Универзитет „Св. Кирил и Методиј" во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Advanced web design

Project Work: Expense Tracker

Researchers: Arijan Berisha 211513

Dorant Tateshi 211224

Date: 07.09.2024

Location: Skopje

Mentors: Boban Joksimoski

Mila Dodevska

Table of Contents

Introduction

Welcome to our comprehensive guide on modern web design and digital banking strategies.
In the most beginning we will guide you through various aspects of creating user-friendly and efficient web interfaces, focusing on key elements that enhance customer experience.

Technologies used in the following project thesis are: Vue.js and Tailwind.
As you can see we provided navigation bar offering options like 'Home', 'Transactions', 'Cards', 'Exchange Office', and 'All tools'.
The layout makes easier navigation through the webpages.

Home Page

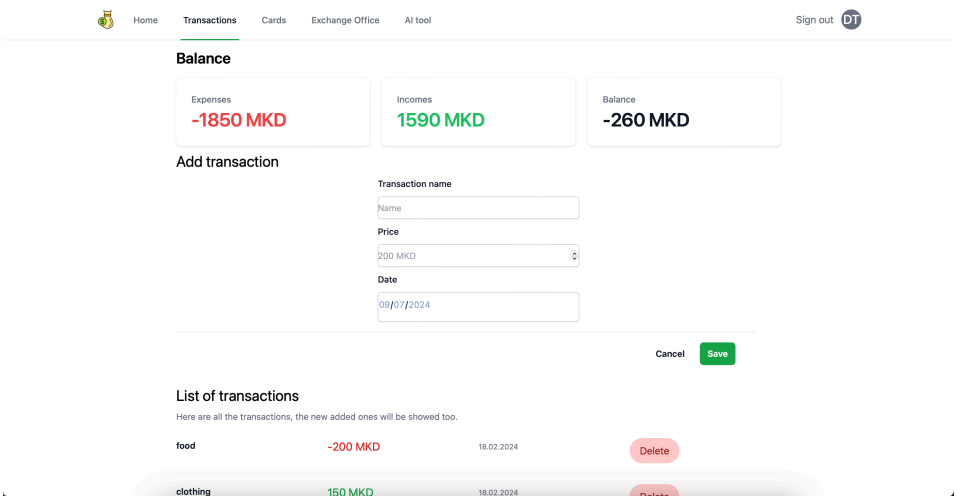
First, our home page was a simple idea as a main page or a dashboard thought.
This is simple template which shows our courage for the thesis and meets the customers around it and makes them feel comfortable as they go through it.

Additionally, talks about the organization such as the location, the leadership etc.

Transactions

The layout features a user interface for a financial management application. It includes sections for transactions, balances, cards, exchange office, and a tool. There are three main balance figures: one showing an expense in red color, another displaying an income of in green, and the last indicating a balance of in black. Below these figures is an option to add a transaction with fields for the transaction name, price (with 200 MKD filled in), and date (08/03/2021 entered). At the bottom part of the interface is a list of transactions happened before, and also which will happen in the future.

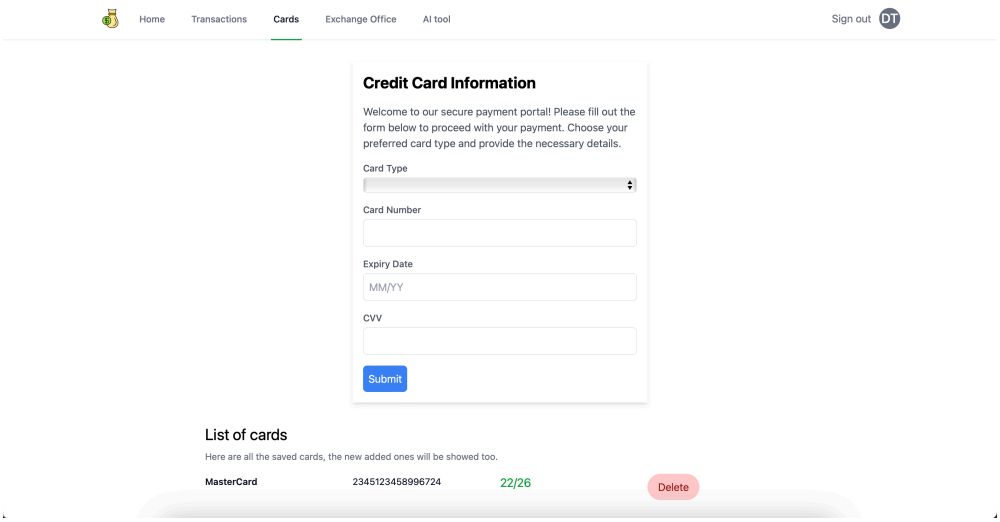
This page is one of the main pages as we express the meaning of Expense Tracker App, which really tells the logged in user, what’s his balance and why his balance is where it is.



Cards

This section of the application shows a secure payment portal where users can input their credit card information. The form allows users to select a card type, input the card number, expiry date, and CVV before submitting the data. Pay attention, if you information is not fully completed and original, the data will not be saved, and you have to redo all the steps in a proper form.

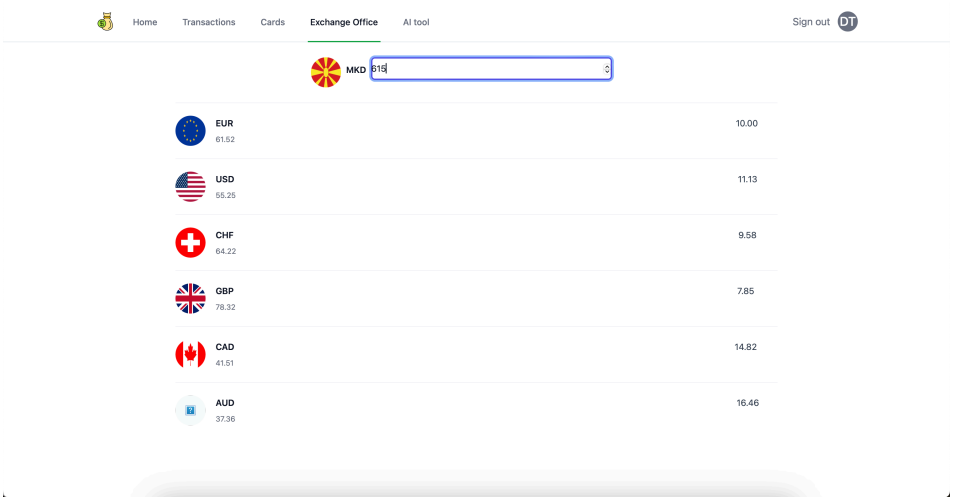
Below the form, there is a list of saved cards, each showing the card type, number, and expiration date, along with an option to delete a saved card. The user-friendly design facilitates both adding and managing payment methods within the application.



Exchange Office

"Exchange Office" feature, allows users to convert currencies. The interface shows the Macedonian denar (MKD) as the selected base currency, with a field to input the amount for conversion. Below, the exchange rates for various currencies such as EUR, USD, CHF, GBP, CAD, and AUD are displayed alongside their respective symbols and flags, providing users with real-time conversion values. This feature simplifies the currency exchange process within the app.

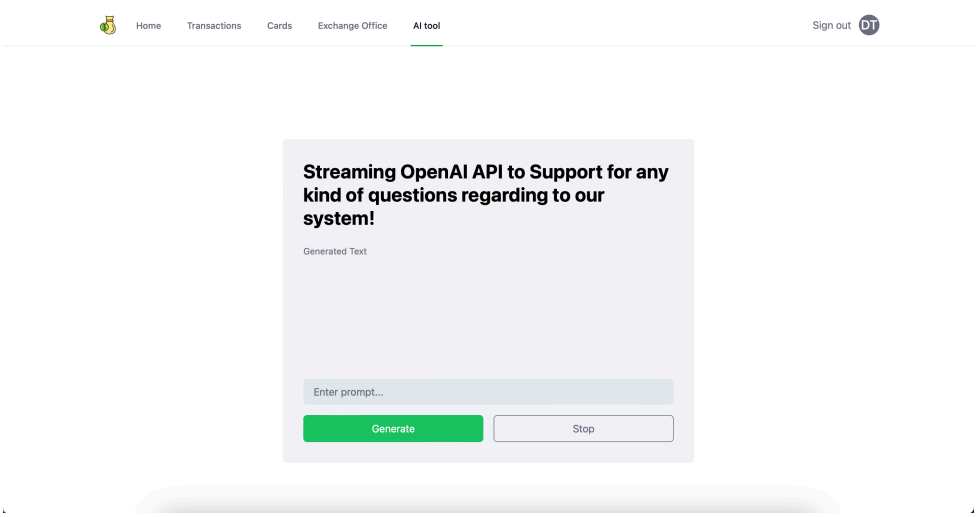
For instance if I input into the field of MKD 615, the EUR will change to 10.00 and the USD to 11.13 and CHF to 9.58 and so on. I will share with you and interaction done here in a single image.



AI Tool

The last but not least, the OpenAI API used for making a possible interaction between artificial intelligence and customer regarding any help he/she needs. As it is last, I think nowadays this section is pretty important for any app out there, because it appears to do the searching and browsing also using safer and softly. This technology is meant to advance in couple of years so we intend to go on time with it and integrate new things.

A customer can simply ask a question about the app, or any difficulties at all, and the AI tool should emphasize the steps needed to be taken and followed for a solution. While pressing Stop, it will Cancel the question, and the customer should refresh it to be able to generate again. This process was meant for safety and security.



REPORTS

We will briefly explain some part of the code, so there is more clarification on the project.

Frist, we will begin the part done together, and then each our parts separated.

As for both Arijan's and Dorant's collaboration code/report:

This part includes only the Home Page

Part of a web page that describes features of a business (possibly for a bank), introduces leadership, and provides contact and location information. The structure is defined using Vue's Composition API, Tailwind CSS for styling, and Heroicons for icons. Below is a detailed breakdown of the components and features used.

```
const features = [
 {
   name: 'Push to deploy',
```

```
    description:
      'Commodo nec sagittis tortor mauris sed...',
    href: '#',
    icon: CloudArrowUpIcon
  },
  {
    name: 'SSL certificates',
    description:
      'Pellentesque enim a commodo...',
    href: '#',
    icon: LockClosedIcon
  },
  {
    name: 'Simple queues',
    description:
      'Pellentesque sit elit congue ante...',
    href: '#',
    icon: ArrowPathIcon
  }
]
```

Purpose: The features array is an object list that stores different business features, each with a name, description, link (href), and an associated icon from Heroicons (e.g., CloudArrowUpIcon, LockClosedIcon).

Use: This data will be dynamically displayed in the template to show key features with their respective icons.

```
const people = [
  {
```

```
    name: 'Leslie Alexander',
    role: 'Co-Founder / CEO',
    imageUrl:
      'https://images.unsplash.com/photo-1494790108377-be9c29b29330?ixlib=rb-1.2.1...'
  }
]
```

Purpose: The people array defines the leadership team, in this case, featuring one person with their name, role, and image.

Use: This will be rendered in the "Meet our leadership" section where leadership members are displayed with their name, role, and picture.

```
<div>
  <h2 class="text-4xl font-bold tracking-tight text-gray-900">
    We built our business on great customer service
  </h2>
  <p class="mt-4 text-gray-500">
    FINKI BANK's strategy focuses on leveraging digital technology...
  </p>
</div>
```

Purpose: This section introduces the business's commitment to customer service. The heading and description are emphasized with large, bold text and supporting content.

```
<dl class="grid max-w-xl grid-cols-1 gap-x-8 gap-y-16 lg:max-w-none lg:grid-cols-3">
  <div v-for="feature in features" :key="feature.name" class="flex flex-col">
    <dt class="flex items-center gap-x-3 text-base font-semibold leading-7 text-gray-900">
```

```
    <component :is="feature.icon" class="h-5 w-5 flex-none text-green-600" aria-hidden="true" />
    {{ feature.name }}
   </dt>
   <dd class="mt-4 flex flex-auto flex-col text-base leading-7 text-gray-600">
    <p class="flex-auto">{{ feature.description }}</p>
    <p class="mt-6">
     <a :href="feature.href" class="text-sm font-semibold leading-6 text-green-600">
      Learn more <span aria-hidden="true">→</span>
     </a>
    </p>
   </dd>
  </div>
</dl>
```

Purpose: Displays business features, dynamically rendered from the features array. Each feature has an associated icon, name, and description, with a "Learn more" link.

```
<ul role="list" class="grid gap-x-8 gap-y-12 sm:grid-cols-2 sm:gap-y-16 xl:col-span-2">
  <li v-for="person in people" :key="person.name">
   <div class="flex items-center gap-x-6">
    <img class="h-16 w-16 rounded-full" :src="person.imageUrl" alt="" />
    <div>
     <h3 class="text-base font-semibold leading-7 tracking-tight text-gray-900">
      {{ person.name }}
     </h3>
     <p class="text-sm font-semibold leading-6 text-green-600">{{ person.role }}</p>
    </div>
```

```
    </div>
  </li>
</ul>
```

Purpose: This section introduces the leadership team. The data from the people array is used to generate a list of people with their photos, names, and roles.

```
<div class="rounded-2xl bg-gray-50 p-10">
  <h3 class="text-base font-semibold leading-7 text-gray-900">Collaborate</h3>
  <dl class="mt-3 space-y-1 text-sm leading-6 text-gray-600">
    <div>
      <dt class="sr-only">Email</dt>
      <dd>
        <a class="font-semibold text-green-600"
href="mailto:collaborate@example.com">collaborate@example.com</a>
      </dd>
    </div>
    <div class="mt-1">
      <dt class="sr-only">Phone number</dt>
      <dd>+1 (555) 905-2345</dd>
    </div>
  </dl>
</div>
```

Purpose: Displays various contact methods like email and phone number for different departments (e.g., Collaboration, Press, Careers). This section allows users to easily get in touch with different teams.

As for Arijan's code/report:

First I will be talking about Cards template, both View and Display:

This section involves creating a secure payment portal that allows users to input and store credit card details, and view or delete previously saved cards. Here's a breakdown of the report based on your provided code:

Main Features:

1. Credit Card Information Form:
    - Card Type: A dropdown menu allows users to select between several card types such as MasterCard, Visa, Apple Pay, PayPal, or Debit Card.
    - Card Number: A text input field allows users to provide their card number.
    - Expiry Date: A field for entering the card's expiration date in MM/YY format.
    - CVV: A text input field for entering the card's security code.
    - Form Submission: The form submits user input by invoking the checkCards() method to validate input and store the card details.
2. Notifications:
    - Success Notification: A notification component (SuccessfullNotification) is displayed when the form is successfully submitted, indicating that the card was registered successfully.
    - Error Notification: An error notification (ErrorNotification) appears when mandatory fields are missing in the form, asking the user to fill out all required fields.
3. Cards Display and Deletion:
    - Cards List: A list of saved credit cards is rendered using the CardsDisplay component. The component shows the type, number, and expiry date of each card.
    - Delete Button: Each card has a delete button that removes the card from the displayed list when clicked, by invoking the handleDelete method and updating the state.

Code Review:

Template Structure:

- The structure is well-organized:
    - The form fields are clearly laid out with appropriate labels and placeholders.
    - Notifications are conditionally rendered (v-if) based on the success or error state of the form submission.
    - The card list and delete buttons are neatly presented in a grid format.

Reactivity:

- **Vue's ref()** is used to manage form inputs (cardType, cardNumber, cardDate, cardCvc) and the card state (cardsState).
- The reactivity of the cardsState ensures that the card list updates in real-time when cards are added or deleted.

Form Validation and Handling:

- The checkCards() method:
    - **Validation**: Ensures that all required fields are filled before proceeding. If any field is missing, it sets postError to true, triggering the error notification.
    - **Card Registration**: Adds the new card details to cardsState when validation passes, and clears the form inputs afterward.
- There is a clear separation between error and success handling, making it easy to manage form feedback.

Card Deletion Logic:

- The deleteTransaction() method in the CardsDisplay component filters out the card by its number and passes the updated list back to the parent via the updateState() method. This successfully removes the selected card from the display.

UI Components:

- **CardsDisplay**:
    - Uses Vue's v-for directive to dynamically generate a list of saved cards.
    - The card type, number, and expiration date are shown, and a delete button is included for each entry.
- **Notifications**:
    - Two notification components handle success and error scenarios, ensuring the user is informed of the action's result (e.g., successful card registration or missing inputs).

Potential Improvements:

1. **Input Validation Enhancements**:
    - **Regex Validation**: Implement regex-based validation for the card number, expiration date, and CVV to ensure proper formats (e.g., card number should be numeric and of a specific length, expiration date should be in MM/YY format).
    - **Error Messages**: Provide more specific error messages for each field when validation fails, such as "Invalid card number" or "Invalid expiration date format."
2. **Security Considerations**:

- **Card Masking**: For security, consider masking the card number (e.g., only showing the last four digits) in the CardsDisplay component.
- **CVV Handling**: The CVV should not be stored or displayed in the CardsDisplay component for security reasons.

3. **Usability Enhancements**:
   - **Clearer Form Reset**: After a successful submission, reset the form fields and set the default card type more explicitly, such as resetting the card type to a default option like MasterCard and adding a reset button if needed.
   - **Disable Submit Button**: Disable the submit button if the form is incomplete or has invalid data, preventing accidental submissions.
4. **Card Management**:
   - **Edit Option**: Provide an option to edit card details instead of just adding or deleting them.
   - **Confirmation for Deletion**: Add a confirmation dialog before deleting a card to avoid accidental deletions.
5. **Mobile Responsiveness**:
   - Ensure the form and card display components are optimized for mobile devices, using responsive design principles with CSS grid and flexbox.

Afterwards I will be talking about implementation of AI Tool:

The following Component represents a user interface for generating text using the OpenAI API, specifically the GPT-3.5-turbo model. The system enables users to input a prompt and generate responses, while also providing functionality to stop the text generation process.

Key Functionalities

1. **User Input and Display**: The UI is straightforward and user-friendly, designed for users to input their prompt and retrieve generated text. It includes an input field where users can type their questions or prompts, and a section where the generated responses are displayed. The UI is responsive, with a clean design that fits most screen sizes, thanks to the use of utility classes from Tailwind CSS.
2. **API Call to OpenAI**: The core functionality revolves around making an API request to OpenAI's gpt-3.5-turbomodel. The generateText function handles the process of sending the request. It starts by setting the isGenerating flag to true, which disables the "Generate" button to prevent multiple submissions. It then sends the user's prompt to the OpenAI API via a POST request:

   const response = await fetch(API_URL, {

```
  method: 'POST',
  headers: {
   'Content-Type': 'application/json',
   Authorization: `Bearer ${API_KEY}`,
  },
  body: JSON.stringify({
   model: 'gpt-3.5-turbo',
   messages: [{ role: 'user', content: prompt.value }],
   stream: true,
  }),
 });
```

The request includes necessary headers like the Authorization header, which contains the user's API key, and the body, which passes the prompt and specifies that streaming is enabled.

3. Handling API Response: After making the API request, the generateText function waits for a response and processes it. If the response contains the expected format, i.e., a valid response from GPT-3.5, the generated text is stored in the generatedText ref and displayed in the UI. If the response format is unexpected, an error message is shown:

```
if (data.choices && data.choices.length > 0 && data.choices[0].message) {
  generatedText.value = data.choices[0].message.content;
} else {
  generatedText.value = 'Error: Unexpected response format.';
}
```

4. Error Handling: If the API request fails due to network issues or an invalid prompt, an error is caught, and the user is informed through a message:

   ```
   generatedText.value = 'Error occurred while generating.';
   ```

This ensures the system does not crash or display confusing messages, maintaining a user-friendly experience even when something goes wrong.

5. **Stopping Generation**: Another feature in this component is the ability to stop the text generation process. The stopText function is provided for users who may want to cancel the operation while it's in progress. This function simply sets the isGenerating flag to false, which disables the "Stop" button and halts the current generation process.
6. **Conditional Button States**: The component also incorporates conditional button states to ensure a seamless user experience. The "Generate" button is disabled while text is being generated to prevent multiple requests, and the "Stop" button is disabled when no generation is taking place.

```
<button @click="generateText" :disabled="isGenerating" class="w-1/2 ...">
  Generate
</button>
<button @click="stopText" :disabled="!isGenerating" class="w-1/2 ...">
  Stop
</button>
```

Potential Improvements

1. **Streaming Response**: While the request includes the stream: true option, the current implementation doesn't utilize WebSockets or any specific streaming mechanism to show intermediate results as they are generated. Implementing true streaming functionality would enhance the user experience by displaying partial results while the full response is still being generated.
2. **Security Best Practices**: Storing the API key directly in the front-end code is not ideal, as it exposes the key to anyone inspecting the website. It would be better to move the API request logic to a back-end server where the API key can be kept securely.

References

https://vuejs.org/guide/introduction.html

https://v2.tailwindcss.com/docs