Универзитет Св. Кирил и Методиј - Скопје

Факултет за информатички науки и компјутерско инженерство

Прв циклус на студии

Проектна задача по предмететот

Напреден веб дизајн

На тема:

Excel Clone in Astro

Ментор:

д-р Бобан Јоксимоски

Кандидат:

Ѓорги Нечовски

број на индекс 211177

Содржина

Вовед

Оваа апликација е развиена за потребите на предметот Напреден веб-дизајн и претставува груба копија на функционалностите на Microsoft Excel, имплементирана со користење на современи веб-технологии. Проектот е изработен со Astro, модерен frontend framework кој овозможува флексибилна интеграција на повеќе frontend frameworks во рамките на една апликација. Во овој случај, апликацијата комбинира React и Vue за да создаде интерактивен и динамичен кориснички интерфејс. Целта на овој проект е да се демонстрира моќта на Astro како алатка за градење комплексни веб-апликации, истовремено овозможувајќи споделување на податоци меѓу различни frontend frameworks и имплементирајќи функционалности слични на оние во Excel.

Овој вовед има за цел да го постави контекстот на проектот, објаснувајќи ја основната идеја и технологиите што се користат. Во следните поглавја, детално ќе се посветеме на Astro како frontend framework, неговите предности, како и на начинот на кој React и Vue работат заедно за да овозможат имплементација на подобна апликација.

Што е Astro

Astro е модерен frontend framework кој се истакнува по својата уникатна способност да комбинира повеќе технологии и пристапи за создавање на брзи, ефикасни и флексибилни веб апликации. За разлика од традиционалните frontend frameworks како React, Vue или Angular, Astro е дизајниран со фокус на **Static Site Generation (SSG)** и **Server-Side Rendering (SSR)**, што овозможува оптимизирани перформанси и подобро корисничко искуство. Главната цел на Astro е да го направи развојот на веб апликации поедноставен, додека истовремено ја намалува количината на JavaScript што се испраќа до прелистувачот, што резултира со побрзо вчитување на страниците.

Клучни карактеристики на Astro

1. **Компонентен пристап со повеќе frameworks** Astro овозможува користење на компоненти од различни frontend frameworks, како што се React, Vue, Svelte и други, во рамките на една апликација. Ова значи дека развивачите можат да ги искористат предностите на различни технологии без да бидат ограничени на еден единствен framework. На пример, во нашиот проект, Astro ни овозможува да користиме React за динамични компоненти и Vue за други делови од корисничкиот интерфејс, при што сите тие работат заедно беспрекорно.

2. **Zero JavaScript by default** Една од најреволуционерните карактеристики на Astro е тоа што стандардно генерира статичен HTML без да вклучува JavaScript, освен ако тоа не е експлицитно побарано. Ова значително ја подобрува брзината на вчитување на страниците и е идеално за апликации каде што перформансите се клучни

3. **Островна архитектура (Island Architecture)** Astro ја воведува концептот на островна архитектура, каде што интерактивните компоненти се третираат како "острови" на динамичност во море од статичен HTML. Ова значи дека само одредени делови од страницата се рендерираат динамично, додека остатокот останува

статичен, што ги намалува ресурсите потребни за работа на апликацијата. Во контекст на нашиот Excel проект, оваа архитектура ни овозможува да ги оптимизираме перформансите на табеларниот приказ, каде што само интерактивните делови (како внесување податоци или пресметки) се хидрирани.

4. **Флексибилност во рендерирањето** Astro поддржува повеќе начини на рендерирање, вклучувајќи Static Site Generation (SSG), Server-Side Rendering (SSR) и Client-Side Rendering (CSR). Ова му дава на развивачот слобода да избере најсоодветен пристап за секој дел од апликацијата. На пример, во нашиот проект, можеме да користиме SSG за статични делови од апликацијата, како документацијата, и SSR за динамични делови, како табелата на Excel.

Зошто Astro за овој проект?

Изборот на Astro за имплементација на груба копија на Excel беше мотивиран од неговата способност да комбинира повеќе frontend frameworks и да обезбеди високи перформанси. Со Astro, можеме да ги искористиме предностите на React за динамични пресметки и Vue за визуелни компоненти, додека истовремено ја одржуваме апликацијата лесна и брза. Дополнително, островната архитектура ни овозможува да ги оптимизираме перформансите на табелата, каде што само одредени ќелии или функции се интерактивни, додека остатокот од интерфејсот останува статичен.

Astro, исто така, го олеснува развојот со својата едноставна структура на проекти и интуитивен API. Ова е особено важно за едукативни проекти, каде што фокусот е на учење и експериментирање со нови технологии. Преку користењето на Astro, можеме да покажеме како модерните веб-технологии можат да се комбинираат за да се создаде функционална апликација со минимален напор и максимална ефикасност.

Зошто одбрав React и Vue за овој проект?

При изборот на технологии за овој проект, одлуката да се користат React и Vue беше мотивирана од комбинација на практични и едукативни причини. Овие два frontend frameworks беа избрани за да се искористат нивните индивидуални силни страни, да се овозможи флексибилност во развојот и да се постигне рамнотежа меѓу користење на познати алатки и истражување на нови технологии.

Првата причина за изборот на React е моето постоечко искуство со овој framework. React е добро познат по својата моќна екосистема, компонентен пристап и флексибилност при креирање на динамични кориснички интерфејси. Со оглед на тоа што веќе сум запознаен со неговите концепти, како што се функционалните компоненти, hooks и управувањето со состојбата, користењето на React ми овозможи да работам ефикасно и да се фокусирам на имплементацијата на сложените функционалности на апликацијата, како што се пресметките и интерактивните елементи на Excel. Оваа познатост ми даде сигурност дека можам брзо да изградам стабилни и сигурни компоненти за проектот.

Од друга страна, Vue беше избран со цел да се прошират моите вештини и да се истражи нов frontend framework. Vue е познат по својата едноставност, интуитивен API и леснотијата на учење, што го прави идеален за развивачи кои сакаат да експериментираат со нова технологија. Со вклучувањето на Vue во проектот, имав можност да научам како функционира неговиот реактивен систем, како се структурираат неговите компоненти и како може да се интегрира со други технологии, како што е Astro. Ова не само што го збогати моето искуство, туку и ми овозможи да ги споредам предностите и разликите меѓу React и Vue во реален проект.

Изработка на апликацијата

Поделба на компоненти

За изработката на оваа апликација, одлучив да ја поделам на два главни дела со цел да ја организирам структурата и да ја олеснам имплементацијата. Првиот дел е Grid, односно табеларниот приказ каде што се наоѓаат ќелиите на Excel-подобната апликација. Вториот дел е менито за управување, кое служи за контрола и манипулација со податоците во ќелиите. Оваа поделба овозможи јасна сепарација на функционалностите, при што секој дел беше развиен со специфични технологии и пристапи за да се постигне оптимална ефикасност и модуларност.

Grid: Табеларен приказ во React

Grid делот, кој го претставува срцето на апликацијата, е целосно имплементиран во React. Изборот на React за овој дел беше природен поради неговата способност да управува со динамични и комплексни кориснички интерфејси. Grid-от е составен од мрежа од ќелии, каде што секоја ќелија може да содржи податоци внесени од корисникот, формули или резултати од пресметки. React ни овозможи ефикасно да го рендерираме овој табеларен приказ, да управуваме со состојбата на ќелиите и да овозможиме интерактивност, како што е селекција на ќелии или внесување податоци.

Користењето на React за Grid-от, исто така, го олесни управувањето со голем број компоненти, особено кога станува збор за рендерирање на големи табели со многу редици и колони. Со помош на React-ови техники за оптимизација, како мемоизација и виртуелно рендерирање, успеав да обезбедам брз и респонзивен кориснички интерфејс, дури и при работа со поголеми сетови на податоци.

Мени за управување: Хибриден пристап со React и Vue

Менито за управување е втор главен дел од апликацијата и е поделено на повеќе компоненти, од кои дел се имплементирани во React, а дел во Vue. Оваа хибридна имплементација беше мотивирана од желбата да се искористат предностите на двата

frontend frameworks и да се демонстрира способноста на Astro да ги интегрира. На пример, React компонентите во менито беа користени за делови каде што беше потребна поголема интерактивност и комплексна логика, како што се копчиња за извршување на пресметки или форматирање на ќелиите. Од друга страна, Vue компонентите беа искористени за визуелно ориентирани делови, како што се менија за избор на стилови или алатници, поради нивната едноставност и интуитивен синтаксис.

Оваа поделба не само што ја направи апликацијата пофлексибилна, туку и ми овозможи да експериментирам со различни пристапи за развој во рамките на истиот проект. Astro одигра клучна улога во оваа интеграција, овозможувајќи React и Vue компонентите да коегзистираат во една апликација без конфликти.

Проблемот со споделување на состојба меѓу React и Vue

Еден од најголемите предизвици во текот на развојот беше наоѓањето начин за споделување на состојба на податоци меѓу React и Vue компонентите. Бидејќи Grid-от и делови од менито зависат од истите податоци (на пример, содржината на ќелиите или селектираните ќелии), беше неопходно да се имплементира механизам за нивна синхронизација. Во почетокот, се обидов да пронајдам начин React и Vue да разменуваат податоци директно, на пример, преку проследување на props. Меѓутоа, брзо станав свесен дека React и Vue не можат директно да комуницираат еден со друг, бидејќи Astro ги повикува нивните компоненти независно, што го ограничува директното проследување на податоци меѓу нив.

Следниот обид беше да се искористи заеднички Redux store за управување со состојбата. Redux е популарен за управување со состојба во React апликации, па се надевав дека би можел да се примени и за Vue. Меѓутоа, открив дека една Redux инстанца е тесно поврзана со специфичниот framework во кој се користи, што значи дека не може да се споделува меѓу React и Vue без значителни компликации. Ова ме наведе да побарам поедноставно и поуниверзално решение.

Решение: Чист JavaScript Store со Listener-Observer однесување

По неколку неуспешни обиди, заклучив дека состојбата на податоците мора да се чува надвор од React и Vue, во чист JavaScript. За таа цел, развив сопствен store за управување со податоците, кој имплементира едноставно, но ефективно Listener-Observer однесување. Овој store служи како централизирано место за складирање на сите критични податоци на апликацијата и овозможува React и Vue компонентите да се синхронизираат преку него.

Структура на Store

Store-от е имплементиран како JavaScript објект кој ги чува следните податоци:

- grid: Низа од објекти што ги претставуваат ќелиите во табелата, вклучувајќи ги нивните вредности, формули и други метаподатоци.
- numRows: Бројот на редици во Grid-от, што овозможува динамичко прилагодување на големината на табелата.
- selectedCells: Листа на селектирани ќелии, идентификувани преку нивните ред и колона, што се користи за следење на корисничките интеракции.
- listeners: Низа од функции што се повикуваат кога состојбата на store-от се менува, овозможувајќи реактивност.

Предности на овој пристап

- Универзалност: Store-от е напишан во чист JavaScript, што го прави независен од специфичните frameworks и лесно достапен за React и Vue.
- Едноставност: Listener-Observer моделот е лесен за имплементација и разбирлив, што го прави идеален за едукативни проекти.
- Скалабилност: Store-от може лесно да се прошири за да поддржува нови податоци или функционалности, како што се форматирање на ќелии или историја на промени.

- Реактивност: Автоматското известување на слушателите овозможува беспрекорна синхронизација меѓу компонентите.

```
type SelectedCell = {
 row: number;
 col: number;
};

type Listener = () => void;

export const state = {
 grid: [] as Cell[],
 numRows: 0,
 selectedCells: [] as SelectedCell[],
 listeners: [] as Listener[],

 setGrid(newGrid: Cell[]) {
  this.grid = newGrid;
  this.notifyListeners();
 },

 setNumRows(newNumRows: number) {
  this.numRows = newNumRows;
  this.notifyListeners();
 },

 setSelectedCells(newSelectedCells: SelectedCell[]) {
  this.selectedCells = newSelectedCells;
  this.notifyListeners();
 },

 subscribe(listener: Listener) {
  this.listeners.push(listener);
 },

 notifyListeners() {
  this.listeners.forEach((listener) => listener());
 },
};
```

Изработка на потребни модели

За потребите на оваа апликација, беше создаден само еден основен модел наречен Cell, кој ги опфаќа сите неопходни податоци за функционирање на ќелиIите во Excel-подобната апликација. Овој модел е клучен за управување со податоците во табеларниот приказ и овозможува ефикасно складирање и манипулација со информациите за секоја ќелија во Grid-от. Моделот Cell е дизајниран да биде флексибилен и да ги задоволи потребите за функционалностите на апликацијата, вклучувајќи внесување на податоци, пресметки преку формули и стилизирање на ќелиите.

Структура на моделот Cell

Моделот **Cell** содржи неколку клучни атрибути кои овозможуваат управување со податоците и визуелниот приказ на ќелиите:

- **value**: Вистинската вредност на ќелијата, која може да биде текст, број или формула (на пример, sum(A1+B1)=.
- **displayValue**: Вредноста што се прикажува во ќелијата, односно резултатот од пресметката доколку ќелијата содржи формула, или истата вредност како value доколку нема формула.
- **row**: Редниот број на ќелијата во Grid-от, што ја означува нејзината позиција по вертикала.
- **col**: Колоната на ќелијата во Grid-от, што ја означува нејзината позиција по хоризонтала.
- **style**: Објект кој содржи информации за стилизирање на ќелијата, како што се боја на позадината, боја на текстот, големина на фонтот, подебелина (bold) или други визуелни атрибути.

```
export interface ICellStyle {
  width: string | undefined;
  height: string | undefined;
```

```
    backgroundColor: string | undefined;
    border: string | undefined;
    padding: string | undefined;
    textAlign: TextAlign | undefined;
    justifyContent?: string;
    alignItems?: string;
    fontSize?: string;
    fontWeight?: string;
    fontFamily?: string;
    fontStyle?: string;
    textDecoration?: string;
    color?: string;
  }
  export class Cell {
    constructor(
      public realValue: any,
      public displayValue: any,
      public row: number,
      public col: number,
      public disabled: boolean,
      public roundNumbers?: number,
      public styles?: ICellStyle | undefined,
    ) {}
  }
```

Изработка на Grid

Сега ќе зборуваме за најважната компонента, односно срцето на апликацијата – компонентата Grid, имплементирана во React. За почеток, потребно е да се генерираат почетните ќелии. Ако направиме споредба со Excel, таму колоните се означени со букви (A, B, C...), додека редиците се обележани со броеви (1, 2, 3...). Како го постигнуваме ова? Имаме среќа, бидејќи можеме да ја искористиме ASCII вредноста на буквите за да ги претвориме во броеви. Со едноставно одземање на 65 од ASCII вредноста на буквата (на пример, 'A' = 65), добиваме соодветна нумеричка вредност (A = 0, B = 1, итн.). Ова значи дека генерирањето на ќелиите нема да претставува проблем.

Потребни Hooks

За потребите на оваа Grid компонента, со цел да имаме добро организиран и модуларен код, развив четири custom React hooks:

1. **useCellSelection**: Овој hook се користи за селектирање на ќелии преку кликнување и влечење (click and drag). Овозможува истовремено селектирање на повеќе ќелии, што е корисно кога сакаме да извршиме промени на повеќе ќелии одеднаш, како на пример промена на вредности или форматирање.

2. **useGridResize**: Овој hook се активира кога сакаме да ја промениме големината на одреден ред или колона. Преку кликнување и влечење (click and drag), корисникот може динамички да ја прилагоди ширината на колоните или висината на редовите.

3. **useGridScroll**: Овој hook е задолжен за генерирање на нови ќелии кога корисникот скрола надолу или надесно. Наместо Grid-от да остане без ќелии при скролање, овој hook динамички додава нови редици и колони, обезбедувајќи непрекинато корисничко искуство.

4. **useGridState**: Овој hook е централниот менаџер на состојбата на Grid-от. Тој ги чува сите податоци поврзани со ќелиите, вклучувајќи ги нивните вредности, изглед (стилови), формули и други метаподатоци. Со овој hook, Grid-от станува единствен извор на вистина за сите податоци во апликацијата.

Потребни React ефекти

Компонентата **Grid** користи три hooks, што може да изгледа многу и да привлече коментари од поискусен колега при преглед на Merge Request. Сепак, во овој случај, сите три се неопходни и имаат јасно дефинирана улога.

1. Првиот useEffect: Овој ефект следи секое кликнување на корисникот. Неговата цел е да утврди дали е започната селекција на повеќе ќелии (на пример, преку кликнување и влечење). Ако корисникот селектира повеќе ќелии, овој ефект го регистрира тоа и овозможува соодветна обработка на селекцијата.

2. Вториот useEffect: Се активира само кога нема активна селекција на повеќе ќелии. Овој ефект е задолжен за обработка на промени во вредноста на една ќелија преку внесување од тастатура. На пример, кога корисникот уредува содржина на една ќелија, овој ефект ја ажурира нејзината состојба.

3. Третиот useEffect: Овој ефект следи секој внес од тастатура и проверува дали тековната вредност на ќелијата завршува со знакот Ако условот е исполнет, ефектот активира пресметка на формула, овозможувајќи динамичка евалуација на формули слични на оние во Excel.

Context Menu

Компонентата Grid вклучува и прилагодено однесување за десниот клик, со што се заменува стандардното мени на пребарувачот. Кога корисникот кликне со десен клик на одредена ќелија, наместо да се прикаже контекстуалното мени на пребарувачот, се отвора сопствено мени. Ова мени нуди опции за форматирање, како на пример избор на број на децимални места за заокружување на нумеричките вредности во селектираната ќелија или ќелии.

Изработка на мени за контрола со ќелии

Менито за управување во оваа Excel-подобна апликација е составено од неколку клучни компоненти кои, иако не се особено сложени, играат суштинска улога во обезбедувањето на функционалностите за контрола и манипулација со Grid-от. Овие компоненти се дизајнирани да бидат интуитивни, со јасни и описни имиња, овозможувајќи им на корисниците лесно да ги користат за форматирање, управување и прилагодување на податоците во табелата. Компонентите се имплементирани делумно во React и делумно во Vue, искористувајќи ја флексибилноста на Astro за нивна беспрекорна интеграција. Тие вклучуваат:

- Color Picker
- Alignment Buttons
- Font Settings
- CSV Controls
- Help

Color Picker

Компонентата Color Picker е изработена во React е дел од менито за управување во оваа Excel подобна апликација и игра клучна улога во овозможувањето на корисниците да го прилагодат визуелниот изглед на ќелиите во Grid-от. Оваа компонента вклучува два одделни input елементи, секој со специфична функција за манипулација со стиловите на селектираните ќелии:

- Првиот input ја менува бојата на позадината на ќелиите.
- Вториот input ја менува бојата на текстот (буквите) во ќелиите.

Важно е да се напомене дека овие input елементи не влијаат на вредностите на ќелиите (т.е. value или displayValue во моделот Cell), туку исклучиво на нивните визуелни атрибути, односно на својството style дефинирано во моделот Cell. Како што беше споменато во претходните поглавја, секоја ќелија содржи објект style, кој ги складира сите можни стилски промени, како што се боја на позадината, боја на текстот, големина на фонтот и други.

Color Picker компонента

Alignment Buttons

Компонентата Alignment Buttons е првата Vue компонента во менито за управување на оваа Excel подобна апликација. Таа им овозможува на корисниците да го прилагодат хоризонталното подредување на текстот во селектираните ќелии, нудејќи три опции: лево, централно и десно подредување. Иако е едноставна по својата имплементација, оваа компонента е суштинска за подобрување на визуелната организација и читливоста на податоците во Grid-от. Дизајнирана е да биде интуитивна, составена од три копчиња кои директно го менуваат својството style во моделот Cell, овозможувајќи брзи и прецизни промени во стилот на ќелиите.

Компонентата Alignment Buttons се состои од три копчиња, секое поврзано со една од следните опции за подредување:

- Лево подредување: Текстот во ќелијата се позиционира кон левата страна.
- Централно подредување: Текстот се позиционира во средината на ќелијата.
- Десно подредување: Текстот се позиционира кон десната страна.

Font Settings

Компонентата Font Settings е React компонента дел од менито за управување во оваа Excel-подобна апликација. Таа е задолжена за прилагодување на визуелните атрибути на текстот во ќелиите на Grid-от, овозможувајќи им на корисниците да го персонализираат изгледот на податоците. Оваа компонента нуди неколку можности за форматирање на фонтот, вклучувајќи промена на големината на текстот, избор на тип на фонт и примена на стилови како подебел (bold), курзив (italic) и подвлечен (underlined) текст. Со овие функционалности, Font Settings придонесува за подобрување на читливоста и визуелната организација на табелата, правејќи ја апликацијата попрофесионална и прилагодлива.

CSV Controls

Компонентата CSV Controls е една од поинтересните и технички предизвикувачки компоненти во менито за управување на оваа Excel-подобна апликација. Имплементирана во Vue, оваа компонента овозможува две клучни функционалности: експортирање на податоците од Grid-от во CSV датотека и импортирање на податоци од CSV датотека во табелата. Иако на прв поглед изгледа едноставно со само две копчиња Export и Import CSV. Нејзината имплементација бараше значителна употреба на Vanilla JavaScript, особено за работа со Blob класата и мануелно управување со генерирањето и читањето на датотеки, вклучувајќи прецизно ракување со енкодирањето. Ова ја направи компонентата изненадувачки комплексна,

но и многу вредна за функционалноста на апликацијата.

```
const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });
const link = document.createElement('a');
const url = URL.createObjectURL(blob);
link.setAttribute('href', url);
link.setAttribute('download', 'spreadsheet.csv');
document.body.appendChild(link);
link.click();
document.body.removeChild(link);
URL.revokeObjectURL(url);
```

Дел од кодот за Export на Grid

Help component

Компонентата Help е последната компонента во менито за управување на оваа апликација. Иако е едноставна во својата имплементација, таа игра важна улога во подобрувањето на корисничкото искуство, обезбедувајќи упатства и информации за користење на апликацијата. Имплементирана во Vue, оваа компонента е минималистичка и не вклучува сложена логика или интерактивност, што ја прави идеален кандидат за потенцијална реимплементација директно во Astro датотека, искористувајќи ја целосната моќ на Astro за статички генерирани содржини без JavaScript.

Имплементирани Excel формули

Како клон на Excel, оваа апликација мора да поддржува автоматизирани пресметки преку формули, што е една од клучните функционалности за корисниците. Формулите овозможуваат динамично пресметување на вредности врз основа на податоците во ќелиите, правејќи ја апликацијата практична и моќна за обработка на податоци. За да се имплементираат формулите, беше развиен систем кој ги обработува корисничките внесови во специфичен формат, ги парсира и ги извршува соодветните пресметки. Овој систем е интегриран со React компонентите на Grid-от и користи централизирана датотека за управување со формулите преку switch-case логика.

Формат на формулите

За да се пресмета формула, корисникот мора да ја внесе во ќелија во следниот формат:

- Формулата започнува со името на формулата (на пример, sum, average, min, max) и проследена со знакот =.
- По името на формулата, во мали загради () се наведуваат ќелиите или опсегот на ќелии врз кои ќе се изврши пресметката.
- Опсегот може да се специфицира на два начина:

1. Експлицитно наведување на ќелии: Секоја ќелија се наведува одделно, одделена со запирка (на пример, sum(A1,A5,C1,C5)=).
2. Опсег со две точки: Се наведуваат првата и последната ќелија од опсегот, одделени со : (на пример, sum(A1:C5)=), што го вклучува целиот правоаголен опсег меѓу овие две ќелии.

Овој формат е интуитивен и сличен на оној што се користи во Microsoft Excel, што го олеснува користењето за корисниците запознаени со такви апликации.

За да работи оваа функционалност ако се сеќавате погоре спомнавме дека колоните се зачувуваат со помош на букви, па потребно е да се најди која колона одговара на која ќелија преку бројка, за тоа имаме 2 помошни функции parseFormula и cellReferenceToIndices.

Функцијата parseFormula користи регуларен израз (/[A-Z]+\d+/g) за да извлече референци на ќелии (на пример, A1, B2) од стринг на формула и ги враќа како низа. Регуларниот израз бара една или повеќе буквио бара комбинација од букви проследени со бројки. Ако нема совпаѓања, враќа празна низа. Функцијата cellReferenceToIndices конвертира референца на ќелија (на пример, A1 или AB12) во објект со ред (row) и колона (col). Таа ги одделува буквите (за колоната) и бројот (за редот) од референцата. Буквите се обработуваат за да се пресмета индексот на

колоната, каде што секоја буква придонесува за вредност базирана на 26 (на пример, A=1, B=2, ..., Z=26, AA=27). За секоја буква, се множи тековниот резултат со 26 и се додава нејзината вредност (од 1 до 26, каде A е 1). На крај, се одзема 1 од редот за да се прилагоди за индексирање од 0. Овие функции се клучни за парсирање и обработка на референци на ќелии во формули во Excel-подобната апликација. Тие овозможуваат прецизно мапирање на референците во координати за работа со Grid-от.

Проблеми со крајната апликација

И покрај функционалноста на оваа Excel-подобна апликација, постојат два главни проблеми кои влијаат на нејзиното користење и треба да се земат предвид:

1. Ограничено селектирање на колони. Функционалноста за селектирање на повеќе колони е ограничена и работи само кога селекцијата се врши од лево-горе кон долу-десно. Ова значи дека селектирањето во други насоки (на пример, од десно кон лево или од долу кон горе) не функционира како што е очекувано, што може да го ограничи корисничкото искуство при работа со големи табели.

2. Компатибилност со прелистувачи. Апликацијата беше првенствено развивана и тестирана на прелистувачи базирани на Chromium што резултираше со недоволно тестирање на други прелистувачи, особено на Firefox. На Firefox, селектирањето на повеќе колони не е можно поради проблеми со имплементацијата, што ја ограничува употребата на апликацијата на овој прелистувач. Затоа, се препорачува користење на Chromium-базирани прелистувачи за оптимално искуство.

Овие проблеми, иако значајни, не ја намалуваат целокупната вредност на апликацијата, но укажуваат на области за подобрување во идни итерации, особено во однос на поддршката за различни насоки на селекција и компатибилноста со повеќе прелистувачи.

Заклучок

Оваа Excel апликација, развиена за предметот Напреден веб-дизајн, успешно демонстрира иновативна употреба на Astro за интеграција на React и Vue во рамките на еден проект. Преку поделбата на функционалностите на Grid (имплементиран во React) и менито за управување (комбинација од React и Vue), апликацијата овозможува динамичен табеларен приказ со поддршка за формули, стилови и CSV операции. Централизираниот store со Listener-Observer однесување овозможи беспрекорна комуникација меѓу различните frontend frameworks, решавајќи го предизвикот за споделување на состојба. Моделот Cell, со своите атрибути за вредности и стилови, обезбеди флексибилна основа за управување со податоците, додека компонентите како Color Picker, Alignment Buttons, Font Settings, CSV Controls и Help го збогатија корисничкото искуство. Имплементацијата на формули, со поддршка за опсези и индивидуални ќелии, ја направи апликацијата практична за автоматизирани пресметки. Сепак, ограничувањата како селектирањето на колони само во една насока и проблемите со компатибилноста на Firefox укажуваат на простор за подобрување. Генерално, проектот ја истакнува моќта на Astro за комбинирање на повеќе технологии, нудејќи вредна студија на случај за модерен веб-развој и поставувајќи солидна основа за идни надградби.

Главната **index.astro** датотека служи како централна точка за интеграција на сите компоненти во оваа Excel апликација, овозможувајќи нивно заедничко функционирање во рамките на Astro frontend framework-от. Оваа датотека ги повикува сите клучни компоненти Grid и компонентите од менито за управување (Color Picker, Alignment Buttons, Font Settings, CSV Controls и Help) демонстрирајќи ја ефикасноста на модуларниот пристап во развојот. Може да се забележи дека кодот во главната датотека е минимален и јасен, што е резултат на добро осмислената поделба на компонентите и нивните функционалности. Секоја компонента е независно развиена со специфична улога, што ја прави апликацијата лесна за одржување и проширување. Овој пристап ја истакнува моќта на Astro да оркестрира React и Vue компоненти,

обезбедувајќи беспрекорна интеграција и оптимални перформанси. Главната Astro датотека, со својата концизна структура, е доказ за успешната организација на проектот и ефективното искористување на модуларноста во модерниот веб-развој.

```
const cells = createInitialGrid(50, 25);
---
<div class='grid-container'>
  <div class='controls-row'>
    <ColorPicker client:load />
    <AlignmentButtons client:load />
    <FontSettings client:load />
    <CSVControls client:load />
    <Help client:load />
  </div>
  <CellDetails client:load />
  <div class='grid-wrapper'>
    <Grid client:load cells={cells} />
  </div>
</div>
```