МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Программное обеспечение средств вычислительной техники и автоматизированных систем»

**Отчёт по практической работе №6**

**по курсу «Технологии разработки программного обеспечения»**

**по теме: «Архитектуры и паттерны.»**

**Вариант 1**

Выполнил:

Благушин Н.А.

Группа: М3О-107СВ-24

Приняли:

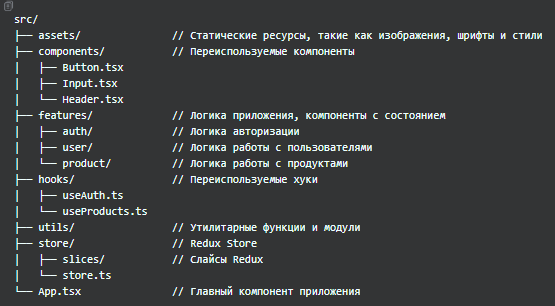
Титов Юрий Павлович

Москва 2024 г.

**Задание**

Практическая работа нацелена на изучения основ проектирования архитектуры, изменения, модернизации кода для удобства модификации и понимания кода программного обеспечения. В рамках практической работы необходимо составить архитектурную структуру разрабатываемого в творческой работе программного обеспечения, выделить отдельные компоненты, блоки, указать паттерны проектирования. В отчете по практической работе рассмотреть архитектурные решения с точки зрения принципов SOLID. Кроме составленной схемы, которая может быть составлена коллективом разработчиков или выполняться в рамках творческого задания (отдельных task в системе управления проектами) студентам необходимо предложить модернизацию программного кода путем добавления необходимых паттернов по вариантам: (1 - Добавление паттерна фабрика)

1. Примерная структура кода проекта:



1. Принципы SOLID в архитектуре:

S — Single Responsibility Principle (Принцип единственной ответственности): Каждый компонент и модуль отвечает за свою отдельную задачу. Например, компонент Button не должен заниматься логикой обработки форм, это делает компонент Form.

O — Open/Closed Principle (Принцип открытости/закрытости): Компоненты и модули должны быть открыты для расширений, но закрыты для изменений. Пример: для добавления новых типов кнопок можно добавить новый компонент, не изменяя существующий Button.

L — Liskov Substitution Principle (Принцип подстановки Лисков): Если компонент расширяет другой компонент, его можно безопасно использовать вместо родительского компонента без изменений в логике.

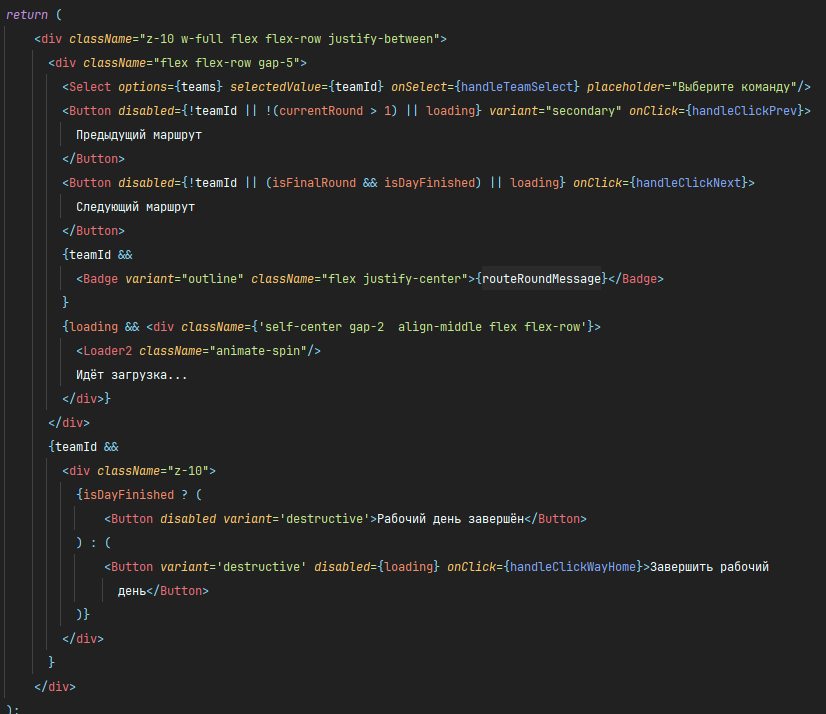
I — Interface Segregation Principle (Принцип разделения интерфейсов): Не заставляйте классы реализовывать интерфейсы, которые они не используют. Например, компонент должен содержать только те методы и свойства, которые ему необходимы.

D — Dependency Inversion Principle (Принцип инверсии зависимостей): Высокоуровневые модули не должны зависеть от низкоуровневых, и те, и другие должны зависеть от абстракций. Например, использование контекста или хуков для управления состоянием вместо прямого использования Redux внутри компонентов. Интерфейс для репозитория, чтобы инвертировать зависимость сервиса от репозитория на интерфейс. Сервис зависит от интерфейса, а не от конкретной реализации репозитория.

1. Добавление паттерна "Фабрика":

Паттерн "Фабрика" используется для создания объектов без явного указания класса создаваемого объекта. В контексте вашего проекта можно применить его для создания различных типов компонентов или сервисов.

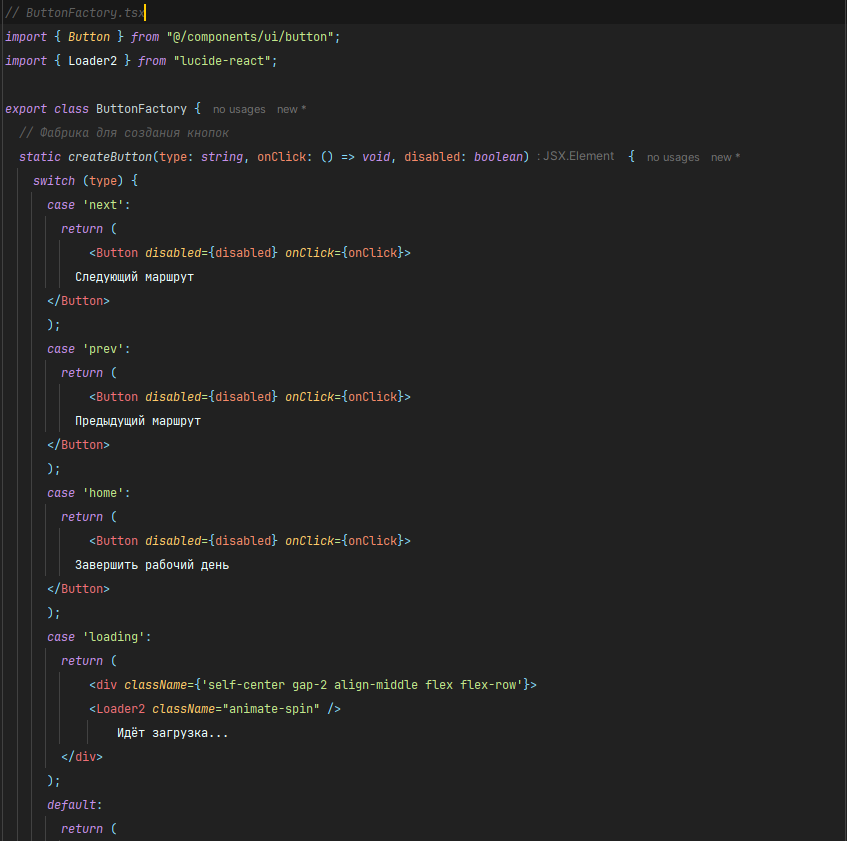
Было:

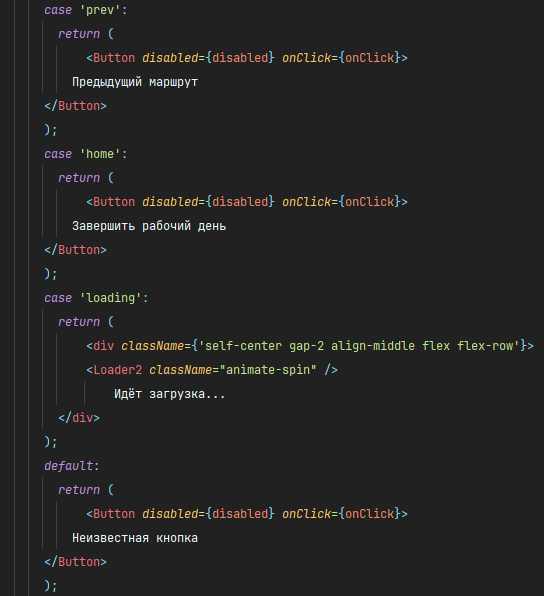


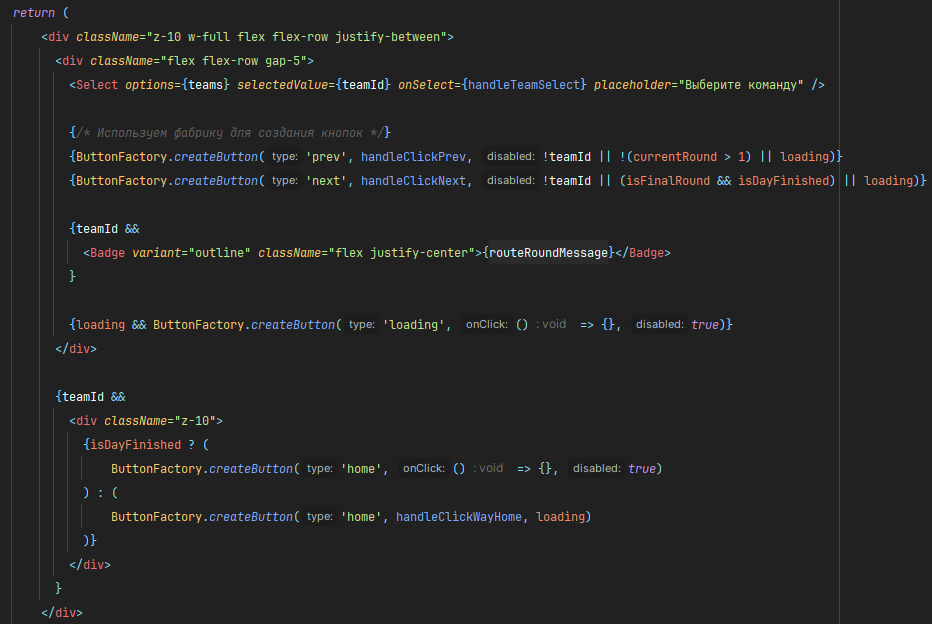
Пояснения:

В компоненте Form.tsx постоянно описываются разные виды и состояния кнопок в теле самого компонента, однако было бы лучше, согласно паттерну “Фабрика” создать единый компонент, который будет решать, какую именно кнопку отрисовать в Form.tsx и всех последующих компонентах проекта, которым необходима кнопка

Стало:







Объяснение изменений:

Мы создали класс ButtonFactory, который отвечает за создание различных кнопок в зависимости от типа и состояния.

В компоненте Form заменили явные проверки на вызов фабрики для кнопок. Это позволяет централизовать логику создания кнопок и упростить код компонента.

Фабрика возвращает кнопку с нужными свойствами в зависимости от переданных параметров (например, для кнопки "Следующий маршрут" или "Завершить рабочий день").

Этот подход упрощает добавление новых типов кнопок в будущем и делает код более расширяемым и модульным.