UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2025 Mykhailo Onipchenko

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Webová komponenta – Tabulka

Bakalářská práce

2025 Mykhailo Onipchenko

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Mykhailo Onipchenko**
Osobní číslo:	**I21204**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Webová komponenta – Tabulka**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je navrhnout a implementovat webovou komponentu pro práci s tabulkovými daty, která bude snadno použitelná, flexibilní a výkonná.
V teoretické části bakalářské práce bude proveden úvod do tvorby webových komponent s důrazem kladeným na dostupné standardy. Dále bude v práci provedena rešerše dostupných řešení na trhu.
V aplikační části bude navržena a implementována webová komponenta pro práci s tabulkovými daty, která bude podporovat filtrování, řazení a seskupování dat, postupné načítání dat, stránkování dat, editaci dat buňky tabulky, zobrazení/skrytí sloupců tabulky. Tabulka bude optimalizována pro práci s velkým objemem dat a bude podporovat dvoucestnou vazbu k aktualizaci hodnot mezi komponentami.
K vytvořené komponentě bude zpracován popis API pro programátory.

Rozsah pracovní zprávy: **30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

FLANAGAN, David. JavaScript: the definitive guide. Seventh edition. Sebastopol: O´Reilly, 2020. ISBN 1491952024.
ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. Brno: Computer Press, 2015. ISBN 9788025145739.
BEASLEY, Michael. Practical web analytics for user experience: how analytics can help you understand your users. Amsterdam: Morgan Kaufmann, an imprint of Elsevier, 2013.
ANISOVÁ, Hana a MÜLLER, Miroslav. UML srozumitelně. 2. aktualiz. vyd. Brno: Computer Press, 2007. ISBN 8025110834.
SOMMERVILLE, Ian. Softwarové inženýrství. Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

Vedoucí bakalářské práce: **Ing. Lukáš Čegan, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. prosince 2024**
Termín odevzdání bakalářské práce: **16. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2025

Prohlašuji:

Práci s názvem Webová komponenta – Tabulka jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 21.07.2025

Mykhailo Onipchenko

PODĚKOVÁNÍ

Chtěl bych vyjádřit svou vděčnost vedoucímu mé bakalářské práce Ing. Lukáši Čeganovi, Ph.D., za jeho přínos, pomoc a rozvoj této práce. Dále bych chtěl poděkovat své rodině za možnost studovat v České republice, zejména na Univerzitě Pardubice. Také děkuji své přítelkyni a přátelům za podporu, kterou mi v průběhu tohoto období poskytovali

ANOTACE

Bakalářská práce se zaměřuje na návrh a implementaci webové komponenty pro práci s tabulkovými daty. Teoretická část analyzuje historii a architekturu webových komponent, včetně klíčových standardů (Custom Elements, Shadow DOM, HTML Templates). Práce provádí podrobnou rešerši současných tabulkových komponent dostupných na trhu (SAP UI5, Vaadin Grid, AG Grid) a zkoumá různá API pro práci s daty (REST, GraphQL, OData). Zvláštní pozornost je věnována problematice responzivity tabulek pro různá výstupní zařízení. V praktické části je navržena a implementována komponenta <my-table> využívající knihovnu LitElement. Komponenta poskytuje pokročilé funkce jako filtrování, řazení, stránkování, inline editaci, lazy loading a virtualizaci pro velké datasety. Architektura je navržena s důrazem na modularitu, rozšiřitelnost a výkon. Komponenta podporuje různé datové zdroje prostřednictvím adaptérů a zajišťuje plnou přístupnost podle standardů WCAG. Výsledkem je znovupoužitelná, framework-agnostická komponenta využitelná v moderních webových aplikacích.

KLÍČOVÁ SLOVA

webové komponenty, tabulková data, LitElement, Shadow DOM, Custom Elements, responzivní design, virtualizace, lazy loading, CRUD operace, datové adaptéry, přístupnost, web standards

TITLE

Web component – table

ANNOTATION

The bachelor's thesis focuses on the design and implementation of a web component for working with tabular data. The theoretical part analyzes the history and architecture of web components, including key standards (Custom Elements, Shadow DOM, HTML Templates). The work conducts a detailed research of current table components available on the market (SAP UI5, Vaadin Grid, AG Grid) and examines various APIs for data manipulation (REST, GraphQL, OData). Special attention is paid to the issue of table responsiveness for different output devices. In the practical part, a <my-table> component is designed and implemented

using the LitElement library. The component provides advanced features such as filtering, sorting, pagination, inline editing, lazy loading, and virtualization for large datasets. The architecture is designed with emphasis on modularity, extensibility, and performance. The component supports various data sources through adapters and ensures full accessibility according to WCAG standards. The result is a reusable, framework-agnostic component suitable for modern web applications.

KEYWORDS

web components, tabular data, LitElement, Shadow DOM, Custom Elements, responsive design, virtualization, lazy loading, CRUD operations, data adapters, accessibility, web standards

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

API – Application Programming Interface (Aplikační programové rozhraní)

ARIA – Accessible Rich Internet Applications (Přístupné bohaté internetové aplikace)

CRUD – Create, Read, Update, Delete (Vytvořit, Číst, Aktualizovat, Smazat)

CSS – Cascading Style Sheets (Kaskádové styly)

CSV – Comma-Separated Values (Hodnoty oddělené čárkou)

DOM – Document Object Model (Objektový model dokumentu)

ES – ECMAScript (Standard pro JavaScript)

GraphQL – Graph Query Language (Grafový dotazovací jazyk)

HTML – HyperText Markup Language (Hypertextový značkovací jazyk)

HTTP – HyperText Transfer Protocol (Hypertextový přenosový protokol)

JSON – JavaScript Object Notation (JavaScriptový objektový zápis)

OData – Open Data Protocol (Otevřený datový protokol)

REST – Representational State Transfer (Reprezentační přenos stavu)

SSR – Server-Side Rendering (Vykreslování na straně serveru)

UI – User Interface (Uživatelské rozhraní)

UML – Unified Modeling Language (Unifikovaný modelovací jazyk)

URL – Uniform Resource Locator (Jednotný lokátor zdrojů)

UX – User Experience (Uživatelská zkušenost)

W3C – World Wide Web Consortium (Konsorcium pro World Wide Web)

WCAG – Web Content Accessibility Guidelines (Pokyny pro přístupnost webového obsahu)

WHATWG – Web Hypertext Application Technology Working Group (Pracovní skupina pro technologie webových hypertextových aplikací)

TERMINOLOGIE

Webová komponenta – znovupoužitelný, zapouzdřený HTML element s vlastní funkcionalitou a styly, který lze použít napříč různými webovými aplikacemi nezávisle na frameworku

Shadow DOM – izolovaný DOM podstrom, který umožňuje enkapsulaci struktury a stylů komponenty tak, aby nebyly ovlivněny vnějším dokumentem

Custom Elements – webové API umožňující definovat vlastní HTML tagy a jejich chování prostřednictvím JavaScriptových tříd

LitElement – lehká základní třída pro vytváření webových komponent s reaktivními vlastnostmi a efektivním renderováním

Tabulková data – strukturovaná data organizovaná do řádků a sloupců, kde každý řádek představuje záznam a každý sloupec určitý atribut

Virtualizace – technika optimalizace výkonu, při které se renderují pouze viditelné prvky velkého datasetu

Lazy loading – postupné načítání dat nebo zdrojů až v okamžiku, kdy jsou skutečně potřebné

Responzivní design – přístup k návrhu webových stránek, který zajišťuje optimální zobrazení a použitelnost napříč různými zařízeními a velikostmi obrazovek

CRUD operace – základní operace pro manipulaci s daty: Create (vytvoření), Read (čtení), Update (aktualizace), Delete (smazání)

Datový adaptér – abstraktní vrstva, která převádí data mezi různými formáty nebo API a jednotným rozhraním komponenty

Lifecycle callbacks – metody volané v různých fázích životního cyklu webové komponenty (constructor, connectedCallback, disconnectedCallback atd.)

Event-driven architektura – návrhový vzor, kde komponenty komunikují prostřednictvím událostí místo přímých volání metod

ÚVOD

Moderní webové aplikace stále častěji pracují s velkými objemy strukturovaných dat, která je potřeba přehledně zobrazit, filtrovat, řadit a editovat. Tabulkové zobrazení dat představuje jeden z nejzákladnějších a nejpoužívanějších způsobů prezentace těchto informací uživatelům. Navzdory své zdánlivé jednoduchosti však vytvoření plně funkční, výkonné a přístupné tabulkové komponenty představuje značnou výzvu pro vývojáře.

Tradiční přístup k tvorbě tabulek často vede k těsně provázaným řešením závislým na konkrétním frameworku nebo knihovně. To komplikuje znovupoužitelnost kódu a migraci mezi technologiemi. Vývojáři jsou nuceni znovu implementovat stejnou funkcionalitu pro různé projekty nebo se spoléhat na těžké knihovny třetích stran, které nemusí plně vyhovovat jejich potřebám.

Webové komponenty, standardizované konsorciem W3C, nabízejí elegantní řešení těchto problémů. Umožňují vytvářet skutečně nezávislé, zapouzdřené a znovupoužitelné UI prvky, které fungují v jakémkoliv webovém prostředí bez ohledu na použitý framework. Díky technologiím jako Shadow DOM, Custom Elements a HTML Templates lze vytvořit komponenty s jasně definovaným rozhraním a izolovaným chováním.

Cílem této bakalářské práce je navrhnout a implementovat univerzální webovou komponentu pro práci s tabulkovými daty, která bude splňovat požadavky moderních webových aplikací. Komponenta musí zvládat zobrazení velkých datasetů, poskytovat pokročilé funkce jako filtrování, řazení a editaci, být plně responzivní a přístupná, a to vše při zachování vysokého výkonu.

Teoretická část práce se zaměřuje na analýzu současného stavu webových komponent, jejich architektury a principů fungování. Provádí rešerši existujících tabulkových řešení na trhu a identifikuje jejich silné a slabé stránky. Zvláštní pozornost je věnována problematice responzivity a různým strategiím práce s daty prostřednictvím moderních API.

Praktická část detailně popisuje návrh a implementaci komponenty <my-table> s využitím knihovny LitElement. Představuje architekturu řešení, API rozhraní, implementované funkce a optimalizační techniky. Součástí je i ukázková aplikace demonstrující možnosti komponenty v reálném použití.

Výsledkem práce je produkčně použitelná webová komponenta, která může sloužit jako základ pro tabulková řešení v různých typech webových aplikací. Komponenta je publikována jako open-source projekt a je k dispozici vývojářské komunitě pro další využití a rozvoj.

1 WEBOVÁ KOMPONENTA

Tato kapitola se věnuje návrhu a implementaci webové komponenty pro práci s tabulkovými daty. Nejprve představuje teoretické základy samotné tvorby webových komponent — včetně jejich historie, principů fungování a klíčových standardů (Custom Elements, Shadow DOM, HTML Templates). Následně popisuje, jaké principy návrhu vedou k tomu, aby komponenta byla co nejjednodušší na použití, zároveň dostatečně flexibilní a zároveň výkonná. V závěru kapitoly je provedena rešerše dostupných řešení na trhu, ve které je navrhovaný přístup porovnán s podobnými knihovnami či frameworky a rozebrány jejich silné i slabé stránky.

1.1 Historie vzniku webových komponent

První pokusy o rozšíření HTML o opakovaně použitelné komponenty sahají do roku 1998, kdy Microsoft představil HTML Components (HTC) pro Internet Explorer. Toto řešení však fungovalo pouze v IE 5.5+ a vyžadovalo použití JScriptu či VBScriptu, což komplikovalo vývoj a údržbu aplikací. HTC bylo nakonec opuštěno a v IE 10 byly tyto prvky odstraněny. [1]

Na počátku 2000. let Mozilla pracovala na XUL (XML User Interface Language) a XBL (XML Binding Language) pro prohlížeče Netscape a později Firefox. Tyto technologie umožňovaly vytvářet izolované widgety s vlastním chováním a styly, ale nebyly standardizovány a nepodporovaly je ostatní prohlížeče. [2]

Pravý obrat nastal v roce 2011, kdy Alex Russell na konferenci Fronteers představil koncept Shadow DOM jako vývojářského API pro izolaci vnitřní struktury a stylů komponent. Tento impuls spustil spolupráci WHATWG a W3C na definici univerzálních rozhraní pro komponenty.[2]

Roku 2012 vydala W3C pracovní koncept „Components“ (WD-components-intro), který formalizoval tři klíčové stavební bloky Web Components: Custom Elements pro definici vlastních HTML prvků, Shadow DOM pro izolaci DOM a CSS a HTML Templates pro deklarativní definici znovupoužitelných šablon. [3]

V květnu 2015 Google oficiálně vydal knihovnu Polymer 1.0, která poskytla polyfilly a syntaktický cukr pro snadné vytváření a kombinování Web Components napříč prohlížeči. [4]

Během následujících let prohlížečové implementace postupně dorůstaly do nativní podpory Web Components, což umožnilo jejich široké využití v moderních aplikacích bez potřeby externích knihoven. [2]

1.2 Architektura webových komponent

Webové komponenty představují nativní, standardizovaný model pro tvorbu izolovaných, znovupoužitelných UI prvků v prohlížeči. Architektura Web Components se skládá ze šesti vzájemně propojených principů, které zaručují konzistentní chování, enkapsulaci a interoperabilitu bez závislosti na externích frameworkech[5]:

- Jednotné API (Uniform API) - Všechny komponenty se definují jednotným způsobem voláním a implementují sadu lifecycle callbacků (constructor, connectedCallback, attributeChangedCallback, disconnectedCallback). Tento jednotný přístup usnadňuje naučit se jakoukoliv novou komponentu a okamžitě aplikovat stejné metody i v jiných projektech

```
customElements.define('my-element', MyElementClass);
```

Příklad 1: Způsob volání web komponenty

- Enkapsulace (Shadow DOM) - Pomocí this.attachShadow*({ mode: 'open' }*) se vytvoří izolovaný podstrom DOM a scope-ované styly, které jsou skryté před vnějším dokumentem. Díky tomu vnější CSS nemůže nechtěně ovlivnit vnitřní strukturu komponenty a naopak. [5]
- Deklarativní šablony & sloty (HTML Templates & Slots) - <*template*> definuje neaktivní HTML, které se klonuje až při inicializaci komponenty, zatímco <*slot*> označuje místa, kam se může vkládat obsah z hostitelského DOM. To umožňuje čisté oddělení šablony od runtime a podporuje flexibilní kompozici prvků. [5]
- Životní cyklus (Lifecycle callbacks)

  Specifikace nabízí čtyři hlavní callbacky:

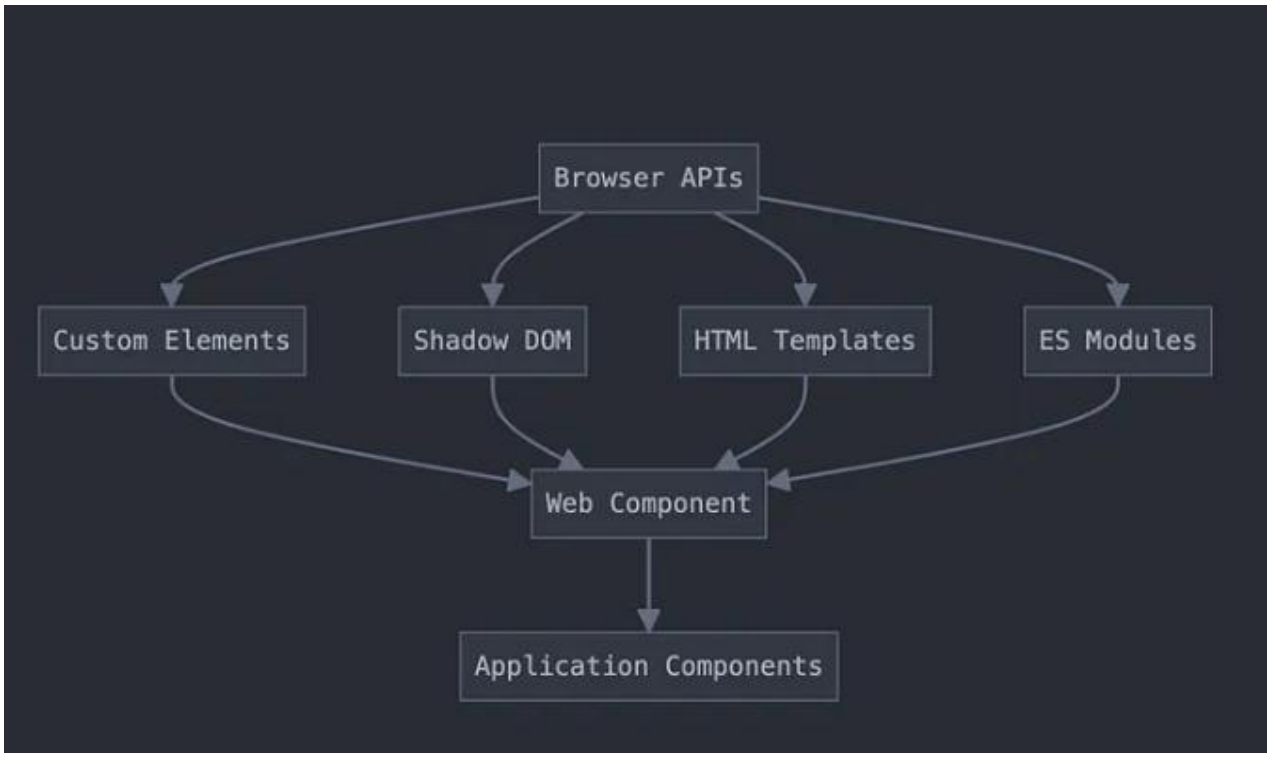
  - `constructor()` pro základní inicializaci (vytvoření Shadow DOM),
  - `connectedCallback()` spouštěný po připojení do dokumentu,
  - `attributeChangedCallback(name, old, new)` pro reaktivní změny atributů,
  - `disconnectedCallback()` pro úklid zdrojů při odstranění.

Díky nim komponenta spolehlivě řídí svůj stav bez nutnosti externích eventů

- Kompozice a komunikace (Composition & Events) - Jednotlivé komponenty lze vkládat do sebe (nested components) a předávat jim data prostřednictvím vlastností nebo HTML atributů. Pro notifikace změn se používají CustomEvent, které hostitelská aplikace snadno zachytí a zpracuje. [5]
- Modularita a lazy loading (ES Modules & Performance) - Každá komponenta je dodávána jako samostatný ES modul *(<script type="module">*), což umožňuje dynamický import (*import()*) a načítání kódu až ve chvíli potřeby. Tím se minimalizuje počáteční zátěž a zrychluje se render klíčové části stránky. [5]

Klíčové stavební bloky:

- **Custom Elements** – Definice vlastního tagu rozšířením *HTMLElement* a registrací přes *customElements.define.*
- **Shadow DOM** – Izolovaný podstrom DOM a scoped CSS, vytvořený voláním attachShadow*({mode: 'open'}).*
- **HTML Templates** - *<template>* pro deklarativní markup, klonovaný při instanciaci.
- **Slots** - *<slot>* pro vkládání externího obsahu z hostitelského DOM.
- **Lifecycle callbacks** - *constructor, connectedCallback, attributeChangedCallback, disconnectedCallback.*
- **ES Modules** - Samostatné moduly s podporou dynamického importu (*import ()*) a lazy-loadingu.



Obrázek 1 – Schéma webové komponenty[39]

Pracovní tok vývoje komponenty:

1. Definice třídy:

```
class MyElement extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open' });
    // inicializace stavu
  }
  connectedCallback() { /* render, přidání event listenerů */ }
  static get observedAttributes() { return ['foo', 'bar']; }
  attributeChangedCallback(name, oldVal, newVal) { /* reakce na změny */ }
  disconnectedCallback() { /* odstranění listenerů, uvolnění zdrojů */ }
}
```

Ukázka kódu 1: Definice třídy

2. Registrace

```
customElements.define('my-element', MyElement);
```

Příklad 2: Registrace web komponenty

3. Inicializace

```
<my-element foo="123">
<span slot="header">Nadpis</span>
</my-element>
```

1.3 Rešerše současných implementací tabulkových komponent

Kromě porovnání vizuálních a funkčních možností jednotlivých knihoven je důležité sledovat také jejich podporu a komunitní aktivitu, která ovlivňuje budoucí udržitelnost řešení.

1.3.1 SAP UI5 Web Components Table

Popis: Robustní tabulková komponenta založená na UI5 design systému, určená pro enterprise aplikace.

Klíčové vlastnosti:

- Předdefinované sloupce s možností sortování a filtrování.
- Podpora různých typů dat (text, čísla, datum, checkbox).
- Integrovaná podpora pro multi-select a single-select řádků.
- Responsivní design s možností skrytí sloupců na menších obrazovkách.
- Podpora pro gruppování dat podle sloupců.
- Integrovaná paginace a virtualizace pro velké datové sady.

Dekompozice komponent:

ui5-table
- ui5-table-header-row
  - ui5-table-header-cell
  - ui5-table-column-menu
- ui5-table-row
  - ui5-table-cell
  - ui5-table-row-action
- ui5-table-group-row
- ui5-busy-indicator
- ui5-pagination

Výhody:

- Vysoká úroveň přístupnosti (WCAG 2.1 AAA).
- Konzistentní design systém.
- Robustní API pro enterprise použití.
- Podpora pro různé datové formáty.

Nevýhody:

- Velká velikost bundle (>200 kB).
- Složitá konfigurace pro jednoduché use cases.

- Závislost na celém UI5 ekosystému.
- Omezená customizace vzhledu.

1.3.2 Vaadin Grid

Popis: Výkonná tabulková komponenta s podporou pro miliony řádků dat díky virtualizaci.

Klíčové vlastnosti:

- Virtualizace řádků a sloupců pro optimal výkon.
- Inline editace s podporou pro validaci.
- Podpora pro tree-data struktury.
- Drag & drop pro řádky a sloupce.
- Integrované filtrování a sorting.
- Responsivní design s možností definice breakpointů.

Dekompozice komponent:

vaadin-grid
- vaadin-grid-column-group
  - vaadin-grid-column
- vaadin-grid-filter
- vaadin-grid-sorter
- vaadin-grid-tree-toggle
- vaadin-grid-selection-column
- vaadin-grid-cell-content

Výhody:

- Excelentní výkon i s velkými datovými sadami.
- Flexible API pro customizaci.
- Podpora pro complex data types.
- Dobrá dokumentace a community.

Nevýhody:

- Komerční licence pro pokročilé funkce.
- Složitější learning curve.
- Závislost na Vaadin frameworku.

Nevýhody:

- Komerční licence pro pokročilé funkce.
- Složitější learning curve.
- Závislost na Vaadin frameworku.

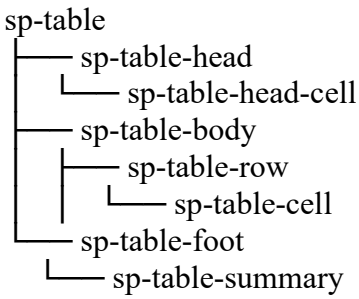
1.3.3 Spectrum <sp-table>

Popis: Lehká, framework-agnostická tabulková webová komponenta z knihovny Spectrum Web Components (Adobe). Je implementována jako Custom Element založený na LitElement a respektuje designový systém Adobe Spectrum. Zaměřuje se na konzistenci UI, přístupnost a snadnou integraci do moderních webových aplikací.

Klíčové vlastnosti:

- Implementace jako <sp-table> s podporou Shadow DOM.
- Možnost změny hustoty zobrazení (compact, spacious).
- Konzistence se stylem Adobe Spectrum (barevnost, typografie, interakce).
- Jednoduché API s využitím dalších komponent rodiny Spectrum (např. řádky, buňky).
- Podpora interaktivního řazení a práce s velkými datasety.

Dekompozice kompnent:



Výhody:

- Plná integrace s designovým systémem Adobe Spectrum.
- Bez závislostí na konkrétním frameworku (funguje čistě jako Web Component).
- Možnost jednotného vzhledu v kombinaci s ostatními Spectrum komponentami.
- Podpora responzivity a přístupnosti.

Nevýhody:

- Omezená dokumentace a menší množství ukázkových scénářů.
- Pokročilé funkce (např. filtrování, stránkování) nejsou přímo součástí jádra.
- Vyšší provázanost s vizuálním stylem Adobe Spectrum, což omezuje využití mimo tento ekosystém.

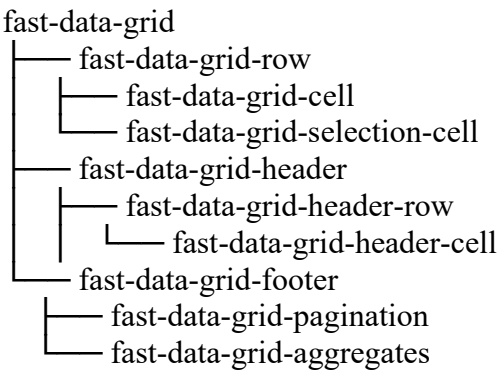
1.3.4 FAST Data Grid

Popis: Tabulková web komponenta z Microsoft FAST knihovny. Poskytuje vysoký výkon díky virtualizaci a optimalizovanému renderování, přičemž zůstává plně framework-agnostická.

Klíčové vlastnosti:

- Nativní Web Components s podporou Shadow DOM.
- Virtualizace řádků a sloupců.
- Customizace přes CSS a design tokens.
- Integrace s FAST data bindingem nebo libovolným zdrojem dat.
- Přístupnost dle WCAG standardů.

Dekompozice komponent:



Výhody:

- Optimalizace pro výkon a nízkou latenci.
- Dobrá integrace do enterprise prostředí.
- Silná podpora od Microsoftu.

Nevýhody:

- Méně pokročilých vestavěných funkcí než u specializovaných tabulek.
- Menší počet komunitních pluginů.

1.3.5 Smart Table (UI5 Web Components)

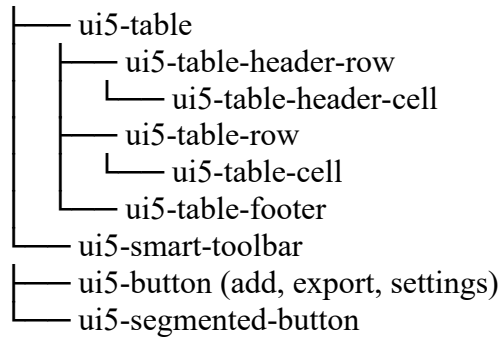
Popis: Pokročilá tabulková web komponenta z rodiny UI5 Web Components, navržená pro dynamické generování tabulek na základě metadat. Umožňuje automatické vytváření sloupců a jejich konfiguraci přímo z datového modelu.

Klíčové vlastnosti:

- Implementace pomocí Custom Elements a Shadow DOM.
- Automatické generování sloupců z metadat.
- Vestavěné funkce: řazení, filtrování, stránkování.
- Možnost rozšíření o vlastní šablony buněk.
- Integrace s OData a REST API.
- Podpora responzivního designu a přístupnosti.

Dekompozice komponent:

ui5-smart-table



Výhody:

- Minimální konfigurace díky automatickému generování sloupců.
- Silná integrace s enterprise back-endy (OData, SAP systémy).
- Konzistentní UI5 design systém.

Nevýhody:

- Větší závislost na UI5 ekosystému.
- Méně flexibilní pro velmi specifické vizualizační scénáře.

1.4 Analýza API pro práci s daty

1.4.1 REST API

REST (Representational State Transfer) je nejrozšířenější přístup pro práci s daty v tabulkách. Klíčové vlastnosti:

Standardní operace:

- GET /api/items - získání seznamu
- GET /api/items/{id} - detail položky
- POST /api/items - vytvoření nové položky
- PUT /api/items/{id} - aktualizace položky
- DELETE /api/items/{id} - smazání položky

Parametry pro tabulky:

```
GET /api/users?
  page=1&
  size=20&
  sort=name,asc&
  filter[status]=active&
  filter[age][gte]=18
```

1.4.2 GraphQL

GraphQL poskytuje flexibilnější přístup k datům s možností specifikovat přesně požadovaná pole:

```
query GetTableData($page: Int!, $size: Int!, $sort: SortInput, $filter:
FilterInput) {
```

```
  users(page: $page, size: $size, sort: $sort, filter: $filter) {
    items {
      id
      name
      email
      status
    }
    pageInfo {
      total
      hasNextPage
      hasPreviousPage
    }
  }
}
```

1.4.3 OData

OData (Open Data Protocol) je standardizovaný protokol pro vytváření a konzumaci RESTful API:

Query options:

- $filter - filtrování dat
- $orderby - řazení
- $top a $skip - stránkování
- $expand - načtení related entit
- $select - výběr polí
- $count - celkový počet záznamů

Příklad:

```
GET /odata/Products?
  $filter=Price gt 20&
  $orderby=Name desc&
  $top=10&
  $skip=20&
  $select=Name,Price,Category&
  $expand=Category
```

1.4.4 WebSocket pro real-time data

Pro aplikace vyžadující real-time aktualizace dat:

```
const ws = new WebSocket('wss://api.example.com/table-updates');

ws.onmessage = (event) => {
  const update = JSON.parse(event.data);
  switch(update.type) {
    case 'ROW_ADDED':
      table.appendRow(update.data);
      break;
    case 'ROW_UPDATED':
      table.updateRow(update.id, update.data);
      break;
    case 'ROW_DELETED':
      table.deleteRow(update.id);
```

```
        break;
    }
};
```

1.5 Problematika responzivity tabulek

U responzivních tabulek je nutné řešit, jak budou jednotlivé sloupce a ovládací prvky zobrazeny na zařízeních s různými velikostmi obrazovek. Správná implementace zvyšuje použitelnost na mobilních telefonech a tabletech, aniž by došlo ke ztrátě důležitých informací.

1.5.1 Výzvy responzivního designu tabulek

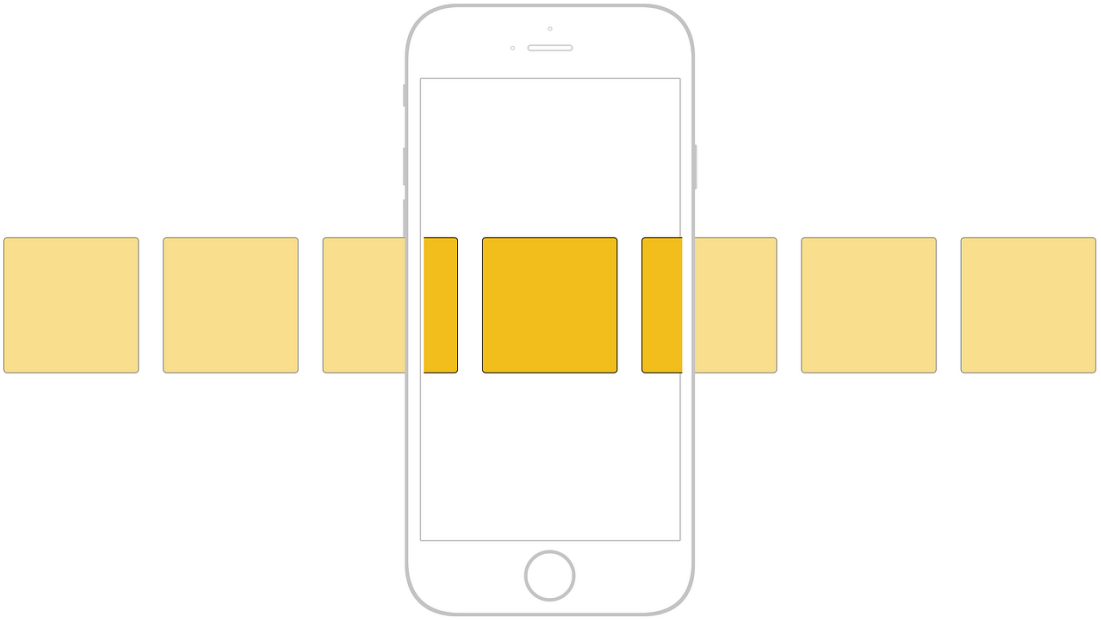
Tabulky představují jedinečnou výzvu pro responzivní design. Jejich rigidní struktura založená na řádcích a sloupcích se obtížně přizpůsobuje malým obrazovkám mobilních zařízení.

Hlavní problémy:

- Horizontální prostor – tabulka s 10 sloupci vyžaduje minimálně 1000px šířky, ale mobil nabízí pouze 375-414px.
- Čitelnost – při proporcionálním zmenšení se text stává nečitelným, zvětšení písma zase rozbíjí layout.
- Interaktivita – dotykové ovládání vyžaduje větší cílové oblasti (min. 44x44px) než ovládání myší.
- Výkon – mobilní zařízení mají omezený výkon pro renderování velkých DOM struktur.

1.5.2 Strategie responzivního zobrazení

1.**Horizontální scrolling**



Obrázek 2 – Příklad horizontálního scrolligu[40]

Nejjednodušší řešení – zachování původní tabulky s možností horizontálního posuvu. Vhodné pro tabulky, kde jsou důležité vztahy mezi všemi sloupci.

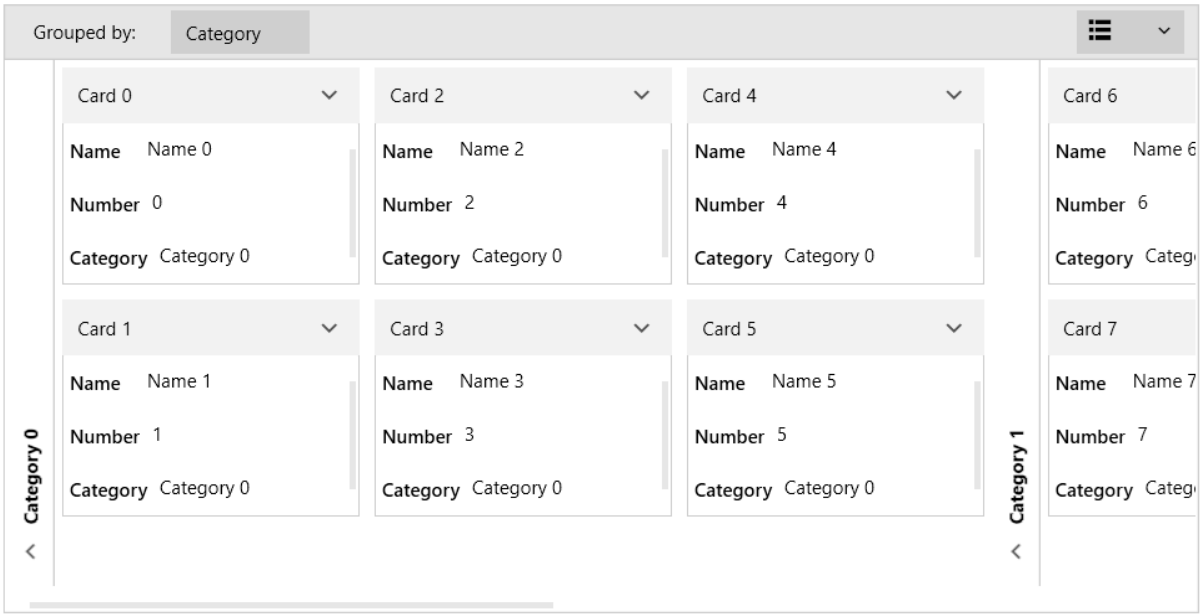
```
.table-container {
  overflow-x: auto;
  -webkit-overflow-scrolling: touch;
}
```

2. **Prioritizace sloupců**

Skrytí méně důležitých sloupců na malých obrazovkách. Vyžaduje analýzu, které informace jsou kritické.

```
const priorityColumns = ['name', 'status', 'action'];
const visibleColumns = window.innerWidth < 768
  ? columns.filter(col => priorityColumns.includes(col.key))
  : columns;
```

3. **Kartový layout**



Obrázek 3 – Přiklad kartového layoutu[41]

Transformace každého řádku na samostatnou kartu. Data se reorganizují do vertikálního layoutu, což lépe využívá dostupný prostor.

```
@media (max-width: 768px) {
  .table-row {
    display: block;
    border: 1px solid #ddd;
    margin-bottom: 10px;
    padding: 15px;
  }
}
```

4. Reflow tabulky

Každá buňka se zobrazí jako samostatný řádek s popiskem. Zachovává sémantiku tabulky

při změně vizuální prezentace.

1.5.3 Breakpointy a adaptivní chování

Správná volba breakpointů je klíčová pro plynulý přechod mezi různými režimy zobrazení:

- Mobile (<480px) - kartový layout nebo reflow
- Tablet (481-768px) - prioritizace sloupců + horizontální scroll
- Desktop (>768px) - plná tabulka

```
const getViewportStrategy = (width) => {
  if (width <= 480) return 'cards';
  if (width <= 768) return 'priority';
  return 'full';
};
```

1.5.5 Performance optimalizace

Pro zajištění plynulého chodu na mobilních zařízeních:

- Virtualizace – renderování pouze viditelných řádků.
- Lazy loading – postupné načítání dat při scrollování.
- Debouncing – omezení frekvence náročných operací (filtrování, vyhledávání).
- CSS transforms – použití GPU akcelerovaných animací.

1.6 Zhodnocení přínosů webových komponent

Webové komponenty představují nativní standardizovaný model pro tvorbu izolovaných, znovupoužitelných a interoperabilních UI prvků přímo v prohlížeči. Na rozdíl od frameworkově vázaných knihoven (React, Angular, Vue) Web Components využívají vestavěná API – Custom Elements, Shadow DOM a HTML Templates – která jsou podporována napříč všemi moderními prohlížeči bez nutnosti externích runtime knihoven. [5]

Dlouhodobá stabilita a nezávislost – Standardizace Web Components pod WHATWG/W3C zaručuje zpětnou kompatibilitu a dlouhodobou životnost. Komponenty definované nativně fungují i po letech bez nutnosti migrace na novou verzi frameworku či překódování existujících prvků. [6]

Silná enkapsulace (Shadow DOM) - Díky *attachShadow({ mode: 'open' })* vytváří každá komponenta vlastní izolovaný podstrom DOM i scope-ované styly, které chrání její interní strukturu před vlivy z vnějšího dokumentu a zabraňují kolizím CSS tříd. [5]

Deklarativní šablony a flexibilní kompozice – Šablony *(<template>)* a sloty *(<slot>)* umožňují definovat markup, který se klonuje až při inicializaci komponenty, a zároveň umožňují hostitelské stránce vkládat vlastní obsah na konkrétní místa v šabloně. To

odděluje definici struktury od jejího využití a podporuje složení i vnořené komponenty. [5]

Reusabilita a interoperabilita – Jednou vytvořenou komponentu lze zcela nezávisle znovu použít napříč různými projekty bez úprav, a to i vedle jiných frameworků. *Například* <*user-card*> implementovaný jako Web Component lze bez změn vložit jak do "vanilkového" JavaScriptu, tak do React či Angular aplikace. [7]

Modularita a microfrontendové nasazení – Díky tomu, že každá komponenta žije ve vlastním ES6 modulu (<*script type="module"*>), je možné ji dynamicky importovat (*import()*) až ve chvíli potřeby. Tento přístup zpřístupňuje lazy loading, snižuje počáteční velikost bundle a zrychluje vykreslení stránky. [8]

Theming a přizpůsobení vzhledu – Webové komponenty podporují CSS custom properties (proměnné) a CSS Shadow Parts (*::part()*), které umožňují hostitelské aplikaci měnit barvy, rozestupy či fonty uvnitř Shadow DOMu, aniž by se zasahovalo do interní implementace komponenty. [9]

Přístupnost (Accessibility) - Správné využití WAI-ARIA rolí a atributů v kombinaci s ElementInternals API zajistí, že komponenty budou přístupné pro uživatele se speciálními potřebami. Semantické role, *tabindex* i *aria-** atributy lze definovat uvnitř komponenty a zároveň je přiřadit hostitelské aplikaci. [10]

Testovatelnost a kvalita kódu – Isolovaný Shadow DOM umožňuje snadné psaní unit testů (např. pomocí *@open-wc/testing*) a snapshotů struktury komponenty, aniž by docházelo k vedlejším efektům v globálním DOM. End-to-end testy (Playwright, Cypress) pak ověří UX scénáře a správnou interakci komponenty s uživatelem.

Díky těmto vlastnostem — **stabilitě**, **enkapsulaci**, **reusabilitě**, **modularitě**, **themingu**, **optimálnímu výkonu**, **přístupnosti** a **testovatelnosti** — poskytují Web Components robustní základ pro udržitelné budování moderních webových aplikací.

1.7 Nevýhody architektury webových komponent

Navzdory řadě výhod představují webové komponenty také několik omezení a úskalí, která je dobré zvážit před jejich nasazením:

Nízká úroveň API a zvýšená komplexita -- webové komponenty byly původně navrženy jako stavební blok pro framework-vývojáře, ne jako „hotové" UI komponenty. Jejich API (Custom Elements, Shadow DOM, Templates) je relativně nízkoúrovňové a vyžaduje značné množství boilerplate kódu pro každou komponentu, což může zpomalit vývoj a zvyšovat mentální zátěž v porovnání s frameworky jako React nebo Vue. [11]

Omezená podpora nativních funkcí (např. formuláře) - Native form-associated elementy (jako <input>, <button>) uvnitř Shadow DOMu nejsou automaticky integrovány do hostitelského <form>. To znamená, že komponenty s vlastními poli musí implementovat další rozhraní (FormAssociatedCustomElement) nebo komplikované workaroundy, aby fungovaly se standardními form-metodami (form.submit()). [12]

Problémy se stylováním a tematizací – I když Shadow DOM chrání interní styly komponenty, znesnadňuje zároveň aplikaci globálních stylů nebo design-systémů. Mechanismy jako CSS Shadow Parts (::part()) a CSS Custom Properties jsou poměrně složité a nedostačující pro pokročilé scénáře, například responzivní tematizaci či stylování vnějších wrapperů. [13]

Server-Side Rendering (SSR) a výkon – webové komponenty spoléhají na browser-specifické DOM API (Shadow DOM), která nejsou dostupná na serveru. To ztěžuje nebo znemožňuje tradiční SSR a předrenderování HTML bez JavaScriptu, případně vyžaduje těžkopádné polyfilly, Puppeteer-based workaroundy či externí knihovny (Lit SSR, Stencil Hydrate). [13] Navíc polyfilly pro starší prohlížeče mohou výrazně zvýšit velikost bundle a zpomalit načítání. [14]

Fragmentace ekosystému a nízká povědomost – I přes standardizaci chybí jednotná referenční implementace či robustní tooling, jaký nabízí velké frameworky. Marketingová a adopční křivka je strmá -- mnoho týmů nemá dostatek zkušeností s Web Components, což vede k obavám z nedostatku know-how, testovacích nástrojů i komunitní podpory. [11]

Migrace a interoperabilita – Ačkoliv lze webové komponenty používat vedle všech frameworků, reálná interoperabilita je omezena. Komponenty často nejsou plně kompatibilní napříč různými verzemi frameworku nebo build-nástrojů, což komplikuje jejich nasazení v prostředí, kde se technologie rychle mění. [15]

Přístupnost (Accessibility) - Izolace Shadow DOMu klade výzvy při implementaci WAI-ARIA rolí a správě focusu. Vývojář musí ručně zajistit ElementInternals, správné role, aria-* atributy a tabindex, jinak hrozí, že komponenta nebude přístupná pro uživatele se speciálními potřebami.[14]

Testovatelnost – Shadow DOM vyžaduje speciální testovací přístupy -- standardní end-to-end nástroje nemusí Shadow DOM automaticky procházet a snímání snapshotů může vyžadovat dodatečné konfigurační úpravy v test-frameworku (např. @open-wc/testing). [11]

1.8 Implementační knihovny pro webové komponenty

1.8.1 LitElement

LitElement je dnes jednou z nejrozšířenějších knihoven pro Web Components. Vedle reaktivity a šablonového zápisu nabízí také:

Ecosystém doplňků – Můžete snadno přidat @lit-labs/virtualizer pro virtualizaci dlouhých seznamů, @lit-labs/context pro sdílení datových kontextů mezi komponentami nebo @lit-labs/task pro asynchronní operace.

Debugging a telemetrie – Díky updateComplete promise lze v testech čekat na dokončení vykreslení. Chrome DevTools umí zobrazit Shadow DOM, přepínat CSS custom properties a debugovat event-handlery přímo v šabloně.

Přístupnost – V rámci render() můžete definovat ARIA atributy a role (např. <table role="grid">), přičemž LitElement nijak neomezuje styl ani semantiku, takže lze snadno zajistit vysokou míru WCAG compliance.

Nasazení a CI/CD – Komponenty snadno zabudujete do jakéhokoliv build chainu (Webpack, Rollup, Vite). Při použití Vite získáte ultra rychlý HMR (hot module replacement) a okamžitou zpětnou vazbu během vývoje.

Instalace a základní nastavení:

```
npm install lit
```

Vytvoření komponenty:

```
import { LitElement, html, css } from 'lit';

export class MyTable extends LitElement {
  static properties = {
    columns: { type: Array },
    data: { type: Array },
    pageSize: { type: Number }
  };

  static styles = css`
    table { width: 100%; border-collapse: collapse; }
    th, td { padding: 0.5rem; border: 1px solid #ccc; }
    th { background: #f0f0f0; cursor: pointer; }
  `;

  constructor() {
    super();
    this.columns = [];
    this.data = [];
    this.pageSize = 10;
  }

  render() {
    return html`
      <table>
```

```
        <thead>
          <tr>
            ${this.columns.map(col => html`
              <th @click=${() => this._sort(col.key)}>
                ${col.label}
              </th>`)}
          </tr>
        </thead>
        <tbody>
          ${this.data.slice(0, this.pageSize).map(row => html`
            <tr>
              ${this.columns.map(col => html`<td>${row[col.key]}</td>`)}
            </tr>`)}
        </tbody>
      </table>`;
  }

  _sort(key) {
    this.data = [...this.data].sort((a, b) => a[key] > b[key] ? 1 : -1);
  }
}

customElements.define('my-table', MyTable);
```

Výhody LitElementu:

- Reaktivní aktualizace bez ručního DOM managementu.
- Efektivní šablonovací engine lit-html pro podmínky a cykly.
- Vestavěný scoped CSS a podpora CSS custom properties pro theming.

1.8.2 Stencil

Stencil vybočuje tím, že je to **kompilátor**, nikoli runtime knihovna. To umožňuje:

Production-ready výstup – V produkci vznikají čisté Web Components se všemi polyfilly jen pro nutné prohlížeče. Výsledný bundle je optimalizován na minimální velikost.

SSR & hydration – Stencil podporuje generování statických HTML i „hydratační" kód, takže lze komponentu před-renderovat na serveru a na klientu pouze "oživit".

Design systémy – Ionic Framework, který používá Stencil, nabízí připravené UI komponenty (karty, tlačítka, seznamy), jež snadno přizpůsobíte svým pravidlům brandingu.

```
npm init stencil
```

```
import { Component, Prop, h } from '@stencil/core';
@Component({
  tag: 'my-table',
  styleUrl: 'my-table.css',
  shadow: true
})
export class MyTable {
  @Prop() columns = [];
```

```
  @Prop() data = [];
  @Prop() pageSize = 10;

  private sort(key) { /* ... */ }

  render() {
    return (
      <table>...</table>
    );
  }
}
```

1.8.3 SkateJS

SkateJS je výjimečný svou modularitou:

Mix-in architektura – Místo jedné „velké" třídy nabízí několik mixinů pro lifecycle, reaktivitu, eventy a přidávání pluginů, takže si složíte přesně to, co potřebujete.

Choice of renderer – Není vázáno na lit-html -- můžete použít uhtml, preact nebo úplně vlastní funkci, co generuje DOM.

Snadná integrace – Komponentu definujete a pak ji hned použijete vedle jiných frameworků, bez nutnosti extra loaderů, protože registrujete čistý customElements.define.

```
npm install @skatejs/element lit-html
import { define, props } from "@skatejs/element";
import { html, render } from "lit-html";

class MyTable extends HTMLElement {
  static props = {
    columns: props.array.default(() => []),
    data: props.array.default(() => []),
    pageSize: props.number.default(10),
  };

  connectedCallback() {
    this._update();
  }

  updated() {
    this._update();
  }

  _update() {
    render(
      html`<table>
        ...
      </table>`,
      this
    );
  }
}

define("my-table", MyTable);
```

Výhody SkateJS:

- Extrémně malá velikost
- Plná kontrola nad lifecycle
- Flexibilní výběr render-engine

1.8.4 Vanilla Custom Elements

Nativní cesta bez jakéhokoli externího runtime. Komponenta je čistá třída rozšiřující HTMLElement, ve které ručně vytvoříte <template>, vkládáte Shadow DOM a spravujete změny state → DOM:

```
const tpl = document.createElement("template");
tpl.innerHTML = `
  <style>...</style>
  <table>
    <thead><tr id="header"></tr></thead>
    <tbody id="body"></tbody>
  </table>`;

class MyTable extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" }).appendChild(
      tpl.content.cloneNode(true)
    );
  }

  set columns(cols) {
    this._columns = cols;
    this._renderHeader();
  }

  set data(rows) {
    this._data = rows;
    this._renderBody();
  }

  _renderHeader() { /* ... */ }
  _renderBody() { /* ... */ }
}

customElements.define("my-table", MyTable);
```

Čisté API se hodí tam, kde chceme **absolutní kontrolu**:

Polyfilling – Pro staré prohlížeče (IE11) stačí načíst @webcomponents/webcomponentsjs polyfill bundle. Odtud pak vše funguje identicky.

Manuální optimalizace – Můžete cache-ovat šablonu, dělit Shadow DOM na menší fragmenty, nebo dokonce implementovat vlastní „dirty-checking" pro state-to-DOM pipeline.

Zero-overhead – Pokud vaše aplikace provozuje jen pár widgetů, nemusíte do bundle zahrnovat žádnou další knihovnu -- stačí váš JS soubor s definicí komponenty.

1.9 Souhrn

Tato kapitola poskytla komplexní přehled webových komponent od jejich historie přes architekturu až po současné implementace. Byly analyzovány klíčové tabulkové

komponenty dostupné na trhu včetně jejich dekompozice na subkomponenty. Kapitola také prozkoumala různá API pro práci s daty (REST, GraphQL, OData) a detailně rozebrala problematiku responzivity tabulek pro různá zařízení.

Webové komponenty představují zajímavou alternativu k tradičním frameworkům díky své nativní podpoře v prohlížečích, silné enkapsulaci a interoperabilitě. Na druhou stranu mají i své limity, zejména v oblasti SSR, komplexity API a ekosystému.

Pro implementaci tabulkové komponenty se jako nejvhodnější jeví použití LitElement knihovny, která poskytuje dobrý kompromis mezi jednoduchostí použití, výkonem a velikostí. V následující kapitole bude představen detailní návrh takové komponenty včetně všech požadavků a implementačních detailů.

2 NÁVRH WEBOVÉ KOMPONENTY PRO PRÁCI S TABULKOVÝMI DATY

Tato kapitola se věnuje komplexnímu návrhu webové komponenty <my-table> pro práci s tabulkovými daty. Nejprve jsou analyzovány funkční a nefunkční požadavky, následně je představena architektura řešení včetně dekompozice na jednotlivé subkomponenty. Kapitola dále obsahuje návrh API rozhraní, zpracování responzivity pro různá zařízení a detailní UX/UI návrh.

2.1 Analýza požadavků

2.1.1 Funkční požadavky

Funkční požadavky definují, co všechno musí komponenta umět z pohledu koncového uživatele a vývojáře, který ji bude integrovat:

FR1 – Zobrazení dat

- Komponenta musí umět zobrazit tabulková data předaná jako pole objektů.
- Musí podporovat dynamické sloupce definované za běhu.
- Musí umožnit definici vlastních šablon pro jednotlivé buňky.

FR2 – Filtrování

- Umožnit filtrování podle libovolného sloupce.
- Podporovat různé typy filtrů (text, číslo, datum, výběr z možností).
- Real-time filtrování při psaní.
- Možnost kombinace více filtrů současně.

FR3 – Řazení

- Řazení vzestupně/sestupně kliknutím na záhlaví sloupce.
- Vizuální indikace aktuálního řazení.
- Podpora multi-column řazení s prioritami.

FR4 – Stránkování

- Konfigurovatelná velikost stránky.

- Navigace mezi stránkami (první, předchozí, následující, poslední).
- Přímý skok na konkrétní stránku.
- Zobrazení informace o aktuální stránce a celkovém počtu.

FR5 – Seskupování

- Seskupování řádků podle hodnoty vybraného sloupce.
- Rozbalování/sbalování skupin.
- Agregační funkce pro skupiny (součet, průměr, počet).

FR6 – Editace dat

- Inline editace buněk dvojklikem nebo klávesou Enter.
- Validace vstupních dat podle typu sloupce.
- Možnost zrušit editaci (Escape).
- Podpora pro různé editory (text, číslo, datum, select).

FR7 – Výběr řádků

- Single a multi-select režim.
- Select all/deselect all.
- Klávesové zkratky pro výběr (Shift+click, Ctrl+click).

FR8 – Postupné načítání dat (Lazy loading)

- Komponenta musí podporovat načítání dat po částech během scrollování nebo stránkování.
- Uživatel pracuje vždy jen s relevantní částí datasetu, což zlepšuje výkon a odezvu aplikace.
- Při dosažení prahu se automaticky spustí event table-data-request s parametry pro další data.

FR9 – Zobrazení/skrytí sloupců tabulky

- Komponenta musí umožnit dynamické skrývání a zobrazování jednotlivých sloupců.

- Uživatel si může nastavit, které atributy chce mít zobrazené v tabulce.
- Funkce je řešena prostřednictvím vlastnosti visible v konfiguraci sloupce a metody setColumnVisibility().
- Komponenta si pamatuje stav viditelnosti sloupců a obnovuje jej při dalším načtení.

FR10 – Dvoucestná vazba (Two-way binding)

- Komponenta podporuje automatickou synchronizaci dat mezi komponentou a hostitelskou aplikací.
- Změny provedené v tabulce se automaticky propagují zpět do zdrojových dat.
- Umožňuje seamless integraci s moderními frameworky (React, Vue, Angular).

2.1.2 Nefunkční požadavky

NFR1 – Výkon

- Plynulé scrollování i při 10 000+ řádcích.
- Odezva na uživatelské akce <100ms.
- Virtualizace DOM elementů pro velké datasety.

NFR2 – Responzivita

- Adaptivní layout pro různé velikosti obrazovek.
- Horizontální scrollování na malých zařízeních.
- Prioritizace viditelných sloupců podle důležitosti.
- Touch-friendly ovládání na mobilních zařízeních.

NFR4 – Kompatibilita

- Podpora všech moderních prohlížečů (Chrome, Firefox, Safari, Edge).
- Graceful degradation pro starší prohlížeče.
- Nezávislost na konkrétním frameworku.
- Použitelnost v React, Angular, Vue i vanilla JS.

NFR5 – Rozšiřitelnost

- Pluginová architektura pro přidávání funkcí.
- Event-driven komunikace.
- Customizovatelné styly přes CSS proměnné.
- Možnost override jednotlivých částí komponenty.

2.2 Use Case diagram

Use case diagram představuje vysokoúrovňový pohled na funkčnost systému z perspektivy různých aktérů. V kontextu tabulkové komponenty identifikujeme tři hlavní aktéry: **Uživatel** (end-user interagující s tabulkou), **Vývojář** (developer integrující komponentu do aplikace) a **Systém** (automatické procesy běžící na pozadí).

Popis hlavních use cases:

UC1 – Zobrazit data

Aktéři: Uživatel, Vývojář
Popis: Načtení a zobrazení tabulkových dat v komponentě
Hlavní scénář:

1. Vývojář nastaví vlastnost data nebo dataUrl
2. Komponenta validuje a zpracuje data
3. Komponenta vykreslí tabulku s řádky a sloupci
4. Uživatel vidí naplněnou tabulku

Alternativní scénář: Při chybě načítání se zobrazí chybová hláška a emituje se event table-data-error

UC2 – Filtrovat data

Aktéři: Uživatel
Popis: Filtrování řádků podle zadaných kritérií
Hlavní scénář:

1. Uživatel zadá text do filtračního pole
2. Komponenta aplikuje debouncing (300ms)
3. Tabulka zobrazí pouze odpovídající řádky
4. Emituje se event table-filter-changed

UC3 – Řadit data

Aktéři: Uživatel
Popis: Řazení dat podle hodnot ve sloupcích
Hlavní scénář:

1. Uživatel klikne na hlavičku sloupce
2. Data se seřadí vzestupně (první klik), sestupně (druhý klik), nebo se zruší řazení (třetí klik)
3. Zobrazí se vizuální indikátor řazení (▲/▼)
4. Emituje se event table-sort-changed

UC4 – Stránkovat data

Aktéři: Uživatel
Popis: Rozdělení velkých datasetů na menší stránky
Hlavní scénář:

1. Uživatel klikne na navigační prvek paginace
2. Komponenta vypočítá rozsah dat pro zobrazení
3. Tabulka zobrazí data pro vybranou stránku
4. Aktualizuje se informace o paginaci

UC5 – Editovat buňku

Aktéři: Uživatel
Popis: Inline editace hodnot v tabulce
Hlavní scénář:

1. Uživatel dvojklikne na editovatelnou buňku
2. Aktivuje se editační mód s input polem
3. Uživatel změní hodnotu a stiskne Enter
4. Komponenta validuje a uloží změnu
5. Emituje se event table-cell-edited

Alternativní scénář: Při stisku Escape se editace zruší bez uložení

UC6 – Vybrat řádky

Aktéři: Uživatel
Popis: Označení řádků pro další operace
Hlavní scénář:

1. Uživatel klikne na checkbox řádku
2. Řádek se označí jako vybraný
3. Komponenta vizuálně zvýrazní výběr
4. Emituje se event table-row-selected

Alternativní scénáře: Multi-select s Ctrl/Shift, select all checkbox v hlavičce

UC7 – Seskupit data

Aktéři: Uživatel
Popis: Hierarchické seskupení řádků podle hodnot
Hlavní scénář:

1. Uživatel vybere sloupec pro seskupení
2. Komponenta vytvoří skupinové hlavičky
3. Řádky se uspořádají do skupin
4. Umožní se rozbalování/sbalování skupin

UC8 – Exportovat data

Aktéři: Uživatel
Popis: Export dat do externích formátů
Hlavní scénář:

1. Uživatel klikne na tlačítko Export
2. Vybere formát (CSV, Excel, PDF) a rozsah dat
3. Komponenta generuje soubor
4. Prohlížeč zahájí stahování

UC9 – Změnit velikost sloupce

Aktéři: Uživatel
Popis: Úprava šířky sloupců podle potřeby
Hlavní scénář:

1. Uživatel najede na hranici sloupce (kurzor ↔)
2. Táhne myší pro změnu šířky
3. Komponenta real-time aktualizuje layout
4. Emituje se event table-column-resized

Alternativní scénář: Dvojklik na hranici pro auto-fit podle obsahu

UC10 – Přeuspořádat sloupce

Aktéři: Uživatel
Popis: Změna pořadí sloupců drag & drop
Hlavní scénář:

1. Uživatel začne táhnout hlavičku sloupce
2. Zobrazí se vizuální indikátory přesunu
3. Uvolní sloupec na nové pozici

4. Komponenta přeuspořádá sloupce

UC11 – Konfigurovat komponentu

Aktéři: Vývojář
Popis: Nastavení vlastností a chování komponenty
Hlavní scénář:

1. Vývojář definuje konfiguraci sloupců
2. Nastaví vlastnosti komponenty (sortable, filterable, editable)
3. Registruje event listenery
4. Komponenta aplikuje konfiguraci

UC12 – Poslouchat události

Aktéři: Vývojář
Popis: Reakce na uživatelské interakce prostřednictvím událostí
Hlavní scénář:

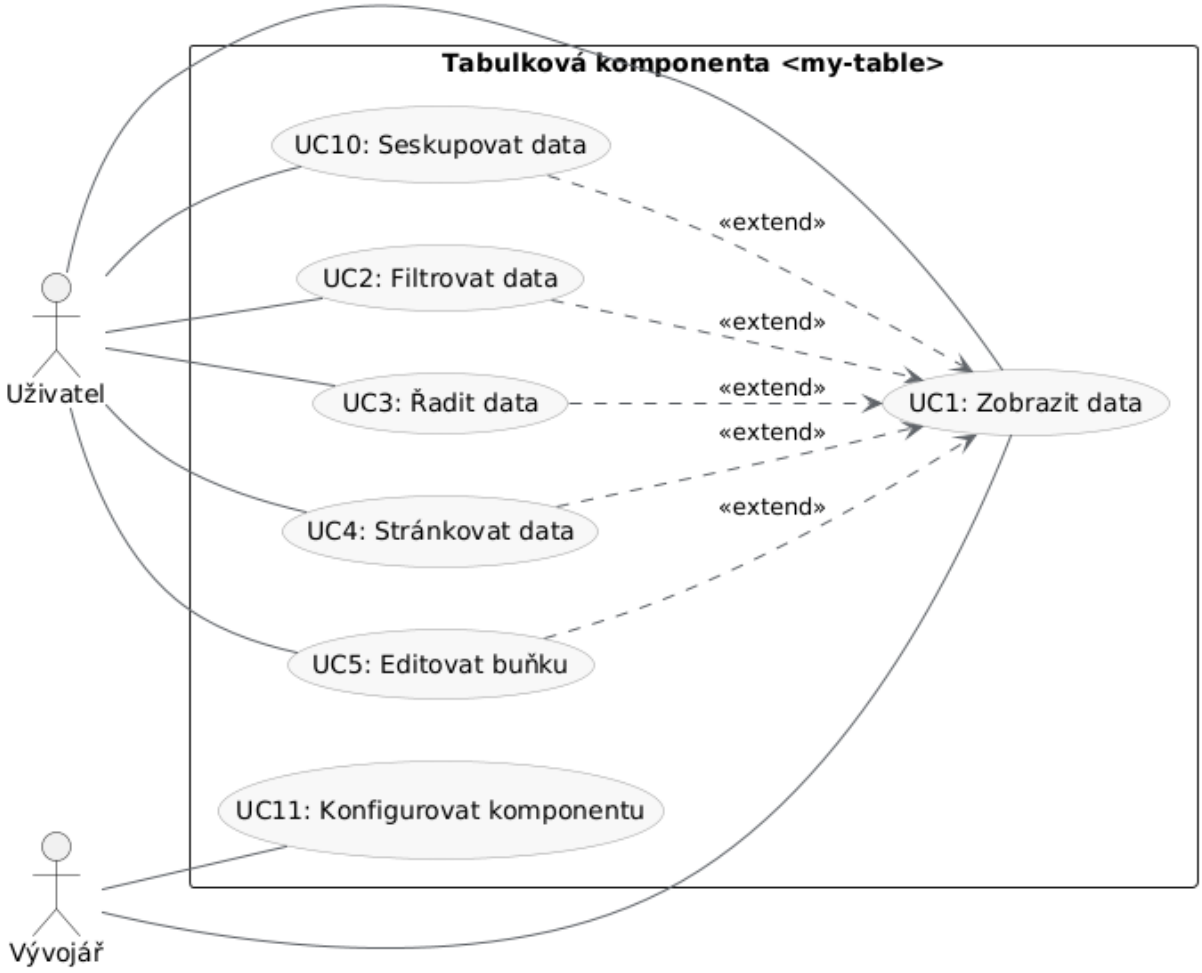
1. Vývojář registruje event listener
2. Uživatel provede akci v tabulce
3. Komponenta emituje custom event
4. Callback funkce zpracuje event data

UC13 – Postupné načítání dat (Lazy Loading)

Aktéři: Uživatel
Popis: Automatické načítání dalších dat při scrollování
Hlavní scénář:

1. Uživatel scrolluje blízko konce tabulky
2. Systém detekuje dosažení loadThreshold
3. Emituje se event table-data-request
4. Vývojář poskytne další data
5. Komponenta připojí nová data k existujícím

Alternativní scénář: Při chybě načítání se zobrazí chybová hláška s možností opakování



Obrázek 4 – Use Case diagram[42]

2.3 Architektura komponenty

Architektura komponenty <my-table> je navržena podle principů separation of concerns, vysoké koheze a nízké provázanosti. Komponenta je rozdělena na logické celky, které spolu komunikují přes dobře definovaná rozhraní.

2.3.1 Diagram komponent

Diagram komponent zobrazuje hierarchickou strukturu a závislosti mezi jednotlivými částmi systému. Hlavní komponenta MyTable orchestruje tři klíčové subkomponenty (Header, Body, Footer) a využívá služby pro business logiku a utility pro pomocné funkce.

Detailní popis komponent:

Header komponenty:

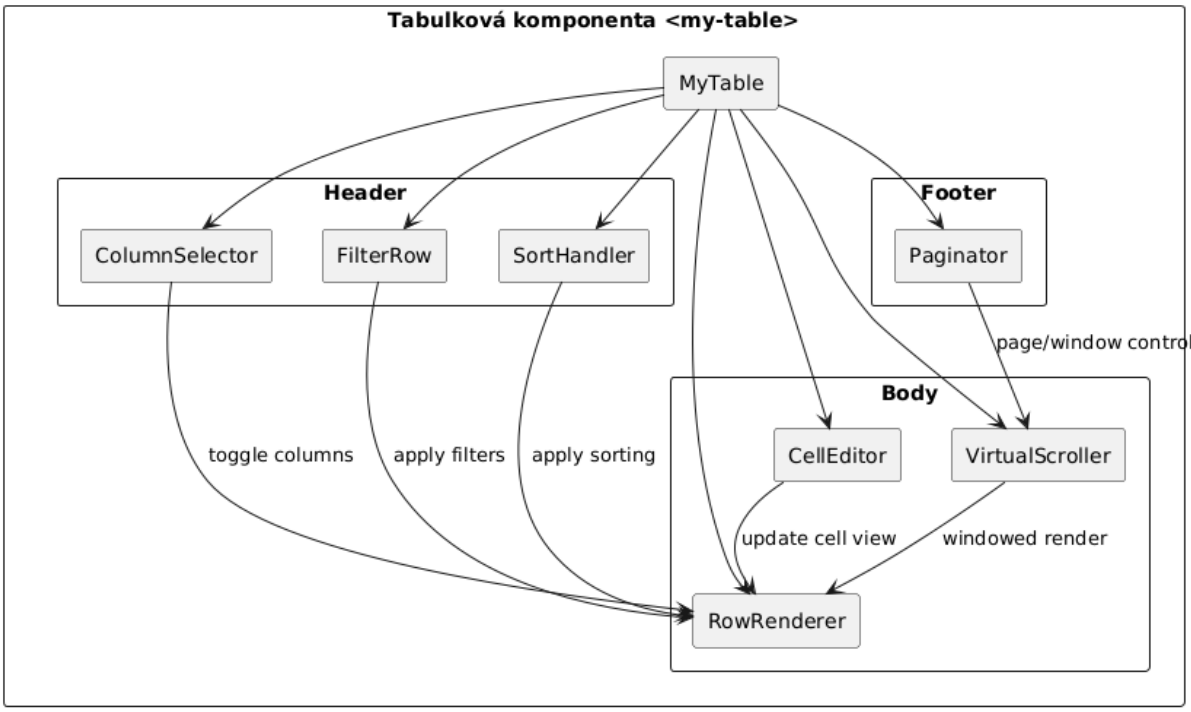
- **ColumnSelector**: Umožňuje uživatelům dynamicky skrývat/zobrazovat sloupce. Udržuje stav viditelnosti sloupců a poskytuje UI pro jejich správu.
- **FilterRow**: Renderuje řádek s filtračními poli pod hlavičkami sloupců. Každé pole je přizpůsobeno datovému typu sloupce (text input, date picker, select box).
- **SortHandler**: Spravuje logiku řazení včetně multi-column sortingu. Udržuje informace o aktuálním řazení a zajišťuje vizuální indikaci.

Body komponenty:

- **RowRenderer**: Zodpovědný za efektivní vykreslování řádků. Implementuje optimalizace jako row recycling a memoizaci.
- **CellEditor**: Poskytuje inline editaci buněk s podporou různých typů editorů. Spravuje lifecycle editace od aktivace po uložení/zrušení.
- **VirtualScroller**: Implementuje virtualizaci pro velké datasety. Renderuje pouze viditelné řádky a spravuje scrollbar.

Footer komponenty:

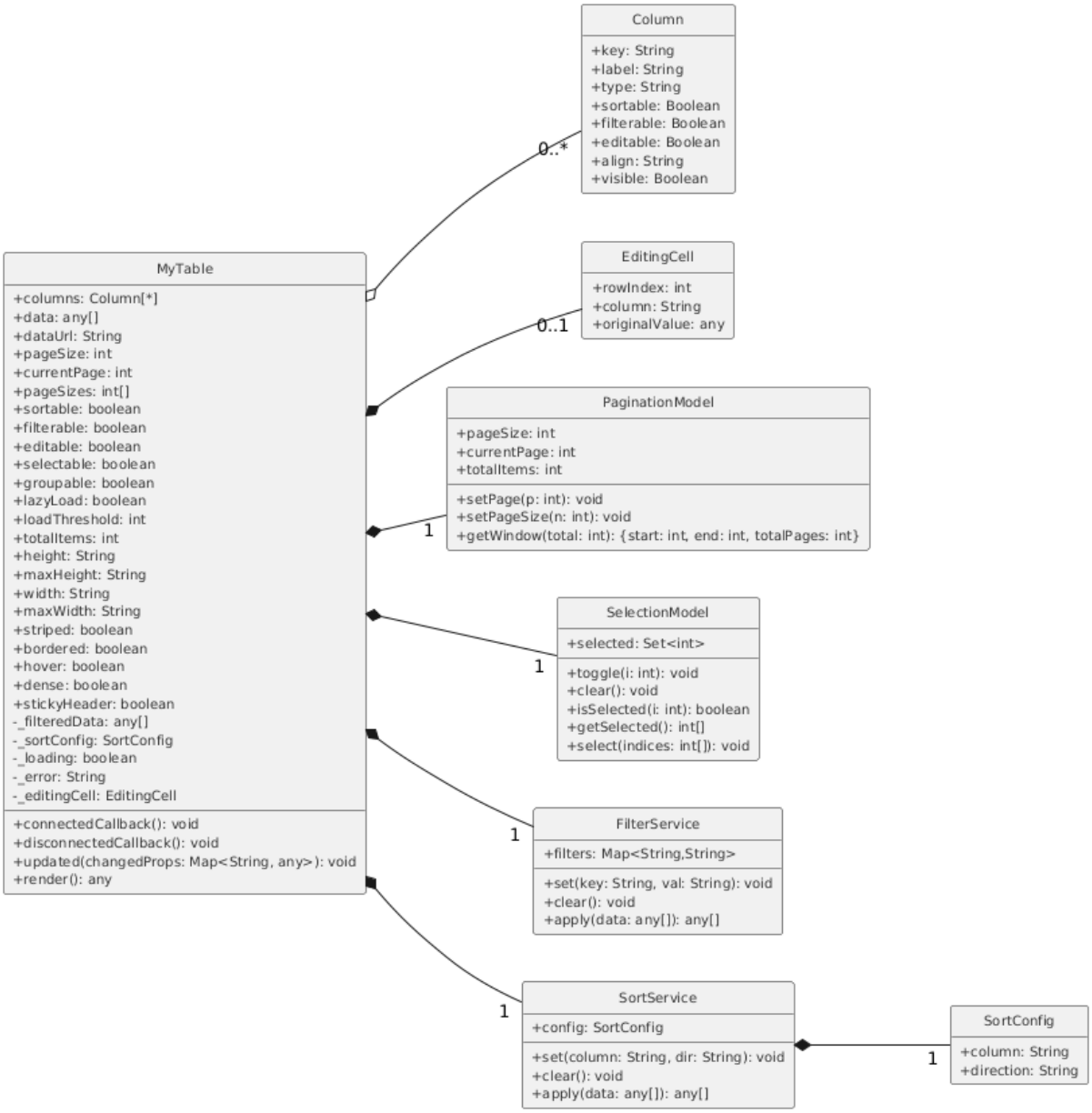
- **Paginator**: Komplexní navigace mezi stránkami s možností změny velikosti stránky a přímého skoku.



Obrázek 5 – Diagram component[43]

2.3.2 Diagram tříd

Diagram tříd detailně popisuje strukturu objektů, jejich atributy, metody a vzájemné vztahy. Tento návrh využívá kompozici a dědičnost pro dosažení flexibility a znovupoužitelnosti.



Obrázek 6 – Diagram tříd[44]

2.4 Návrh API

API komponenty je navrženo s důrazem na intuitivnost, flexibilitu a kompatibilitu s moderními webovými frameworky. Následuje princip "convention over configuration" - komponenta funguje rozumně i s minimální konfigurací, ale umožňuje detailní customizaci.

2.4.1 Vlastnosti komponenty

Vlastnosti komponenty jsou rozděleny do logických skupin podle funkcionality. Každá vlastnost má definovaný typ, výchozí hodnotu a jasný účel.

Detailní popis klíčových vlastností:

Data properties:

- columns definuje strukturu tabulky. Každý sloupec má povinné key (identifikátor v datech) a label (zobrazený text).
- data vs dataUrl - komponenta podporuje jak lokální data (array objektů), tak vzdálené načítání z URL.

Feature flags:

- Boolean vlastnosti umožňují zapnout/vypnout jednotlivé funkce. To zjednodušuje použití a zlepšuje výkon (neaktivní funkce se nerenderují).
- selectable může být boolean nebo string - umožňuje jemnou kontrolu nad typem výběru.

Column configuration:

- formatter funkce transformuje raw hodnotu na zobrazitelný obsah. Může vracet string nebo lit-html template.
- validator zajišťuje integritu dat při editaci. Vrací true pro validní hodnotu nebo string s chybovou hláškou.
- fixed sloupce zůstávají viditelné při horizontálním scrollování.

```
interface MyTableProps {
  // Data
  columns: ColumnConfig[];
  data?: any[];
  dataUrl?: string;

  // Pagination
  pageSize?: number;
  currentPage?: number;
  pageSizes?: number[];

  // Features
  sortable?: boolean;
  filterable?: boolean;
  editable?: boolean;
  selectable?: boolean | 'single' | 'multiple';
  groupable?: boolean;

  // Lazy loading
  lazyLoad?: boolean;
```

```
  loadThreshold?: number;
  totalItems?: number;

  // Appearance
  height?: string;
  striped?: boolean;
  bordered?: boolean;
  hover?: boolean;
  dense?: boolean;

  // Localization
  locale?: string;
  messages?: LocalizationMessages;
}

interface ColumnConfig {
  key: string;
  label: string;
  width?: number | string;
  minWidth?: number;
  maxWidth?: number;
  sortable?: boolean;
  filterable?: boolean;
  editable?: boolean;
  visible?: boolean;
}
```

2.4.2 Methods (Metody)

Veřejné metody poskytují programatický přístup k funkcionalitě komponenty. Jsou navrženy tak, aby byly chainable (vrací this) kde to dává smysl.

```
interface MyTableMethods {
  // Data manipulation
  setData(data: any[]): void;
  appendData(data: any[]): void;
  updateRow(index: number, row: any): void;
  deleteRow(index: number): void;

  // Column management
  addColumn(column: ColumnConfig): void;
  removeColumn(key: string): void;
  updateColumn(key: string, config: Partial<ColumnConfig>): void;
  setColumnVisibility(key: string, visible: boolean): void;

  // Filtering
  setFilter(column: string, value: any): void;
  clearFilter(column?: string): void;
  getActiveFilters(): Filter[];

  // Sorting
  sort(column: string, direction?: 'asc' | 'desc'): void;
  clearSort(): void;

  // Selection
  selectRow(index: number): void;
  deselectRow(index: number): void;
  selectAll(): void;
  deselectAll(): void;
  getSelectedRows(): any[];
```

```
  // Utils
  refresh(): void;
  validate(): boolean;
  scrollToRow(index: number): void;
}
```

2.4.3 Events (Události)

Komponenta komunikuje s okolím pomocí custom events. Každá událost nese detailní informace o změně, což umožňuje fine-grained reakce.

```
interface MyTableEvents {
  // Data events
  'table-data-loaded': { data: any[] };
  'table-data-error': { error: Error };

  // Edit events
  'table-cell-edit-start': { row: number; column: string; value: any };
  'table-cell-edited': { row: number; column: string; oldValue: any; newValue: any
};
  'table-cell-edit-cancel': { row: number; column: string };

  // Selection events
  'table-row-selected': { row: any; index: number };
  'table-row-deselected': { row: any; index: number };
  'table-selection-changed': { selected: any[] };

  // Sort events
  'table-sort-changed': { column: string; direction: 'asc' | 'desc' };

  // Filter events
  'table-filter-changed': { filters: Filter[] };

  // Page events
  'table-page-changed': { page: number; pageSize: number };

  // Column events
  'table-column-resized': { column: string; width: number };
  'table-column-reordered': { from: number; to: number };
  'table-column-visibility-changed': { column: string; visible: boolean };
```

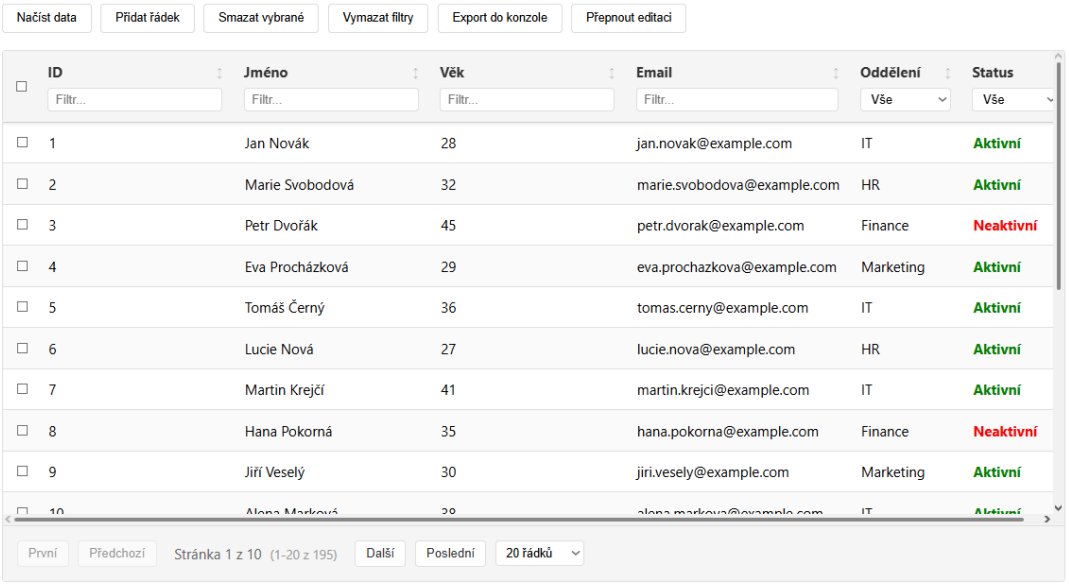
2.5 Návrh UI/UX

2.5.1 Layout komponenty

Návrh desktopové verze komponenty se skládá z několika logických částí. V horní části rozhraní je umístěn panel s ovládacími prvky, který zahrnuje například tlačítka pro export dat či nastavení viditelnosti jednotlivých sloupců. Pod tímto panelem se nachází hlavička tabulky s názvy sloupců a indikátory směru řazení, jež uživateli poskytují možnost interaktivního uspořádání dat. Bezprostředně pod hlavičkou je integrován řádek s filtračními poli, která umožňují zadávat podmínky filtrování pro každý jednotlivý sloupec.

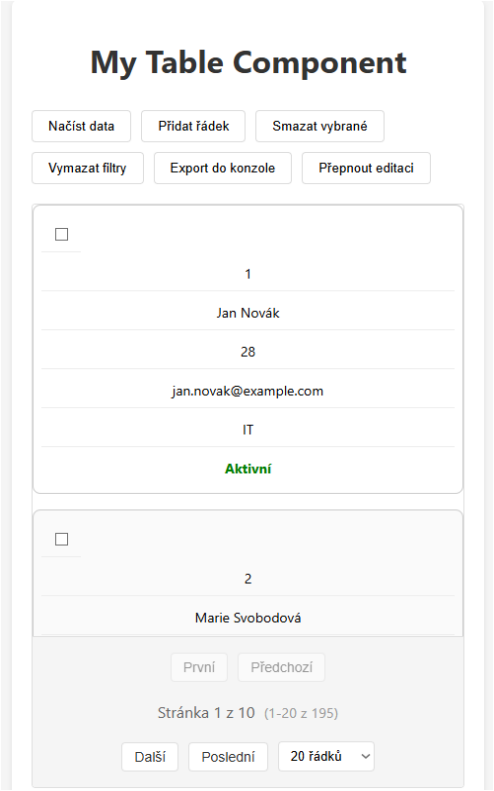
Střední část rozhraní tvoří vlastní tělo tabulky, které zobrazuje data uspořádaná do řádků a sloupců. První sloupec obsahuje výběrové checkboxy, které podporují vícečetný výběr záznamů. Spodní část komponenty pak poskytuje informace o stránkování, například formou textu „Zobrazeno 1–20 z 150“, a zároveň nabízí navigační prvky pro přechod mezi stránkami. K dispozici jsou tlačítka pro přechod na první, předchozí, následující či poslední stránku a také číselná reprezentace jednotlivých stránek.

Celý layout je optimalizován pro práci na velkých obrazovkách s šířkou nad 1024 px, přičemž využívá dostupný prostor k zajištění maximální přehlednosti a efektivního zobrazení dat.



Obrázek 7 – Desktopová verze komponetny[45]

Obrázek 5 představuje mobilní verzi komponenty, která je určena pro zobrazení na zařízeních s šířkou menší než 768 px. Namísto klasické tabulky je použit kartový layout, kdy každý původní řádek tabulky je zobrazen jako samostatná karta. Obsah karty je organizován vertikálně ve formátu název atributu a hodnota, přičemž jsou prioritizovány nejdůležitější informace, jako je název, stav či dostupné akce. Méně významné atributy jsou implicitně skryté a uživatel je může zobrazit prostřednictvím volby „Více informací“. Celý layout je přizpůsoben dotykovému ovládání – interaktivní prvky mají větší plochu pro snadnější ovládání, hlavička se sticky chováním poskytuje rychlý přístup k filtrování a vyhledávání a stránkování je zjednodušeno pro potřeby mobilního prostředí. K dispozici jsou rovněž swipe gesta, která umožňují rychlé akce, například označení či smazání záznamu.



Obrázek 8 – Mobilní verze komponetny[46]

Oba wireframy demonstrují adaptivní přístup k zobrazení tabulkových dat, kde se zachovává funkcionalita při změně prezentační vrstvy podle velikosti obrazovky zařízení.

2.5.2 Interakční vzory

Řazení

- Kliknutí na záhlaví sloupce → řazení vzestupně
- Druhé kliknutí → řazení sestupně
- Třetí kliknutí → zrušení řazení
- Vizuální indikace: ▲ (asc), ▼ (desc)

Změna velikosti sloupců

- Hover nad hranou sloupce → kurzor resize
- Drag → real-time změna šířky
- Dvojklik na hranu → auto-fit podle obsahu

Editace

- Dvojklik na buňku → editační režim
- Enter → uložit a přejít na další řádek

- Tab → uložit a přejít na další buňku

- Escape → zrušit editaci

Výběr řádků

- Klik na checkbox → výběr řádku

Filtrování

- Pole z placeholderem Filtr

- Při zadavání textu, záznamy se filtruji

2.5.3 Responzivní chování

Desktop (>1024px)

- Plný layout se všemi funkcemi

- Horizontální scroll pro široké tabulky

- Hover efekty na řádcích

Tablet (768-1024px)

- Skrytí méně důležitých sloupců

- Zmenšené paddingy

- Touch-friendly velikost ovládacích prvků

Mobile (<768px)

- Kartový layout pro každý řádek

- Vertikální zobrazení dat

- Swipe gesta pro akce

- Sticky hlavička při scrollování

Příklad responzivního layoutu

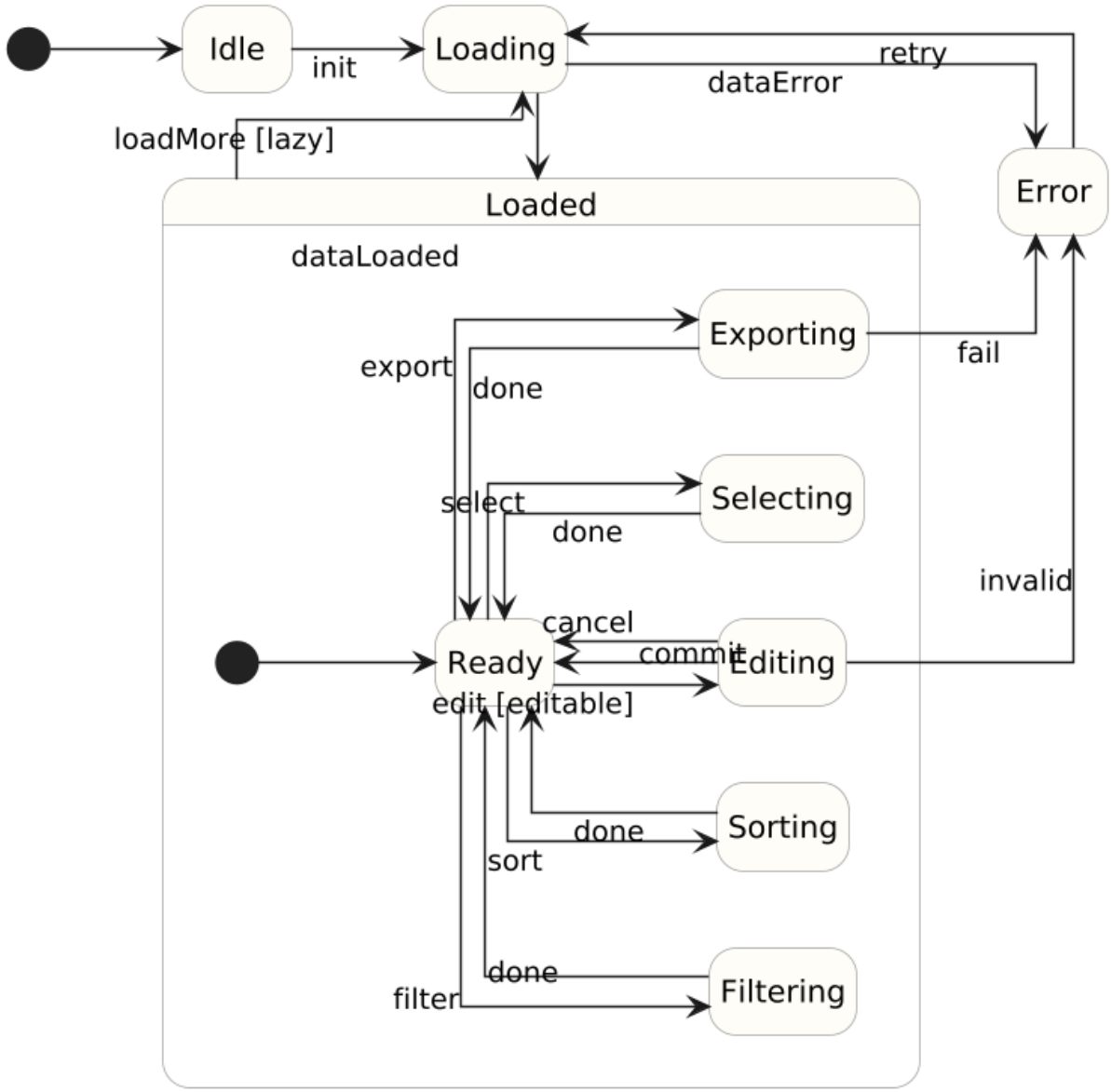
```
@media (max-width: 768px) {
  .table-row {
    display: block;
    border: 1px solid #ddd;
    margin-bottom: 10px;
    padding: 10px;
  }

  .table-cell {
    display: flex;
    justify-content: space-between;
    padding: 5px 0;
  }
```

```
.table-cell::before {
  content: attr(data-label);
  font-weight: bold;
}
```

2.6 Stavový diagram

Stavový diagram představuje dynamický pohled na chování komponenty a zobrazuje, jak se komponenta pohybuje mezi různými stavy v reakci na události a uživatelské interakce. Každý stav reprezentuje konkrétní konfiguraci komponenty s jasně definovanými možnostmi přechodu do dalších stavů.



Obrázek 9 – Stavový diagram[47]

Popis hlavních stavů:

Idle (Nečinný) Počáteční stav komponenty po vytvoření. V tomto stavu:

- Komponenta je inicializována s výchozími hodnotami
- Shadow DOM je vytvořen a připraven
- Čeká se na první data nebo konfiguraci
- Jsou registrovány základní event listenery

Loading (Načítání) Komponenta aktivně načítá data ze zdroje:

- Zobrazuje se loading indikátor (spinner nebo skeleton)
- Uživatelské interakce jsou dočasně zakázány
- Probíhá HTTP request nebo zpracování lokálních dat
- Timeout je nastaven pro případ selhání

Loaded (Načteno) Hlavní pracovní stav komponenty:

- Data jsou úspěšně načtena a zobrazena
- Všechny interaktivní funkce jsou dostupné
- Komponenta reaguje na uživatelské vstupy
- Z tohoto stavu lze přejít do většiny ostatních stavů

Error (Chyba) Stav indikující selhání operace:

- Zobrazuje se chybová zpráva s popisem problému
- Nabízí se možnost opakování (retry button)
- Loguje se detailní informace o chybě
- Může obsahovat fallback zobrazení

Filtering (Filtrování) Přechodný stav během aplikace filtrů:

- Probíhá výpočet filtrovaných dat
- Debouncing zajišťuje, že se filtr neaplikuje při každém keystroke
- Po dokončení se vrací do stavu Loaded s filtrovanými daty

Sorting (Řazení) Komponenta přeuspořádává data:

- Vizuální indikace probíhajícího řazení
- Zachování pozice scrollu pokud možno
- Podpora pro stable sort (zachování pořadí stejných hodnot)

Editing (Editace) Jedna nebo více buněk je v editačním režimu:

- Fokus je na editovaném elementu
- Původní hodnota je uložena pro možnost návratu
- Klávesové zkratky jsou aktivní (Enter, Escape, Tab)

Selecting (Výběr) Uživatel vybírá řádky:

- Podpora single a multi-select režimů
- Vizuální zvýraznění vybraných řádků
- Správa selection state

Exporting (Export) Probíhá export dat:

- Progress bar pro velké datasety

Každý přechod mezi stavy může mít podmínky, které musí být splněny:

```
// Příklad guard clause pro přechod do editace
canTransitionToEditing() {
  return this.editable &&
         !this.isLoading &&
         this.hasEditPermission &&
         !this.hasUnsavedChanges;
}

// Příklad akce při přechodu
onEnterEditingState() {
  this.saveOriginalValue();
  this.focusEditableElement();
  this.attachEditingListeners();
  this.emit('table-edit-started');
}
```

2.6 Závěr návrhu

Navržená komponenta <my-table> představuje komplexní řešení pro práci s tabulkovými daty v moderních webových aplikacích. Architektura je navržena s důrazem na:

- **Modularitu** - každá funkce je oddělená a může být použita samostatně
- **Rozšiřitelnost** - snadné přidávání nových funkcí pomocí pluginů
- **Výkon** - virtualizace a lazy loading pro velké datasety
- **Responzivitu** - optimalizované zobrazení na všech zařízeních

Implementace bude následovat tento návrh s využitím moderních webových standardů a best practices.

3 IMPLEMENTACE WEBOVÉ KOMPONENTY

Tato kapitola detailně popisuje implementaci webové komponenty <my-table>. Kód je rozdělen do logických celků, aby bylo zřejmé, jak fungují jednotlivé části – od definice vlastností, přes renderovací šablony až po obsluhu událostí a veřejné API.

3.1 Implementace komponenty

Pro komponentu pro práci s tabulkovými daty <my-table> byly zvoleny technologie Web Components a knihovna Lit. Komponenta je implementována jako třída MyTable, která dědí z LitElement. Lit poskytuje deklarativní způsob renderování a reaktivitu vlastností, což zjednodušuje práci s daty a jejich zobrazením. Zdrojový kód je psán jako ES moduly a pro build i lokální vývoj je použit nástroj Vite. Vzhled je oddělen do samostatného souboru my-table-styles.js, což umožňuje lepší přehlednost a modularitu. Projektová struktura je navržena tak, aby byla jasně oddělena logika komponenty, její vzhled a ukázkové použití. Hlavní logika se nachází v souboru my-table.js, styly v my-table-styles.js, ukázková

inicializace v main.js a demonstrační rozhraní v index.html.

Životní cyklus komponenty se skládá z několika částí. Konstruktor inicializuje všechny veřejné vlastnosti i interní stav. Metody *connectedCallback* a *disconnectedCallback* přidávají a odebírají posluchače událostí, zatímco metoda *updated* reaguje na změny předaných vlastností. Samotný výstup komponenty zajišťuje metoda *render*, která skládá výslednou HTML šablonu. Veřejné vlastnosti zahrnují například nastavení sloupců a dat, stránkování, funkce filtrování, řazení či editace, podporu *lazy loadu* i parametry vzhledu. Interní stav se stará o informace o načítání, chybách, filtrech, řazení nebo vybraných řádcích.

Vykreslování je řešeno sadou dílčích metod. Hlavičku tabulky s indikátory řazení a případnými filtry zajišťuje metoda *_renderHeader*, jednotlivé řádky generuje *_renderBody* a stránkovací ovládání *_renderPagination*. Stav načítání nebo chyb se zobrazuje pomocí metod *_renderLoading* a *_renderError*. Načítání dat může probíhat jak z lokálních zdrojů předaných přímo přes vlastnost data, tak i z externích API prostřednictvím *dataUrl*. Pokud je aktivní *lazy loading*, komponenta při dosažení definovaného prahu načítá další data. Filtrování funguje tak, že zadané hodnoty se ukládají do interní struktury a metoda *_applyFilters* aplikuje podmínky na dataset. Řazení je implementováno pomocí *_handleSort* a *_applySort*, které zajišťují stabilní uspořádání dat.

Práce se stránkováním je řešena výpočtem aktuální stránky prostřednictvím *_getPageData* a změnou hodnot přes *_changePage* nebo *_changePageSize*. Výběr řádků probíhá metodou *_handleRowSelect* a pro hromadné operace jsou určeny funkce *_handleSelectAll* či *getSelectedRows*. Editace buněk je podporována pouze tam, kde je nastavena vlastnost *editable*. Metoda *_handleCellEdit* přepíná buňku do editačního režimu a *_renderCellEditor* generuje vhodný vstupní prvek, například textové pole, číselný vstup nebo datum. Změny jsou potvrzovány metodou *_handleEditComplete*, která aktualizuje data a vyšle událost *table-cell-edited*, případně se dají zrušit pomocí klávesy Escape.

3.2 Události, API a další aspekty

Komponenta <my-table> komunikuje s okolní aplikací pomocí událostí. Mezi nejdůležitější patří table-cell-edited pro potvrzení editace, table-selection-changed pro změny výběru řádků a table-data-loaded či table-data-error pro stav načítání. Kromě toho jsou podporovány i události jako table-sort-changed nebo table-filter-changed. Vývojáři mají k dispozici také veřejné API, které umožňuje přidávat a aktualizovat data (setData, appendData, updateRow, deleteRow), pracovat s výběrem (getSelectedRows, selectRow,

deselectRow, selectAll, deselectAll) nebo nastavovat řazení a filtrování.

Součástí implementace je také práce s chybovými a načítacími stavy. Při načítání je uživateli zobrazován spinner či skeleton a interakce jsou omezené. V případě chyby se zobrazí chybová hláška s možností opakování akce. Vzhled komponenty je přizpůsobitelný pomocí CSS proměnných a volitelných tříd, které určují například pruhování, orámování, zvýraznění řádků nebo kompaktní zobrazení. Velikost tabulky lze nastavit přes vlastnosti height, maxHeight, width a maxWidth.

Z hlediska výkonu komponenta podporuje postupné načítání dat a pracuje s transformacemi datasetu tak, aby minimalizovala zbytečné přerenderování. Lit navíc seskupuje změny vlastností do jednoho cyklu, což dále zlepšuje efektivitu. Důraz byl kladen také na přístupnost a lokalizaci – editační režim respektuje fokus i klávesové zkratky a texty UI lze přizpůsobit prostřednictvím vlastností locale a messages.

3.3 Ukázky klíčových částí kódu

3.3.1 Definice vlastností a výchozích hodnot

Definice vlastností komponenty specifikuje datové typy a zároveň určuje výchozí hodnoty, které jsou inicializovány v konstruktoru. Tento mechanismus zajišťuje, že komponenta je funkční i bez explicitního nastavení ze strany uživatele a umožňuje snadnou konfiguraci při začlenění do aplikace.

```
export class MyTable extends LitElement {
  static styles = [tableStyles];
 static properties = {
    columns: { type: Array }, data: { type: Array }, dataUrl: { type: String },
    pageSize: { type: Number }, currentPage: { type: Number }, pageSizes: { type:
Array },
    sortable: { type: Boolean }, filterable: { type: Boolean }, editable: { type:
Boolean },
    selectable: { type: Boolean }, groupable: { type: Boolean },
    lazyLoad: { type: Boolean }, loadThreshold: { type: Number }, totalItems: {
type: Number },
    height: { type: String }, maxHeight: { type: String }, width: { type: String
}, maxWidth: { type: String },
    striped: { type: Boolean }, bordered: { type: Boolean }, hover: { type:
Boolean }, dense: { type: Boolean },
    stickyHeader: { type: Boolean }
  };

  constructor() {
    super();
    this.columns = []; this.data = []; this.dataUrl = '';
    this.pageSize = 20; this.currentPage = 1; this.pageSizes = [10, 20, 50, 100];
    this.sortable = true; this.filterable = true; this.editable = false;
this.selectable = false; this.groupable = false;
    this.lazyLoad = false; this.loadThreshold = 100; this.totalItems = 0;
```

```
    this.height = 'auto'; this.maxHeight = '600px'; this.width = '100%';
this.maxWidth = '100%';
    this.striped = true; this.bordered = true; this.hover = true;
    // interní stav...
  }
}
```

3.3.2 Render hlavičky, těla a stránkování

Metoda render() zajišťuje sestavení šablony komponenty. Kombinuje kontejner tabulky, indikátory stavů načítání a chyb, hlavičku a tělo tabulky i stránkovací prvky. Výsledkem je ucelená struktura, která je vykreslena do DOM a představuje vizuální výstup komponenty.

```
render() {
  return html`
    <div class="table-container ${this.stickyHeader ? 'sticky-header' : ''}"
style="${/* inline dims */''}">
      ${this._loading ? this._renderLoading() : ''}
      ${this._error ? this._renderError() : ''}

      ${!this._loading && !this._error ? html`
        <div class="table-scroll-wrapper">
          <div class="table-wrapper">
            <table class="${this._getTableClasses()}">
              ${this._renderHeader()}
              ${this._renderBody()}
            </table>
          </div>
        </div>
        ${this._renderPagination()}
      ` : ''}
    </div>`;
}
```

3.3.3 Filtrování a řazení

Filtrování a řazení jsou implementovány jako transformace nad datasetem. Metody _applyFilters a _applySort provádějí vlastní úpravu dat podle nastavených parametrů, zatímco _handleFilter a _handleSort slouží k ukládání vstupů uživatele a vyvolání přepočtu. Tím je zajištěna dynamická reakce komponenty na změny podmínek filtrování a pořadí záznamů.

```
_applyFilters(data) { /* ... vrací gefiltrované pole ... */ }
_applySort(data)    { /* ... stabilní sort dle _sort ... */ }

_handleFilter(e, col) { /* uloží do _filters a přepočítá view */ }
_handleSort(col)      { /* přepne směr, uloží do _sort, přepočítá */ }
```

3.3.4 Editace a události

Editační logika umožňuje změnu hodnot přímo v tabulce. Metoda _handleCellEdit inicializuje stav editace, _renderCellEditor zobrazuje vhodný vstupní prvek podle typu dat a

_handleEditComplete zajišťuje potvrzení úprav, jejich uložení a vyvolání odpovídající události. Tímto způsobem je dosaženo dvoucestné vazby mezi tabulkou a aplikační logikou.

```
_handleCellEdit(rowIndex, col) { /* nastaví _editState */ }
_renderCellEditor(type, value) { /* input/select podle typu */ }
_handleEditComplete(e) {
  // commit, aktualizace this.data, vyvolání události
  this.dispatchEvent(new CustomEvent('table-cell-edited', { detail: { /* row, col,
old/new */ } }));
}
```

3.3.5 Výběr a veřejné API

Výběr řádků je realizován prostřednictvím správy množiny vybraných indexů. Metoda _handleRowSelect aktualizuje tuto množinu a vyvolává událost informující o změně výběru. Funkce getSelectedRows pak poskytuje přístup k datům vybraných záznamů a tvoří součást veřejného API, které je určeno pro snadnou integraci komponenty do nadřazených aplikací.

```
_handleRowSelect(index, checked) {
  if (checked) this._selectedRows.add(index); else
this._selectedRows.delete(index);
  this.dispatchEvent(new CustomEvent('table-selection-changed', { detail: {
selected: this.getSelectedRows() }}));
}
getSelectedRows() { return Array.from(this._selectedRows).map(i => this.data[i]);
}
```

Kapitola 3 podrobně demonstrovala implementaci komponenty <my-table>. Bylo popsáno, jak je organizován kód, jak funguje životní cyklus, jak se vykresluje a pracuje s daty. Byly vysvětleny klíčové funkce – filtrování, řazení, stránkování, výběr i editace řádků – a také události a API pro programátory. Tento přístup umožňuje snadnou integraci komponenty do libovolné webové aplikace a její rozšíření podle potřeb uživatelů.

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a implementovat webovou komponentu pro práci s tabulkovými daty, která je univerzální, framework-agnostická a splňuje požadavky moderních webových aplikací. V teoretické části byla představena architektura webových komponent, popsány jejich výhody a omezení a byla provedena rešerše dostupných tabulkových řešení na trhu. Analýza ukázala, že ačkoli existuje mnoho robustních knihoven, jejich nevýhodou bývá velká závislost na konkrétním ekosystému, vyšší

složitost integrace nebo omezená možnost přizpůsobení.

Na základě těchto poznatků byla navržena vlastní komponenta <my-table>, která využívá knihovnu LitElement a technologie Custom Elements a Shadow DOM. Návrh vycházel z funkčních a nefunkčních požadavků definovaných v zadání, které zahrnovaly zejména podporu filtrování, řazení, stránkování, editace, výběru řádků, postupného načítání dat a skrývání/zobrazování sloupců. Součástí návrhu komponenty bylo také řešení API pro vývojáře, které poskytuje přehled vlastností, metodů a událostí. Součásti návrhu komponenty byl také stavový diagram, který popisuje životní cyklus komponenty.

Vytvořená komponenta demonstruje výkonnou a flexibilní tabulkovou komponentu s důrazem na modularitu a přístupnost. Komponenta je schopna pracovat jak s lokálními daty, tak s daty načítanými vzdáleně prostřednictvím API, a podporuje také dvoucestnou vazbu pro snadnou integraci do reaktivních frameworků. Díky oddělení prezentace a logiky je komponenta snadno rozšiřitelná a přizpůsobitelná.

Výsledkem práce je produkčně použitelná webová komponenta, která může být využita v široké škále webových aplikací. Přínosem je nejen samotná implementace, ale i metodika návrhu a dokumentace API, která může posloužit jako vzor pro tvorbu dalších webových komponent.

Do budoucna je možné rozšířit komponentu například o podporu pokročilých funkcí (např. drag & drop pro řádky a sloupce, export do dalších formátů, integraci s real-time zdroji dat přes WebSocket) či hlubší napojení na existující frameworky. Tato práce tak poskytuje dostatečný základ pro další vývoj v oblasti univerzálních, znovupoužitelných a výkonných webových komponent.

POUŽITÁ LITERATURA

[1] A brief history of WebComponents and a tutorial to make yours. *Medium* [online]. 2017-08-2017 [cit. 2024-12-20]. Dostupné z: https://medium.com/apprendre-le-web-avec-lior/a-brief-history-of-webcomponents-and-a-tutorial-to-make-yours-a52d329913e7

[2] The Past, Present, and Future of Web Components. *DEV Community* [online]. 2024-05-12 [cit. 2024-12-20]. Dostupné z: https://dev.to/besworks/the-past-present-and-future-of-web-components-2g43

[3] Introduction to Web Components. *W3C Working Group Note 24 July 2014* [online]. 2014-07-24 [cit. 2024-12-20]. Dostupné z: https://www.w3.org/TR/2014/NOTE-components-intro-20140724/

[4] Polymer Project. *Polymer prouject* [online]. 2015-05-29 [cit 2024-12-20]. Dostupné z: https://www.polymer-project.org/blog/2015-05-29-one-dot-oh

[5] Web Components. *MDN web docs* [online]. 2025-04-10 [cit 2025-04-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_components

[6] Why web components. *State of California Design System* [online]. 2024-02-12 [cit. 2024-12-01]. Dostupné z: https://designsystem.webstandards.ca.gov/why-web-components/index.html

[7] Web Components Vs. Framework Components: What's The Difference? *Smashing Magazine* [online]. 2025-03-17 [cit. 2025-04-12]. Dostupné z: https://www.smashingmagazine.com/2025/03/web-components-vs-framework-components/

[8] Web Components Basics and Performance Benefits. *Medium* [online]. 2020-02-07 [cit. 2024-12-25]. Dostupné z: https://medium.com/%40spkamboj/web-components-basics-and-performance-benefits-f7537c908075

[9] Using CSS custom properties (variables). *MDN web docs* [online]. 2025-04-12 [cit. 2025-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties

[10] ARIA. *MDN web docs* [online]. 2025-05-08 [cit. 2025-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

[11] If Web Components are so great, why am I not using them? *Daverupert* [online]. 2023-07-28. [cit. 2025-03-18]. Dostupné z: https://daverupert.com/2023/07/why-not-webcomponents

[12] Web components in 2021: the Good, the Bad and the Ugly. *DEV Community* [online]. 2021-05-02. [cit. 2025-01-13]. Dostupné z: https://dev.to/emileperron/web-components-in-2021-the-good-the-bad-and-the-ugly-3kg

[13] Are Web Components Dead? *Web Highlights* [online]. 2023-08-14. [cit. 2024-12-05]. Dostupné z: https://web-highlights.com/blog/are-web-components-dead/

[14] The problem with web components. *Adam Silver* [online]. 2019-06-10. [cit. 2024-12-05]. Dostupné z: https://adamsilver.io/blog/the-problem-with-web-components

[15] Evaluating the Role of Web Components in 2024: To Use or Not to Use? *ICT Institute*[online]. 2024-05-06. [cit. 2024-12-05]. Dostupné z: https://ictinstitute.nl/webcomponents-in-2024

[16] Componentizing our React app. *MDN web docs* [online]. 2025-04-11. [cit. 2025-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_components

[17] Getting started with Angular. *MDN web docs* [online]. 2025-04-11. [cit. 2025-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Angular_getting_started

[18] Getting started with Vue. *MDN web docs* [online]. 2025-04-11. [cit. 2025-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Vue_getting_started

[19] Getting started with Svelte. *MDN web docs* [online]. 2025-04-11. [cit. 2025-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Svelte_getting_started

[20] Getting Started. *Lit.dev* [online]. [cit. 2025-03-12]. Dostupné z: https://lit.dev/docs/getting-started/

[21] Getting Started. *StencilDocs* [online]. 2025. [cit. 2025-04-24]. Dostupné z: https://stenciljs.com/docs/getting-started

[22] SkateJS. *GitHub* [online]. 2019-11-06. [cit. 2025-04-24]. Dostupné z: https://github.com/skatejs/skatejs

[23] SAP UI5 Web Components. *SAP UI5 Web Components* [online]. [cit. 2025-04-24]. Dostupné z: https://sap.github.io/ui5-webcomponents/

[24] Vaadin Grid. *Vaadin Documentation* [online]. [cit. 2025-04-24]. Dostupné z: https://vaadin.com/docs/latest/components/grid

[25] AG Grid Documentation. *AG Grid* [online]. [cit. 2025-04-24]. Dostupné z: https://www.ag-grid.com/documentation/

[26] DataGrid - Material UI. *Material UI Documentation* [online]. [cit. 2025-04-24]. Dostupné z: https://mui.com/x/react-data-grid/

[27] Tabulator. *Tabulator Documentation* [online]. [cit. 2025-04-24]. Dostupné z: http://tabulator.info/

[28] REST API Design Best Practices. *REST API Tutorial* [online]. [cit. 2025-04-24]. Dostupné z: https://restfulapi.net/

[29] GraphQL Documentation. *GraphQL* [online]. [cit. 2025-04-24]. Dostupné z: https://graphql.org/learn/

[30] OData Protocol. *OData Documentation* [online]. [cit. 2025-04-24]. Dostupné z: https://www.odata.org/documentation/

[31] WebSocket API. *MDN web docs* [online]. [cit. 2025-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

[32] Responsive Web Design. *A List Apart* [online]. 2010-05-25. [cit. 2025-04-24]. Dostupné z: https://alistapart.com/article/responsive-web-design/

[33] Touch Events. *MDN web docs* [online]. [cit. 2025-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Touch_events

[34] Virtual Scrolling. *Web.dev* [online]. [cit. 2025-04-24]. Dostupné z: https://web.dev/articles/virtualize-lists

[35] WCAG 2.1 Guidelines. *W3C* [online]. 2018-06-05. [cit. 2025-04-24]. Dostupné z: https://www.w3.org/WAI/WCAG21/quickref/

[36] Design Patterns for Web Components. *Google Developers* [online]. [cit. 2025-04-24]. Dostupné z: https://developers.google.com/web/fundamentals/web-components/customelements

[37] Gherkin Reference. *Cucumber Documentation* [online]. [cit. 2025-04-24]. Dostupné z: https://cucumber.io/docs/gherkin/reference/

[38] UML Diagrams. *UML.org* [online]. [cit. 2025-04-24]. Dostupné z: https://www.uml.org/what-is-uml.htm

[39] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[40] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[41] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[42] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[43] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[44] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[45] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[46] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

[47] Obrázek byl vygenerován pří pomoci AI https://chatgpt.com

SEZNAM PŘÍLOH

PŘÍLOHA A – Popis API pro programátory

Příloha obsahuje kompletní popis všech možných vlastnosti, metod a událostí, které mohou využít programátoři.

PŘÍLOHA B – Kompletní zdrojový kód aplikaci

Příloha obsahuje zdrojový kód projektu, využívajícího webovou komponentu – tabulku. Tento projekt obsahuje vytvořené soubory a webovou stránku s příkladem využiti této komponenty.