**Отчёт по лабораторной работе №4: Паттерн "Фиктивная служба"**

**Описание проблемы предметной области:**

В повседневной практике многие компании, предоставляющие услуги доставки, нуждаются в программных решениях, позволяющих рассчитывать стоимость доставки товаров в зависимости от их характеристик и расстояния. Однако при разработке и тестировании подобных систем возникает проблема: невозможность протестировать логику интерфейса без наличия доступа к реальному серверу или API, или же ответ с сервера занимает слишком много времени. Это замедляет процесс разработки, поскольку усложняет тестирование и увеличивает сложность отладки.

**Решение:**

Для решения этой проблемы в проекте применён архитектурный паттерн 'Фиктивная служба' (Mock Object). Суть паттерна заключается в создании поддельной реализации интерфейса сервиса доставки, которая имитирует реальное поведение. Это позволяет производить тестирование пользовательского интерфейса без подключения к серверу, а также ускоряет разработку за счёт быстрого получения предсказуемого результата.

Код с паттерном:

*DeliveryForm.tsx:*  
*// src/components/DeliveryForm.tsx*

import React, { useState } from 'react';

import { DeliveryParams, DeliveryResult, DeliveryService } from '../services/DeliveryService';

import './DeliveryForm.css';

interface Props {

  service: DeliveryService;

}

const DeliveryForm: React.FC<Props> = ({ *service* }) => {

  const [form, setForm] = useState<DeliveryParams>({

    weight: 0,

    length: 0,

    width: 0,

    height: 0,

    distance: 0,

  });

  const [result, setResult] = useState<DeliveryResult | null>(null);

  const [error, setError] = useState<string | null>(null);

  const [loading, setLoading] = useState<boolean>(false);

  const handleChange = (*e*: React.ChangeEvent<HTMLInputElement>) => {

    const { name, value } = *e*.target;

    const numValue = parseFloat(value) || 0;

    setForm((*prevForm*) => ({

      ...*prevForm*,

      [name]: numValue,

    }));

  };

  const handleSubmit = async (*e*: React.FormEvent) => {

*e*.preventDefault();

    setResult(null);

    setError(null);

    setLoading(true);

    try {

      const res = await *service*.calculateDelivery(form);

      setResult(res);

    } catch (*err*: any) {

      setError(err.message || 'Произошла ошибка');

    } finally {

      setLoading(false);

    }

  };

  return (

    <div *className*="delivery-form-container">

      <h2>Калькулятор стоимости доставки</h2>

      <form *className*="delivery-form" *onSubmit*={handleSubmit}>

        <div *className*="input-group">

          <label *htmlFor*="weight">Вес, кг:</label>

          <input

*type*="number"

*id*="weight"

*name*="weight"

*value*={form.weight}

*onChange*={handleChange}

*required*

*step*="any"

          />

        </div>

        <div *className*="input-group">

          <label *htmlFor*="length">Длина, м:</label>

          <input

*type*="number"

*id*="length"

*name*="length"

*value*={form.length}

*onChange*={handleChange}

*required*

*step*="any"

          />

        </div>

        <div *className*="input-group">

          <label *htmlFor*="width">Ширина, м:</label>

          <input

*type*="number"

*id*="width"

*name*="width"

*value*={form.width}

*onChange*={handleChange}

*required*

*step*="any"

          />

        </div>

        <div *className*="input-group">

          <label *htmlFor*="height">Высота, м:</label>

          <input

*type*="number"

*id*="height"

*name*="height"

*value*={form.height}

*onChange*={handleChange}

*required*

*step*="any"

          />

        </div>

        <div *className*="input-group">

          <label *htmlFor*="distance">Расстояние, км:</label>

          <input

*type*="number"

*id*="distance"

*name*="distance"

*value*={form.distance}

*onChange*={handleChange}

*required*

*step*="any"

          />

        </div>

        <button *type*="submit" *className*="calc-button">

          Рассчитать

        </button>

      </form>

      {loading && <p>Загрузка...</p>}

      {error && <p *className*="error-msg">Ошибка: {error}</p>}

      {result && (

        <div *className*="result">

          <h3>Результат:</h3>

          <p>Стоимость: {result.price} руб.</p>

          <p>Срок: {result.estimatedDays} дней</p>

        </div>

      )}

    </div>

  );

};

export default DeliveryForm;

*MockDeliveryService.tsx:*  
*//Фиктивная служба*

import { DeliveryParams, DeliveryResult, DeliveryService } from './DeliveryService';

export class MockDeliveryService implements DeliveryService {

  async calculateDelivery(*params*: DeliveryParams): Promise<DeliveryResult> {

    console.log('Mock вызван с:', *params*);

    return new Promise((*resolve*, *reject*) => {

      setTimeout(() => {

        const fail = Math.random() < 0.05; *// 5% ошибок*

        if (fail) {

          reject(new Error('Сбой mock-сервиса'));

        } else {

          resolve({ price: 999, estimatedDays: 3 });

        }

      }, 0);

    });

  }

}

RealDeliveryService.ts:  
*//Реальный сервис*

import { DeliveryParams, DeliveryResult, DeliveryService } from './DeliveryService';

import { calculateDeliveryFromApi } from '../api';

export class RealDeliveryService implements DeliveryService {

  async calculateDelivery(*params*: DeliveryParams): Promise<DeliveryResult> {

    return await calculateDeliveryFromApi(*params*);

  }

}

Api.ts:

import { DeliveryParams, DeliveryResult } from './services/DeliveryService';

*/\**

*\* Симуляция API-запроса на сервер для расчёта доставки.*

*\* Возвращает Promise с задержкой, как будто идёт сетевой вызов.*

*\*/*

export async function calculateDeliveryFromApi(*params*: DeliveryParams): Promise<DeliveryResult> {

  return new Promise((*resolve*) => {

    setTimeout(() => {

      const { weight, length, width, height, distance } = *params*;

      const volume = length \* width \* height;

      const price = weight \* 10 + volume \* 0.05 + distance \* 0.1;

      const estimatedDays = Math.ceil(distance / 500);

      resolve({

        price: Math.round(price),

        estimatedDays,

      });

    }, 800); *// Задержка 800 мс — как будто API отвечает*

  });

}

*app.tsx:*  
…

      <main *className*="main-content">

        <DeliveryForm *service*={service} />

      </main>

…

Без паттерна приходилось бы каждый раз напрямую обращаться к серверу, что замедляло бы процесс разработки:

DeliveryForm.tsx:  
// src/components/DeliveryForm.tsx

import React, { useState } from 'react';

import { DeliveryParams, DeliveryResult } from '../services/DeliveryService';

import { calculateDeliveryFromApi } from '../api';

import './DeliveryForm.css';

const DeliveryForm: React.FC = () => {

const [form, setForm] = useState<DeliveryParams>({

weight: 0,

length: 0,

width: 0,

height: 0,

distance: 0,

});

const [result, setResult] = useState<DeliveryResult | null>(null);

const [error, setError] = useState<string | null>(null);

const [loading, setLoading] = useState<boolean>(false);

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

const numValue = parseFloat(value) || 0;

setForm((prevForm) => ({

...prevForm,

[name]: numValue,

}));

};

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

setResult(null);

setError(null);

setLoading(true);

try {

const res = await calculateDeliveryFromApi(form);

setResult(res);

} catch (err: any) {

setError(err.message || 'Произошла ошибка');

} finally {

setLoading(false);

}

};

return (

<div className="delivery-form-container">

<h2>Калькулятор стоимости доставки</h2>

<form className="delivery-form" onSubmit={handleSubmit}>

<div className="input-group">

<label htmlFor="weight">Вес, кг:</label>

<input

type="number"

id="weight"

name="weight"

value={form.weight}

onChange={handleChange}

required

step="any"

/>

</div>

<div className="input-group">

<label htmlFor="length">Длина, м:</label>

<input

type="number"

id="length"

name="length"

value={form.length}

onChange={handleChange}

required

step="any"

/>

</div>

<div className="input-group">

<label htmlFor="width">Ширина, м:</label>

<input

type="number"

id="width"

name="width"

value={form.width}

onChange={handleChange}

required

step="any"

/>

</div>

<div className="input-group">

<label htmlFor="height">Высота, м:</label>

<input

type="number"

id="height"

name="height"

value={form.height}

onChange={handleChange}

required

step="any"

/>

</div>

<div className="input-group">

<label htmlFor="distance">Расстояние, км:</label>

<input

type="number"

id="distance"

name="distance"

value={form.distance}

onChange={handleChange}

required

step="any"

/>

</div>

<button type="submit" className="calc-button">

Рассчитать

</button>

</form>

{loading && <p>Загрузка...</p>}

{error && <p className="error-msg">Ошибка: {error}</p>}

{result && (

<div className="result">

<h3>Результат:</h3>

<p>Стоимость: {result.price} руб.</p>

<p>Срок: {result.estimatedDays} дней</p>

</div>

)}

</div>

);

};

export default DeliveryForm;

api.ts:  
// src/api.ts

import { DeliveryParams, DeliveryResult } from './services/DeliveryService';

export async function calculateDeliveryFromApi(params: DeliveryParams): Promise<DeliveryResult> {

return new Promise((resolve) => {

setTimeout(() => {

const { weight, length, width, height, distance } = params;

const volume = length \* width \* height;

const price = weight \* 10 + volume \* 0.05 + distance \* 0.1;

const estimatedDays = Math.ceil(distance / 500);

resolve({

price: Math.round(price),

estimatedDays,

});

}, 800); // Задержка 800 мс — как будто API отвечает

});

}

App.tsx:

// src/App.tsx

import React from 'react';

import DeliveryForm from './components/DeliveryForm';

import './App.css';

function App() {

return (

<div className="App">

<header className="App-header">

<h1>Доставка грузов</h1>

</header>

<main className="main-content">

<DeliveryForm />

</main>

</div>

);

}

export default App;

DeliveryService.ts:

// src/services/DeliveryService.ts

export interface DeliveryParams {

weight: number;

length: number;

width: number;

height: number;

distance: number;

}

export interface DeliveryResult {

price: number;

estimatedDays: number;

}

Использование паттерна "Фиктивная служба" (Mock Object) позволяет отделить бизнес-логику от конкретных реализаций внешних сервисов, благодаря чему упрощается тестирование, улучшается модульность и обеспечивается гибкость архитектуры. Такой подход позволяет легко подменять реальные службы на фиктивные в тестовой среде, не влияя на остальной код, и тем самым повышает надёжность системы, ускоряет разработку.

**Диаграмма классов**

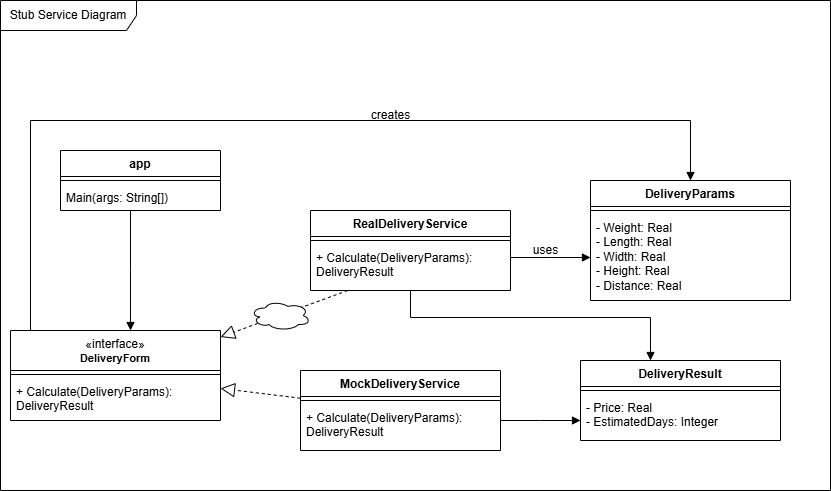


Рисунок 1 – Паттерн “Фиктивная служба” в архитектуре приложения.

**Вывод**

Применение паттерна 'Фиктивная служба' позволило разделить бизнес-логику и реализацию пользовательского интерфейса, а также обеспечило удобство тестирования без зависимости от сетевого окружения. Внедрение интерфейса и mock-класса значительно упростило отладку и сделало возможным быстрое переключение между реальным и тестовым режимом работы.