Лабораторна робота №2
Патерни програмування
Звіт

Виконав студент групи ІПС-21
Міцкевич Костянтин

Посилання на репозиторій:
https://github.com/Kirolen/Dispersion

У рамках навчального проекту було також виконано лабораторну №2. Було реалізовано 10 патернів, а саме:

Патерни створення:

- *Singleton Pattern* - Гарантує, що клас має лише один екземпляр, і надає глобальну точку доступу до нього.
- *Factory Pattern* - Створює об’єкти через спільний інтерфейс, дозволяючи підкласам вирішувати, який клас інстанціювати.

Структурні патерни:

- *Composite Pattern* - Об’єднує об’єкти у древовидну структуру, щоб працювати з окремими об’єктами та їх композиціями однаково.
- *Facade Pattern* - Надає спрощений інтерфейс до складної системи або набору інтерфейсів.
- *Adapter Pattern* - Перетворює інтерфейс одного класу в інтерфейс, очікуваний клієнтом, дозволяючи взаємодіяти несумісним класам.

Патерни поведінки:

- *Observer Pattern* - Визначає залежність один-до-багатьох, коли зміна стану одного об’єкта автоматично повідомляє і оновлює всі залежні об’єкти.
- *Strategy Pattern* - Визначає сімейство алгоритмів, інкапсулює їх і робить взаємозамінними, дозволяючи змінювати алгоритм під час виконання.
- *Command Pattern* - Інкапсулює запит як об’єкт, дозволяючи параметризувати клієнтів різними запитами, відкладати або логувати їх.
- *State Pattern* - Дозволяє об’єкту змінювати свою поведінку залежно від внутрішнього стану, виглядаючи як об’єкт іншого класу.
- *Middleware Pattern* - Проміжний шар обробки, який перехоплює запити чи події, виконуючи додаткові дії (наприклад, авторизація або логування) перед передачею далі.

Нижче наведені приклади використання патернів.

Патерни створення:

- Singleton Pattern

Приклад: api.js – створення єдиного екземпляра axios для викликів API

```
const api = axios.create({
  baseURL: API_URL,
});
```

- Factory Pattern

Приклад: fileUtils.js - Створення піктограм файлів на основі типу файлу

```
import { FaFilePdf, FaFileWord, FaFileImage, FaFilePowerpoint, FaFileExcel, FaFileAlt } from 'react-icons/fa';

export const getFileIcon = (filename) => {
    const extension = filename.split('.').pop().toLowerCase();

    const iconStyle = { fontSize: "20px" };

    return (
        <i style={iconStyle}>
            {extension === 'pdf' && <FaFilePdf />}
            {['doc', 'docx'].includes(extension) && <FaFileWord />}
            {['jpg', 'jpeg', 'png', 'gif'].includes(extension) && <FaFileImage />}
            {['ppt', 'pptx'].includes(extension) && <FaFilePowerpoint />}
            {['xls', 'xlsx'].includes(extension) && <FaFileExcel />}
            {extension === 'accdb' && <FaFileAlt />}
            {!['pdf', 'doc', 'docx', 'jpg', 'jpeg', 'png', 'gif', 'ppt', 'pptx', 'xls', 'xlsx', 'accdb'].includes(extension) && <FaFileAlt />}
        </i>
    );
};
```

Структурні патерни:

- Adapter Pattern:

Приклад: **authService.js, fileService.js, testService.js та інші Service.js** - перетворює ці зовнішні формати у внутрішній єдиний стандарт, з яким працює інтерфейс або логіка програми.

- Facade Pattern:

Приклад: **authService.js, fileService.js, testService.js та інші Service.js** – приховують складну логіку за простим інтерфейсом — дозволяє клієнтському коду працювати з сервісами, не знаючи про їхню внутрішню реалізацію. Приховують логіку запитів, обробки помилок, адаптації та авторизації, надаючи **прості функції.**

- Composite Pattern

Приклад: Layout.jsx - Створення інтерфейсу користувача з компонентів Header, Aside та Footer.

Структура: Layout (Composite)

```
├── Header (Leaf)
├── Aside (Leaf)
├── Main (Composite)
│   └── children (можуть бути Composite або Leaf)
└── Footer (Leaf)
```

Патерни поведінки:

- Observer Pattern

Приклад: SocketContext.js - Обробка подій Socket.io

```
const newSocket = io("http://localhost:5000", {
  query: { token: authToken },
  transports: ["websocket", "polling"],
  reconnection: true,
  reconnectionAttempts: 2,
  reconnectionDelay: 2000
});

newSocket.on("connect", () => makeToast("success", "Socket Connected"));
newSocket.on("disconnect", (reason) => {
  makeToast("error", `Socket Disconnected: ${reason}`);
  console.warn("🔴 Disconnected from server:", reason);
});
newSocket.on("connect_error", (err) => {
  console.error("⚠️ WebSocket connection error:", err);
});
```

- Strategy Pattern

Приклад: TestController.js - Різні стратегії оцінювання для різних типів питань

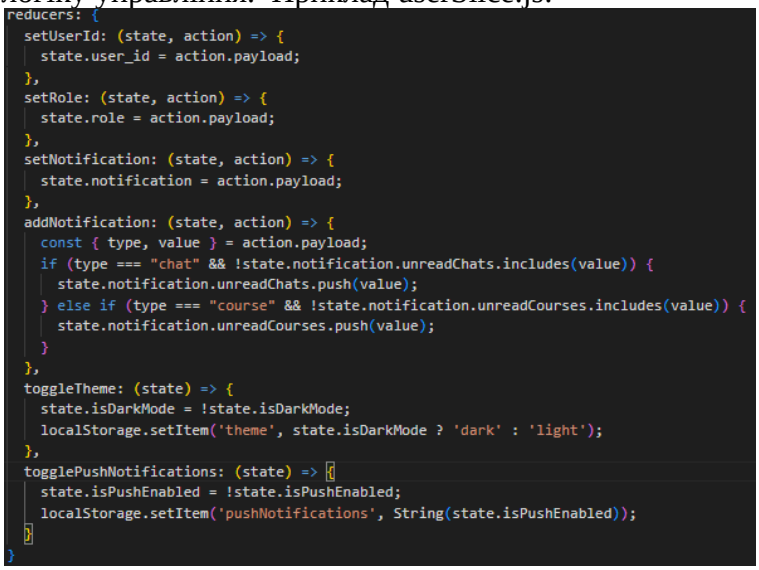
```
switch (original.type) {
  case 'single': ···

  case 'multiple': ···

  case 'short':
  case 'long': ···
}
```

- Command Pattern

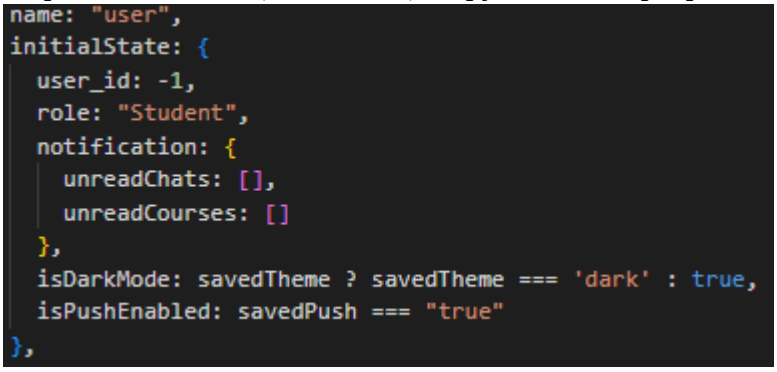
Приклад: опис дії у вигляді об’єкта (action), який відправляється через dispatch для зміни стану. Reducer приймає цю команду і оновлює стан, забезпечуючи чітку та передбачувану логіку управління. Приклад userSlice.js:



```
reducers: {
  setUserId: (state, action) => {
    state.user_id = action.payload;
  },
  setRole: (state, action) => {
    state.role = action.payload;
  },
  setNotification: (state, action) => {
    state.notification = action.payload;
  },
  addNotification: (state, action) => {
    const { type, value } = action.payload;
    if (type === "chat" && !state.notification.unreadChats.includes(value)) {
      state.notification.unreadChats.push(value);
    } else if (type === "course" && !state.notification.unreadCourses.includes(value)) {
      state.notification.unreadCourses.push(value);
    }
  },
  toggleTheme: (state) => {
    state.isDarkMode = !state.isDarkMode;
    localStorage.setItem('theme', state.isDarkMode ? 'dark' : 'light');
  },
  togglePushNotifications: (state) => {
    state.isPushEnabled = !state.isPushEnabled;
    localStorage.setItem('pushNotifications', String(state.isPushEnabled));
  }
}
```

- State Pattern

Приклад: Сховище Redux, що керує станом програми. Наприклад userSlice.js:



```
name: "user",
initialState: {
  user_id: -1,
  role: "Student",
  notification: {
    unreadChats: [],
    unreadCourses: []
  },
  isDarkMode: savedTheme ? savedTheme === 'dark' : true,
  isPushEnabled: savedPush === "true"
},
```

- Middleware Pattern

Приклад: authMiddleware.js - Проміжне програмне забезпечення автентифікації та roleMiddleware.js – проміжна перевірка ролі користувача.