МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Навчально-науковий інститут електричної інженерії
та інформаційних технологій

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ЗВІТ

З ЛАБОРАТОРНИХ РОБІТ

З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

«Frontend-розробка»

Виконав студент групи КН-23-1

Полинько Ігор Миколайович

Перевірив старший викладач кафедри АІС Притчин О. С.

КРЕМЕНЧУК 2025

Лабораторна робота № 4

Тема: Робота з асинхронністю в JavaScript.

Мета: Познайомитись і отримати головні навички роботи з реалізацією асинхронного коду в JavaScript.

Виконання завдання лабораторної роботи:

Створити кілька промісів. Один проміс має повертати дані які будуть надсилатись іншому як параметр. Викликати всі проміси послідовно через синтаксис then. Переписати створений код на синтаксис async await. Обидва варіанти мають бути написані з перевіркою помилок.

lb4var1.jsx:

```
// Лабораторна №4. Варіант з then/catch

const getUser = new Promise((resolve, reject) => {
  setTimeout(() => {
    const success = true;
    if (success) {
      resolve({ id: 1, name: 'Ігор' });
    } else {
      reject('Не вдалося отримати користувача');
    }
  }, 1000);
});

const getPosts = (user) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (!user) {
        reject('Дані користувача відсутні');
      } else {
        resolve([
          { id: 1, title: `Пост користувача ${user.name}` },
          { id: 2, title: `Ще один пост від ${user.name}` },
        ]);
      }
    }, 1000);
  });
};

// Послідовний виклик промісів
getUser
  .then((user) => {
```

```
    console.log('Отримано користувача:', user);
    return getPosts(user);
  })
  .then((posts) => {
    console.log('Отримано пости:', posts);
  })
  .catch((error) => {
    console.error('Помилка:', error);
  })
  .finally(() => {
    console.log('Завершення роботи промісів');
  });
```

lb4var2.jsx:

```
// Лабораторна №4. Варіант з async/await


const getUserAsync = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const success = true;
      if (success) {
        resolve({ id: 1, name: 'Ігор' });
      } else {
        reject('Не вдалося отримати користувача');
      }
    }, 1000);
  });
};

const getPostsAsync = (user) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (!user) {
        reject('Дані користувача відсутні');
      } else {
        resolve([
          { id: 1, title: `Пост користувача ${user.name}` },
          { id: 2, title: `Ще один пост від ${user.name}` },
        ]);
      }
    }, 1000);
  });
};

async function fetchData() {
  try {
    const user = await getUserAsync();
    console.log('Отримано користувача:', user);

    const posts = await getPostsAsync(user);
```

```
    console.log('Отримано пости:', posts);
  } catch (error) {
    console.error('Помилка:', error);
  } finally {
    console.log('Завершення роботи async/await');
  }
}

fetchData();
```

Результат:

```
PS C:\Users\RoRHaT\Documents\GitHub\frontend-polynko-kn-23-1\4\react\lb4> node src/lb4var1.js
Отримано користувача: { id: 1, name: 'Ігор' }
Отримано пости: [
  { id: 1, title: 'Пост користувача Ігор' },
  { id: 2, title: 'Ще один пост від Ігор' }
]
Завершення роботи промісів
```

Рисунок 4.1 – Робота lb4var1

```
PS C:\Users\RoRHaT\Documents\GitHub\frontend-polynko-kn-23-1\4\react\lb4> node src/lb4var2.js
Отримано користувача: { id: 1, name: 'Ігор' }
Отримано пости: [
  { id: 1, title: 'Пост користувача Ігор' },
  { id: 2, title: 'Ще один пост від Ігор' }
]
Завершення роботи async/await
```

Рисунок 4.2 – Робота lb4var2

```
PS C:\Users\RoRHaT\Documents\GitHub\frontend-polynko-kn-23-1\4\react\lb4> node src/lb4var1.js
Помилка: Не вдалося отримати користувача
Завершення роботи промісів
```

Рисунок 4.3 – Робота перевірки на наявність помилки

Висновок: на цій лабораторній роботі ми познайомились і отримали головні навички роботи з реалізацією асинхронного коду в JavaScript.

Контрольні питання:

1. Чи підтримує JS багатопоточність?

Ні, JavaScript однопотоковий. Він виконує код у одному основному потоці, але асинхронні операції (наприклад, setTimeout, запити до сервера) відправляються у Web APIs браузера або Node.js, які працюють окремо. Потім результати повертаються в основний потік через чергу подій (event loop).

2. Для чого в promise функції resolve та reject?

– resolve(value) – викликається, коли операція успішно завершена, і передає результат.

– reject(error) – викликається, коли операція завершилась з помилкою, і передає опис цієї помилки.

3. **Які статуси може мати promise?**

Проміс має три можливі стани:

– pending – очікування (ще виконується);

– fulfilled – виконано успішно (викликано resolve);

– rejected – виконано з помилкою (викликано reject).

4. **Чи можна після then додати ще один блок then?**
Так, можна. then() повертає новий проміс, тому можна створювати ланцюжки промісів:

```
getUser()
  .then(user => getPosts(user))
  .then(posts => console.log(posts))
  .catch(error => console.error(error));
```

Кожен наступний then чекає, поки виконається попередній.

5. Де саме мають розміщуватись слова async та await?

– async ставиться перед визначенням функції, щоб зробити її асинхронною:

– async function fetchData() { ... }

– await ставиться перед викликом асинхронної функції або промісу всередині async-функції:

– const result = await somePromise();

6. **Як написати код для обробки помилок при використанні async/await?**

За допомогою блоку try...catch (а іноді ще й finally):

```
async function fetchData() {
  try {
    const data = await getData();
    console.log('Отримано:', data);
  } catch (error) {
    console.error('Помилка:', error);
  } finally {
    console.log('Завершення');
  }
}
```