Державний вищий навчальний заклад
Ужгородський національний університет
Факультет інформаційних технологій
Кафедра програмного забезпечення систем

ЛАБОРАТОРНА РОБОТА №6

Тема: Бізнес логіка.

Виконав:
студент 3 курсу
спеціальності 121 «Інженерія
програмного забезпечення 3»
Лях Іван Іванович

Ужгород 2024

Мета роботи: Реалізувати бізнес логіку проекту.

Завдання до роботи:

1. Реалізувати бізнес логіку проекту.
2. Створити користувацькі хуки для отримання даних з/на сервер та для авторизації використовуючи fetch або axios.

Хід роботи:

1. Бізнес логіка проекту:

Бізнес-логіка – це будь-який код, що керує основними правилами та поведінкою програми, які стосуються обробки даних і реалізації функціональності.

Бізнес логіка проекту побудована на постійній валідації дій користувача, правильному відображенні компонентів та отриманих даних з серверу.

Перед тим, як перейти на головну сторінку проекту, на фронтенд частині йде перевірка на авторизованість користувача в систему:

```
export const isAuthOK = () => {
    const token = localStorage.getItem('token');
    if (!token) {
      return null;
    }
    try {
      const decodedToken = jwtDecode(token);
      const currentTime = Date.now() / 1000;
      if (decodedToken.exp < currentTime) {
        localStorage.removeItem('token');
        return null;
      }
      return decodedToken;
    } catch {
```

```
        localStorage.removeItem('token');
        return null;
      }
    };
```

Ця функція викликається кожен раз, коли потрібно отримати дані користувача та перевірити його на авторизованість, не відправляючи запит на сервер. Таким способом унеможливлюється спроба несанкціонованого доступу до даних проекту, а також збереження даних користувача у стані.

```
    useEffect(() => {
      if (isAuthOK()) {
        navigate('/', { replace: true });
      }
    }, [navigate]);
```

У випадку авторизованості користувача, його перекидає на кореневий шлях веб-застосунку, тобто на домашню сторінку.

```
const applyFilters = () => {
  let updatedEvents = [...events];
  if (searchQuery) {
    const regex = new RegExp(searchQuery, 'i');
    updatedEvents = updatedEvents.filter((event) => {
      const matchesTitle = regex.test(event.title);
      const matchesTags = event.tags.some((tag) => regex.test(tag));
      return matchesTitle || matchesTags;
    });
  }
  if (filterBy === 'created') {
        updatedEvents = updatedEvents.filter((event) => event.author._id === userId);
  } else if (filterBy === 'participating') {
```

```
                updatedEvents = updatedEvents.filter((event) => event.participants.includes(userId));
  }

  updatedEvents.sort((a, b) => {
    if (sortBy === 'date' || sortBy === 'updatedAt') {
      return order === 'asc'
        ? new Date(a[sortBy]) - new Date(b[sortBy])
        : new Date(b[sortBy]) - new Date(a[sortBy]);
    } else if (sortBy === 'participantsCount') {
          return order === 'asc' ? a.maxParticipants - b.maxParticipants : b.maxParticipants - a.maxParticipants;
    }
    return 0;
  });
  setFilteredEvents(updatedEvents);
};
```

Хорошим прикладом для відображення бізнес-логіки проекту є відображення того, як сортуються дані про подію, перед тим як вони відобразяться користувачеві. Тобто в нас є масив подій, користувач хоче відсортувати або відфільтрувати їх, для цього на сторінці подій він обирає відповідний пункт, після чого відбувається сортування або фільтрація даних, після чого вони відображаються у нього на екрані у вигляді списку.

Зазвичай обробник подій у відповідь на дії користувача виглядає так: `const handleSearch = (e) => { setSearchQuery(e.target.value); };`

Приклад відправки даних на сервер:

```
const handleSaveChanges = async () => {
  const validationError = validateFormData();
  if (validationError) {
```

```
            setError(validationError);
            return;
        }
        const formDataToSend = new FormData();
        formDataToSend.append('fullName', formData.fullName);
        formDataToSend.append('bio', formData.bio);
        formDataToSend.append('dateOfBirth', formData.dateOfBirth);
        formDataToSend.append('tags', formData.tags);
        if (selectedFile) {
            formDataToSend.append('avatar', selectedFile);
        }
        console.log('Form data to send:', [...formDataToSend]);
        try {
              const response = await axiosInstance.put(`/users/${userData._id}`,
formDataToSend, {
                headers: {
                   'Content-Type': 'multipart/form-data',
                },
            });
            onSave(response.data);
        } catch (error_) {
             setError(error_.response?.data?.message || 'Помилка при збереженні
даних');
        }
    };
```

Перед відправкою даних на сервер, ми збираємо дані до купи, структуруємо їх відповідно до форми запиту та властивостей полів, які слугують для збереження цих даних. Також присутня обробка помилок, які

можуть виникнути при збої в роботі сервера або у відповідь на неправильно надіслані дані.

2. Хуки для отримання або надсилання даних з/на сервер(у):

Для деяких типів запитів, які потребують специфічних даних, або даних про користувача, було створено окремий об’єкт бібліотеки axios, який додавав в хедер частину запиту jwt токен користувача:

```
const axiosInstance = axios.create({
    baseURL: 'http://localhost:3000/api',
});
axiosInstance.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  },
);
```

Одним із прикладів використання цього хуку є запит на створення події:

```
await axiosInstance.post(`/event/create`, formDataToSend, {
  headers: { 'Content-Type': 'multipart/form-data' },
});
```

Тобто ми вказуємо наш об’єкт, який був раніше створений, шлях для запиту, дані для надсилання та в тілі запиту вказуємо тип контенту, який ми надсилаємо запитом на сервер.

Далі це є запит типу get, який використовується для отримання даних серверу:

```
const response = await axiosInstance.get(`/event/search?sortBy=${sortBy}&order=${order}`);
```

І так далі, тобто логіка надсилання запиту на сервер з фронтенд частини в проекті всюди однаковий. Запити різняться тільки їх типом та даними, які передаються або отримуються з серверу.

Висновок: на даній лабораторній роботі було реалізовано бізнес-логіку проекту та користувацькі хуки для запиту на сервер за допомогою бібліотеки axios. Бізнес-логіка проекту побудована на постійній валідації вмісту сторінки перед її відображенням користувачеві та даних, які користувач хоче надіслати на сервер.