Khmelnytskyi National University
Department of Computer Engineering and Information Systems

Report

Laboratory work No 4

Discipline: “Object Oriented Programming”

Completed: 2st year student, group SE-24-1 **Mamchur Danylo**

Checked: **Boiko Viacheslav**

Khmelnytskyi, 2025

Laboratory work № 4 "Classes Implementation"

Mamchur Danylo SE-24-1

Purpose: provide students with practical experience in understanding and implementing object-oriented programming concepts, specifically focusing on classes and interfaces; gain familiarity with defining and implementing classes and interfaces, as well as exploring relationships such as inheritance, composition, etc. between them; apply principles of OOP, including encapsulation, abstraction, inheritance, and polymorphism, to build modular and scalable code.

Task execution

1. I started developing all the necessary components for the system. TypeScript with the React framework was chosen as the main language. Currently, I have limited myself to the Landing Page and Auth forms. Since the entire system is too large, I have not yet implemented all of its parts, focusing on high-quality development.

2. Describe the use of the four basic principles of OOP with code snippets.

2.1 **Polymorphism:** in my React project, I have components that can **render differently** according to the specified properties:

```
const Input: React.FC<InputProps> = ({ type, placeholder, label, className,
...rest }) => {
  if (type === "checkbox") {
    return (
      <label className={`${styles.checkbox}${className ? ` ${className}` :
""}`}>
        <input type="checkbox" {...rest} />
        {label}
      </label>
    );
  }

  return (
    <input
      type={type}
      placeholder={placeholder}
      className={`${styles.input}${className ? ` ${className}` : ""}`}
      {...rest}
    />
  );
};
```

2.2 **Inheritance:** I have a **composition of components** that allows to extend the behavior of my components:

```
const HomePage: React.FC = () => {
  return (
    <>
      <Header />
      <section className={styles.main}>
        <h1 className={styles.title}>CapyCode</h1>
        <p className={styles.subtitle}>
          CapyCode is the best platform to help you enhance your skills,
expand your knowledge and
          prepare for technical interviews.
        </p>
        <Button text="Code Now!" className={styles.button} />
      </section>
    </>
  );
};
```

2.3 **Abstraction:** some of the custom and library APIs can **hide complexity** to **increase code readability**. We just use appropriate methods, w/o diving into the details:

```
import { Route, Routes } from "react-router-dom";
import HomePage from "./components/HomePage";
import Login from "./components/Login";

const App = () => {
  return (
    <Routes>
      <Route path="/" element={<HomePage />} />
      <Route path="/login" element={<Login />} />
    </Routes>
  );
};
```

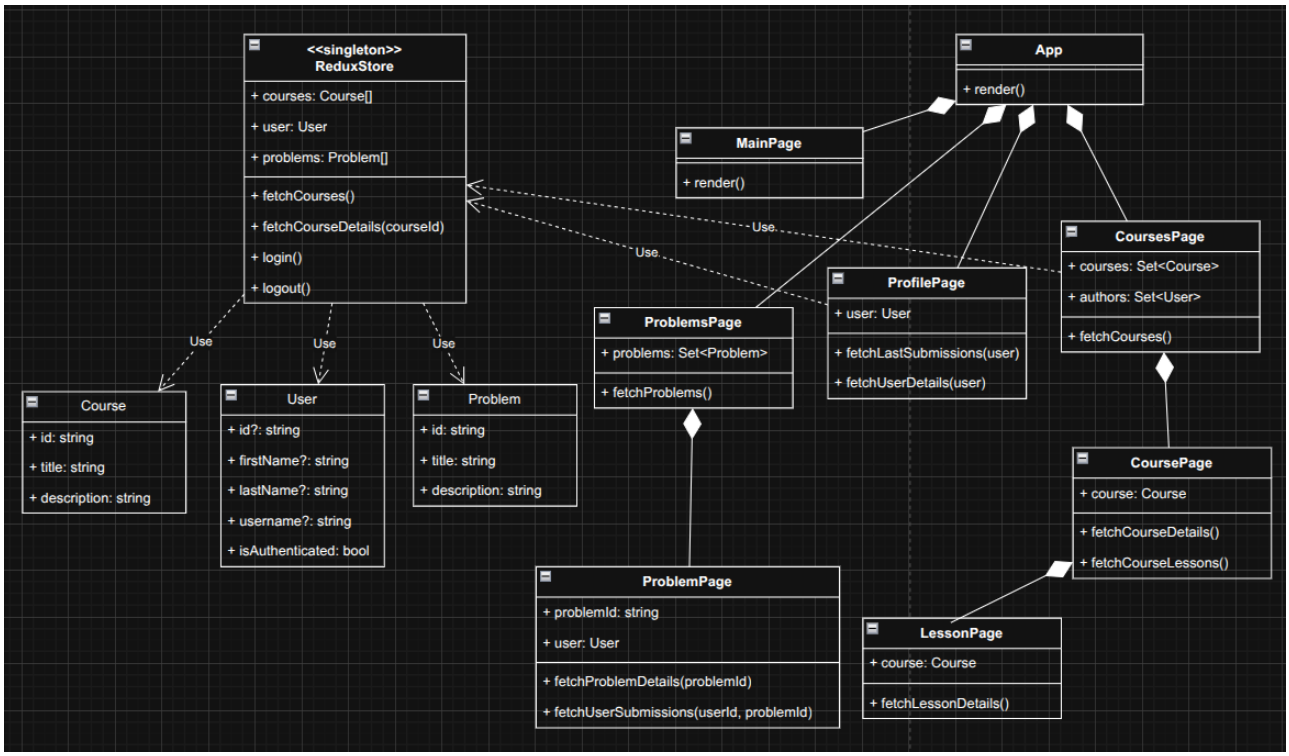
```
const {
  register,
  handleSubmit,
  watch,
  formState: { errors },
} = useForm<LoginInputs>();
const onSubmit: SubmitHandler<LoginInputs> = (data) => console.log(data);
```

2.4 **Encapsulation:** almost each component has its own state (useState, useReducer) and logic (useEffect, handlers) - they are not dirrectly accessable from the outside:

```
const [isLoggedIn, setIsLoggedIn] = useState(false);
const [username, setUsername] = useState("Login");
```

```
useEffect(() => {
  const storedUsername = localStorage.getItem("username");
  if (storedUsername) {
    setIsLoggedIn(true);
    setUsername(storedUsername);
  }
}, []);
```

3. The relationships between the objects:



Questions

1. How OOP principles are implemented:

- **Encapsulation:** Components in React hide their state (useState, useReducer) and expose data via props.
- **Abstraction:** Custom hooks (like useForm) hide complex logic behind a simple API.
- **Inheritance:** Achieved through composition - one component can wrap or reuse another.
- **Polymorphism:** Components or functions behave differently based on props or generic types.

2. Abstract classes and interfaces: **Abstract classes** define a common base with some unimplemented methods - they can't be instantiated directly. **Interfaces** describe the structure (types of properties and methods) that objects or classes must follow. In TypeScript, we mostly use interfaces to define props, data models, or API types.

3. Types of polymorphism in OOP: **Compile-time (static):** Method/function overloading - same name, different parameters. **Run-time (dynamic):** Method overriding - a subclass changes a parent method's behavior.

4. Relationships between objects in code: **Association:** One object uses another (e.g., a parent component renders a child). **Aggregation:** One object contains others but they can exist independently (e.g., list of items). **Composition:** One object owns and manages another's lifecycle (e.g., component owns its state or child components).

```
function Profile({ user }: { user: User }) { // Association
  return <Avatar src={user.photo} />;        // Composition
}
```