocessoru Sass jsou využity i v tomto nástroji.

- „site validation“

Tato položka jako jediná ze všech zde zmíněných nemá určení dostupnosti. Místo toho obsahuje odkaz na stránky validátoru `https://validator.w3.org/`. Přitom se posledně analyzovaná stránka předává validátoru k použití jako vstup a rovnou se spustí kontrola validace.

6.3.4 Report

Všechny tabulky (typu seznam) s výsledky analyzované stránky a popsané v předchozí části, si můžeme také nechat vygenerovat do reportu. Slouží k tomu menu reportu (na obrázku 7) dostupné nad částí Breakpoint availability.



Obrázek 7: menu reportu

V levém sloupci menu vybíráme obsah. Můžeme si vybrat tabulky ze všech částí, nebo jen některou konkrétní tabulku. V pravém sloupci si následně ještě zvolíme vhodný typ formátu. Po kliknutí na „download", se pak podle zvolených parametrů, vygeneruje a uloží výsledný report. Průběh celé operace je popsán ve funkci `createReport()` v technické části dokumentace 5.2.2. Vygenerovaný report se všemi tabulkami (typu seznam) můžeme vidět na obrázku 8.

Responsive design analysis tool

http://youtube.com (2024-07-08)

Breakpoint availability

Breakpoints by type of device	Availability
portrait phones (320-575px)	✓
landscape phones (576-767px)	✓
tablets & small laptops (768-1023px)	✓
desktops & laptops (1024-1199px)	✓
large desktops (1200-1399px)	✓
TV & large screens (1400px & more)	✓
Breakpoint locations	**Availability**
breakpoints in document	✗
breakpoints from file	✓
imported breakpoints	✗

Breakpoint values

Range	min-width (values in pixels)	max-width (values in pixels)
≥300 ~ <500px	392	✗
≥500 ~ <700px	✗	600 656
≥700 ~ <900px	792 882	856
≥900 ~ <1100px	1000	999
≥1100 ~ <1300px	1294	✗
≥1300 ~ <1500px	✗	1311
≥1500px	2244 1720	✗
total	7	5

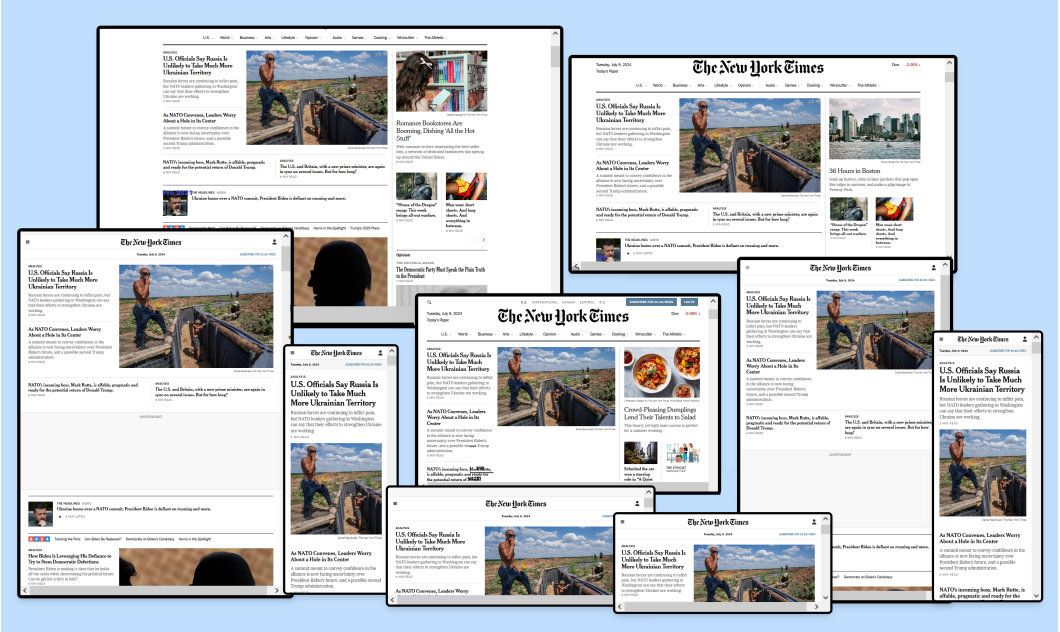
Other items

Description	Availability
grid layout	✗
flexbox layout	✓
AMP component	✗
webform (HTML form)	✗
iframe (inline frame)	✓
Sass (SCSS)	✗

Obrázek 8: vygenerovaný report

6.4 Stránka Viewports

Na stránce `Viewports` budeme testovat, jak se různé typy zařízení s rozdílnými šířkami viewportu přizpůsobují obsahu stránek (na obrázku 9). Začneme tím, že si krátce přiblížíme viewport. Viewport zahrnuje uživatelem viditelnou oblast webové stránky, bez okolních lišt a dalších částí prohlížeče. Důležité je, že má ve většině případů podstatně (podle hodnoty `DPR`[1] často 2–3x) nižší rozlišení, než je skutečné „hardwarové“ rozlišení displeje. Důvod pro snížení rozlišení je, aby byly webové stránky dobře čitelné na zařízeních s malými obrazovkami (tedy hlavně pro mobilní telefony a tablety). V nadcházejících dvou podsekcích věnovaných této stránce, si nejdříve rozebereme nabídku nad pracovní oblastí, kde se především vytváří viewporty. Následovat bude popis položek v panelu nástrojů, jenž slouží pro manipulaci a je dostupný u každého viewportu.

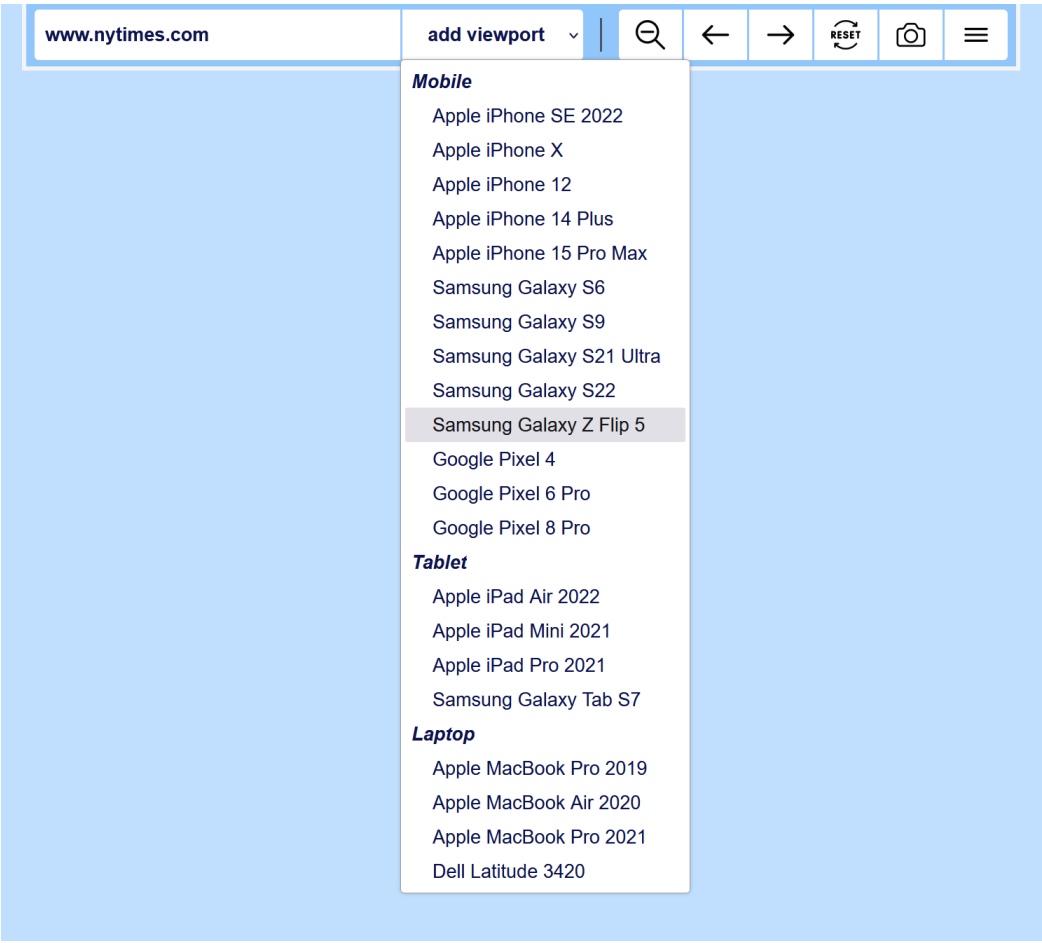


Obrázek 9: testování přizpůsobení obsahu pro různé viewporty

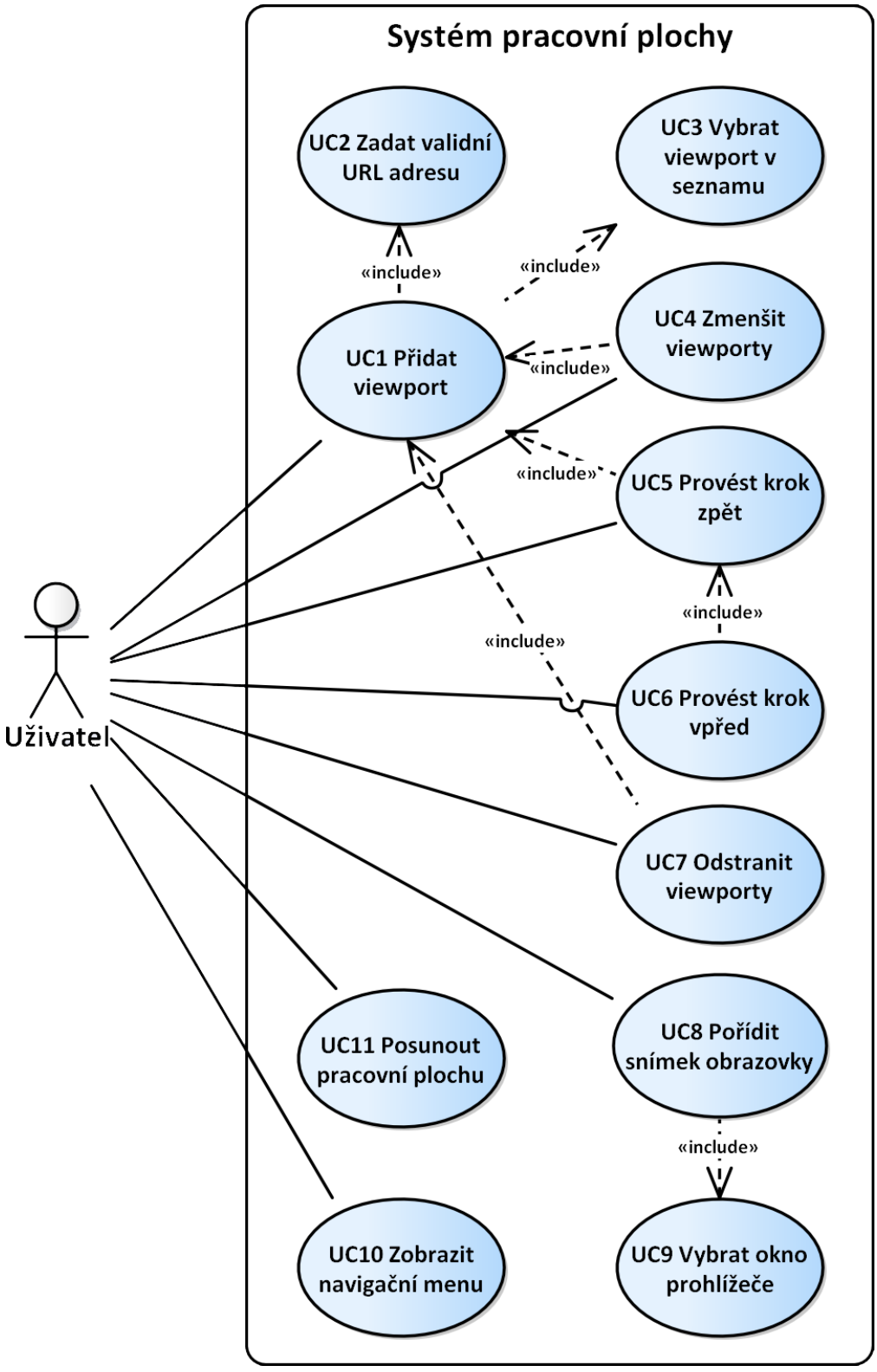
[1] `Device Pixel Ratio (DPR)` udává podíl mezi skutečným rozlišením displeje a rozlišením viewportu. Například Apple iPhone 15 má rozlišení displeje `1179px × 2556px` a `DPR 3`, proto jsou hodnoty rozlišení jeho viewportu trojnásobně nižší, konkrétně `393px × 852px`.

6.4.1 Nabídka nad pracovní plochou

Tato část se věnuje hlavně popisu dostupných interakcí v nabídce nad pracovní oblastí. Jednotlivé interakce jsou znázorněny pomocí grafického jazyka UML (Unified Modeling Language [29]) v podobě Use Case diagramu (diagram případu užití) a poté jsou krátce rozepsány konkrétní scénáře (popis jednotlivých kroků případu užití). Diagram případu užití můžeme vidět na obrázku 11. Předpokládejme, že uživatel právě přišel na stránku a má před sebou prázdnou pracovní oblast. Nejdříve bude jistě chtít přidat první viewport, proto začneme popisovat položky zleva doprava (značení případů užití vzestupně), tedy tak, jak jsou ve skutečnosti umístěny v nabídce nad pracovní plochou. Celou nabídku můžeme vidět na obrázku 10.



Obrázek 10: nabídka nad pracovní plochou



Obrázek 11: Use Case diagram interakcí pracovní plochy

- „UC1 Přidat viewport“

Tento případ užití zahrnuje levou část nabídky, kde všechno začíná. Nejdříve se uvnitř pole pro adresu zadá validní webová adresa („UC2 Zadat validní URL adresu“). Poté se pod položkou „add viewport“ otevře seznam dostupných viewportů („UC3 Vybrat viewport v seznamu“). Tento seznam je zobrazen na obrázku 10. Vybraný viewport se ihned vytvoří a na pracovní plochu se automaticky umístí na první volnou pozici zleva. Jakmile máme vytvořený viewport, zpřístupní se nám v pravé části nabídky položky „undo“ (šipka vlevo) a „reset“. Viewportů můžeme na pracovní plochu přidat libovolné množství.

- „UC4 Zmenšit viewporty“

Jde o první případ v pravé části nabídky a současně jedinou trvale aktivovatelnou položku (ikonka lupy). Ta umožňuje zmenšit všechny viewporty na 75% jejich původní velikosti (a naopak zvětšit zpět na 100%). Tato operace přitom velikost nabídky nad pracovní plochou nemění.

- „UC5 Provést krok zpět“

Případ užití označuje použití položky „undo“ (šipka vlevo), která umožňuje provést libovolný počet kroků zpět. Každý krok je nějaká provedená operace s viewporty. Například takto můžeme vrátit původní šířku nebo již odstraněný viewport s uživatelem nastavenými rozměry. První zaznamenaná operace s viewportem je jeho vytvoření („UC1 Přidat viewport“), přitom se tato položka zároveň aktivuje.

- „UC6 Provést krok vpřed“

UC6 souvisí s položkou „redo“ (šipka vpravo), která slouží k tomu, aby bylo možné provádět krokování vpřed. Této operaci vždy předchází provedení kroku zpět pomocí „undo“, jak popisuje „UC5 Provést krok zpět “. Pokud se tedy vrátíme o několik kroků zpět, tak se pomocí „redo“ zase lze posunout několik kroků vpřed, k posledně provedené operaci (nejčerstvější záznam).

- „UC7 Odstranit viewporty“

Pomocí odpovídající položky „reset“ uvedeme pracovní plochu i nabídku nad ní do výchozího stavu, jako když uživatel přišel na stránku. Tomu vždy předchází upozornění uživatele pomocí okna s potvrzením k provedení této akce. Jsou přitom smazány všechny záznamy s uloženými kroky a není možné vrácení operace.

- „UC8 Pořídit fotku obrazovky“

Případ užití souvisí s položkou „screenshot“ (ikonka fotoaparátu), ta umožňuje „vyfotit“ (zachytit) oblast pracovní plochy s viewporty. Zde se využívá technologie Screen-Capture API (společně s MediaStream Image Capture API [20]). S její pomocí dojde k výběru okna („UC9 Vybrat okno prohlížeče“) a spuštění videozáznamu obrazovky uživatele. Po velice krátkém okamžiku (půl sekundy), se přímo z tohoto záznamu pořídí snímek obrazovky. Následně se záznam automaticky ukončí a zachycený snímek je uložen do zařízení uživatele. Detailní popis průběhu celé operace i se zdrojovým kódem 11 se nachází v části 5.2.1 technické dokumentace.

- „UC10 Zobrazit navigační menu“

Tato poslední položka nabídky „menu“ umožňuje otevřít (a zavřít) výsuvné navigační menu pro přechod mezi stránkami nástroje (aplikace). Je to takto řešeno, pro maximalizaci prostoru na pracovní ploše. Jedná se o stejnou navigaci, jakou můžeme vidět na obrázku 2 domovské stránky.

- „UC11 Posunout pracovní plochu“

UC11 jako jediný nesouvisí s položkami v nabídce a slouží k posunutí celé pracovní plochy s viewporty. Tuto operaci provedeme jednoduše pomocí metody „drag and drop“.

6.4.2 Panel nástrojů viewportu

Panel nástrojů viewportu nabízí několik operací. Tyto operace tentokrát nejsou zobrazeny ve formě Use Case diagramu, protože to z pohledu jeho složitosti nebylo nutné. Místo něho jsou dostupné interakce rozebrány přímo podle přiloženého obrázku 12. Podle umístění příslušných položek v panelu nástrojů popíšeme položky opět zleva doprava.

Obrázek 12: panel nástrojů viewportu

- „drag and drop“

Umožňuje jednoduše „uchopit“ libovolný viewport v oblasti panelu nástrojů (například pomocí myši), přesunout ho po pracovní ploše a nakonec opět „upustit“ na požadovaném místě.

- „změna šířky“

Slouží ke změně šířky viewportu. Do vstupního pole (atribut `width`) je možné zadat pouze číslo, a to zároveň v rozmezí `200–2000`.

- „změna výšky“

Provede změnu výšky viewportu. Omezení vstupního pole (atribut `height`) je stejné jako u změny šířky.

- „rotace“

Pomocí položky s obrázkem rotace měníme orientaci viewportu z `portrait` (na výšku) na `landscape` (na šířku) a naopak.

- „odstranění“

Symbol „křížku“ umožňuje odstranit konkrétní viewport z pracovní plochy. Této operaci předchází upozornění uživatele. Na rozdíl od operace „reset“ ji lze vrátit pomocí položky „undo“ (nad pracovní plochou).

Tabulka 2 nám zobrazuje příklady hodnot viewportů různých typů zařízení.

Tabulka 2: Hodnoty viewportů

název zařízení	**šířka × výška (v pixelech)**	**typ zařízení**
Apple iPhone SE 2022	375 × 667	mobil
Apple iPhone X	375 × 812	mobil
Apple iPhone 12	390 × 844	mobil
Apple iPhone 14 Plus	428 × 926	mobil
Apple iPhone 15 Pro Max	430 × 932	mobil
Samsung Galaxy s9	360 × 740	mobil
Samsung Galaxy s20	360 × 800	mobil
Samsung Galaxy s22 Ultra	384 × 824	mobil
Samsung Galaxy Z Flip 5	412 × 1004	mobil
Google Pixel 4	393 × 830	mobil
Google Pixel 6 Pro	412 × 892	mobil
Google Pixel 8 Pro	448 × 998	mobil
Apple iPad Air 2022	820 × 1180	tablet
Apple iPad Mini 2021	744 × 1133	tablet
Apple iPad Pro 2021	1024 × 1366	tablet
Samsung Galaxy Tab S7	800 × 1280	tablet
Apple MacBook Air 2020	1280 × 800	laptop
Apple MacBook Pro 2021	1728 × 1117	laptop
Dell Latitude 3420 14"	1440 × 809	laptop
Chromebook Pixel	1280 × 850	laptop

7 Srovnání nástroje a existujících řešení

Většina nástrojů na analýzu responzivního chování webové stránky je zaměřená jen na jednu oblast. Výjimku tvoří pouze některé prohlížeče (jak je vidět i v tabulkách 3 a 4). Protože je tento nástroj rozdělený na dvě oddělené části, tak bude i srovnávání s existujícími nástroji probíhat odděleně.

7.1 Srovnání nástrojů na analýzu media query a breakpointů

První si srovnáme nástroje umožňující analyzovat media query a breakpointy. Takových nástrojů je málo a do srovnání se dostal jen jeden a některé webové prohlížeče. Celé srovnání můžeme vidět v tabulce 3.

Tabulka 3: Srovnání nástrojů na analýzu media query a breakpointů

název nástroje	**přítomnost media query**	**přítomnost breakpointů**	**hodnoty breakpointů**	**report**
nástroj součástí této práce	ano	ano	ano	ano
Media Query Responsive Test [30]	ano	ne	ne	ano
Mozzila Firefox Style Editor	ano	ano	ano	ne
Google Chrome DevTools	ano	ano	ano	ne
Microsoft Edge DevTools	ano	ano	ano	ne

7.2 Srovnání nástrojů zobrazujících viewporty

Jako další si srovnáme druhou část nástroje zaměřenou na práci s viewporty. Toto srovnání zahrnuje nástroje umožňující simulovat viewport (nebo více viewportů) od různých zařízení. V tabulce 4 jsou kromě názvu nástroje dostupné ještě čtyři sloupce se srovnávanými možnostmi. Druhý sloupec (zleva) „volba typu" uvádí možnost zvolit si vlastní viewport. Ve sloupci s názvem „společně" je uvedeno, kolik můžeme umístit viewportů současně („společně") například vedle sebe, nebo nad sebou. Další sloupec určuje možnost umístění viewportů a v posledním jsou uvedeny dostupné operace.

Tabulka 4: Srovnání nástrojů s viewporty

adresa (název) nástroje	**volba typu**	**společně**	**umístění**	**operace**
nástroj součástí této práce	ano	20	volitelné	změna rozměru, rotace, posun
Free Responsive Web Design Test Tool [31]	ano	1	vlevo	změna rozměru, rotace
Responsinator [32]	ne	10	nad sebou	ne
Am I Responsive? [33]	ne	4	uprostřed	ne
Website Responsive Testing Tool [34]	ano	1	vlevo	změna rozměru, rotace
Responsive Web Design Checker [35]	ano	1	uprostřed	ne
Responsive Web Design Testing Tool [36]	ne	6	nad sebou	ne
Mozzila Firefox Responsive Design Mode	ano	1	uprostřed, vlevo	změna rozměru, rotace
Google Chrome DevTools	ano	1	uprostřed	změna rozměru, rotace
Microsoft Edge DevTools	ano	1	uprostřed	změna rozměru, rotace

8 Možnosti rozšíření

8.1 Rozšíření stránky `Breakpoints`

Nástroj tvořící součást této práce je implementován tak, aby byl schopný analyzovat „jakýkoliv" element, nebo pravidlo dostupné (i přes externí linky) na analyzované stránce. To nám umožňuje jednoduše přidávat další položky k testování. Z tohoto důvodu byla na konci stránky `Breakpoints` přidána tabulka s názvem „Other items" (popis najdeme v sekci 6.3.3). Právě v ní se postupně nashromáždilo několik zajímavých položek, analyzovaných společně s breakpointy. Tuto tabulku by jistě bylo možné rozšířit o další položky a případně ji vhodně, například podle typu položky rozdělit na samostatné části.

8.2 Rozšíření stránky `Viewports`

Celá nabídka (seznam) dostupných viewportů, je před každým načtením stránky nahrána z jednoho naimportovaného souboru (ve formátu `JSON`). To nám umožňuje nabídku kdykoliv jednoduchou úpravou souboru rozšířit o nové viewporty a udržovat ji tak stále aktuální. Navíc bychom mohli například implementovat lištu okolo pracovní plochy, znázorňující šířku a výšku v pixelech, jako to mají některá existující řešení. Nebo by se mohla přidat možnost nastavovat velikost pracovní plochy. Obě zmíněné rozšíření by neměly být nijak složité na implementaci a jde už o dobrovolné doplňky.

Závěr

Původně bylo cílem této diplomové práce vytvořit nástroj pro jednoduchou analýzu responzivního chování webové stránky. Nástroj měl přitom obsahovat možnost například vygenerovat souhrnný report, a také nabídnout přehled a srovnání existujících řešení. Na základě pozdějších konzultací s vedoucím této práce, došlo na konkrétnější specifikace požadavků k analýze responzivního chování. Nástroj (aplikace) se začal úzce zaměřovat na analýzu Media Queries. Tato technika je dnes jeden ze základních kamenů responzivního webu a je důležitá hlavně proto, že zahrnuje používání breakpointů. Analýza breakpointů se tak stala samostatnou částí (stránkou) a výsledky jejich analýzy si můžeme vygenerovat ve formě reportu.
Dalším požadavkem zmíněným při konzultaci, byla možnost zobrazit zadanou stránku ve viewportu. Jakmile byla tato možnost implementovaná, přišel další rozšiřující požadavek. Jednalo se o možnost porovnání hned několika viewportů (například čtyři) vedle sebe. Řešení takové implementace už zahrnovalo oddělení práce s viewporty do samostatné části (stránky) aplikace. Tento nástroj tak jako jeden z mála (viz srovnání v tabulce 4), skutečně umožňuje porovnání několika viewportů vedle sebe, a to s viewporty, podle vlastního výběru.
Celkově pro mě byla implementace nástroje přínosná. Měl jsem možnost do hloubky prozkoumat, jak dnes funguje oblast responzivního designu, a také si zlepšit znalost zajímavých webových technologií. Jmenujme z nich například velice populární knihovnu React [16], Tailwind [11], preprocesor Sass [10] a další.

Conclusions

This master thesis originally aimed to create a tool for simple analysis of responsive behaviour of a website. For example, the tool was supposed to generate a summary report, and offer an overview and comparison of existing solutions. In later consultations with the supervisor of this thesis, there was a more specific specification of requirements concerning the analysis of responsive behaviour. The tool (application) became closely focused on the analysis of Media Queries. This technique is now one of the cornerstones of responsive web, especially because it involves the use of breakpoints. The breakpoint analysis, has become a separate part (with own page) and we can generate the results of their analysis in a report.
The next request noted during the consultation was the possibility to display the specified page in the viewport. As soon as this option was implemented, came another extending request. It was the possibility to compare multiple viewports (for example four) side by side. Solution of this implementation consisted in separating the viewports into a separate section (page) of the application. So this tool is really one of the very few (see comparison in table 4) that allows the comparison of several viewports side by side, even with viewports of your choice.
Overall, the implementation of this tool was useful for me. I had the opportunity to explore how the responsive design works today in depth and also to improve my knowledge of some interesting web technologies. To name a few, the very popular React library [16], Tailwind [11], the Sass preprocessor [10] and others.

A Obsah elektronických dat

Zde se nachází popis obsahu elektronických dat odevzdaných v systému katedry informatiky spolu s textem. Tato data jsou nedílnou součástí práce a tvoří (datovou) přílohu textu práce.

`text/`
: Adresář s textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech (textových) příloh, a všechny soubory potřebné pro bezproblémové vytvoření PDF dokumentu textu (případně v ZIP archivu), tj. zdrojový text textu a příloh, vložené obrázky, apod.

`README.txt`
: Textový soubor s informacemi o opakovatelném způsobu použití ostatních dat práce – plně reprodukovatelný funkční postup zprovoznění software vytvořeného v rámci práce, tzn. jeho případné instalace/nasazení a spuštění, včetně uvedení všech požadavků pro bezproblémový provoz;

`tool/`
: Adresář a soubory s veškerými ostatními autorskými daty práce (případně v ZIP archivu) – typicky spustitelné a další soubory software vytvořeného v rámci práce potřebné pro bezproblémový provoz software, případně jeho instalační program, a kompletní zdrojové texty software a další data nutná pro plně reprodukovatelné korektní vytvoření spustitelných souborů.

Seznam zkratek

AJAX Asynchronous JavaScript and XML

API application programming interface

BLOB binary large object

CLI Command Line Interface

CSS Cascading Style Sheets

DOM Document Object Model

DPR Device Pixel Ratio

Flexbox Flexible Box Layout

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDE integrated development environment

JIT just-in-time

JPEG Joint Photographic Experts Group

JSON JavaScript Object Notation

NPM Node Package Manager

PNG Portable Network Graphics

PDF Portable Document Format

Sass Syntactically Awesome Style Sheets

SCSS Sassy CSS

SVG Scalable Vector Graphics

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

W3C World Wide Web Consortium

XML Extensible Markup Language

Literatura

[1] ECMA-262. *Ecma International*. 2023. Dostupný také z: ⟨https://ecma-international.org/publications-and-standards/standards/ecma-262/⟩.

[2] Cascading Style Sheets. *W3C®*. 2023. Dostupný také z: ⟨https://www.w3.org/Style/CSS/⟩.

[3] Marcotte, Ethan. Responsive Web Design. *A List Apart*. 2010. Dostupný také z: ⟨https://alistapart.com/article/responsive-web-design⟩.

[4] CSS Flexible Box Layout Module Level 1. *W3C®*. 2018. Dostupný také z: ⟨https://www.w3.org/TR/css-flexbox-1/⟩.

[5] CSS Grid Layout Module Level 1. *W3C®*. 2020. Dostupný také z: ⟨https://www.w3.org/TR/css-grid-1/⟩.

[6] Media Queries Level 3. *W3C®*. 2024. Dostupný také z: ⟨https://www.w3.org/TR/mediaqueries-3/⟩.

[7] Grid system. *Bootstrap*. Dostupný také z: ⟨https://getbootstrap.com/docs/4.0/layout/grid/⟩.

[8] A dead simple, responsive boilerplate. *Skeleton*. Dostupný také z: ⟨http://getskeleton.com/⟩.

[9] HTML Living Standard. 2024. Dostupný také z: ⟨https://html.spec.whatwg.org/⟩.

[10] Sass: Syntactically Awesome Style Sheets. *Sass ©*. 2024. Dostupný také z: ⟨https://sass-lang.com/⟩.

[11] Rapidly build modern websites without ever leaving your HTML. *tailwindcss*. Dostupný také z: ⟨https://tailwindcss.com/⟩.

[12] Myers, Mark. *A Smarter Way to Learn JavaScript*. First. 2014. 256 s.

[13] What is jQuery? *OpenJS Foundation*. 2024. Dostupný také z: ⟨https://jquery.com/⟩.

[14] Garrett, Jesse James. Ajax: A New Approach to Web Applications. *Adaptive Path*. 2005. Dostupný také z: ⟨https://designftw.mit.edu/lectures/apis/ajax_adaptive_path.pdf⟩.

[15] Introducing JSON. *ECMA*. 2017. Dostupný také z: ⟨https://www.json.org/json-en.html⟩.

[16] React: The library for web and native user interfaces. *Meta Open Source*. 2024. Dostupný také z: ⟨https://react.dev/⟩.

[17] Wieruch, Robin. *The Road to React*. 2021. 340 s.

[18] DOM. 2024. Dostupný také z: ⟨https://dom.spec.whatwg.org/⟩.

[19] Run JavaScript Everywhere. *© OpenJS Foundation*. 2024. Dostupný také z: ⟨https://nodejs.org/⟩.

[20] MediaStream Image Capture API. *Mozilla Corporation*. 2023. Dostupný také z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Image_Capture_API⟩.

[21] Build amazing things. *npm, Inc.* 2024. Dostupný také z: ⟨https://www.npmjs.com/⟩.

[22] Hall, James. A library to generate PDFs in JavaScript. *npm, Inc.* 2022. Dostupný také z: ⟨https://www.npmjs.com/package/jspdf⟩.

[23] html-to-image. *npm, Inc.* 2023. Dostupný také z: ⟨https://www.npmjs.com/package/html-to-image⟩.

[24] react-confirm-alert. *npm, Inc.* 2021. Dostupný také z: ⟨https://www.npmjs.com/package/react-confirm-alert⟩.

[25] Drag and Drop in React. *Syncfusion, Inc.* 2023. Dostupný také z: ⟨https://ej2.syncfusion.com/react/documentation/common/drag-and-drop⟩.

[26] CORS Unblock. *Mozilla Corporation*. Dostupný také z: ⟨https://addons.mozilla.org/en-US/firefox/addon/cors-unblock/⟩.

[27] Cross-Origin Resource Sharing (CORS). *Mozilla Corporation*. 2024. Dostupný také z: ⟨https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS⟩.

[28] AMP is a web component framework to easily create user-first experiences for the web. *OpenJS Foundation*. 2024. Dostupný také z: ⟨https://amp.dev/⟩.

[29] Unified Modeling Language® (OMG UML®). *Object Management Group*. 2017. Dostupný také z: ⟨https://www.omg.org/spec/UML/2.5.1/PDF⟩.

[30] Media Query Responsive Test. *© SEO Site Checkup*. Dostupný také z: ⟨https://seositecheckup.com/tools/media-query-responsive-test⟩.

[31] Free Responsive Web Design Test Tool. *Designmodo Inc.* Dostupný také z: ⟨https://designmodo.com/responsive-test/⟩.

[32] Tama Pugsley, Andy Hovey. Responsinator. Dostupný také z: ⟨www.responsinator.com⟩.

[33] Am I Responsive? Dostupný také z: ⟨https://ui.dev/amiresponsive⟩.

[34] Kadudhus, Dinson. Website Responsive Testing Tool. Dostupný také z: ⟨https://responsivetesttool.com⟩.

[35] Responsive Web Design Checker. *Media Genesis, Inc.* Dostupný také z: ⟨https://www.responsivedesignchecker.com/⟩.

[36] Responsive Web Design Testing Tool. *pixeltuner.de*. Dostupný také z: ⟨http://responsive.pixeltuner.de/⟩.