Методические рекомендации

Тема: 1. Теория: Проектирование и документация в разработке информационных систем

Ссылка на интерактивную методичку: https://olgakraven.github.io/GDRMI.PP.PM.03-method/

Навигация по разделам

1. Проектирование и документация в разработке информационных систем

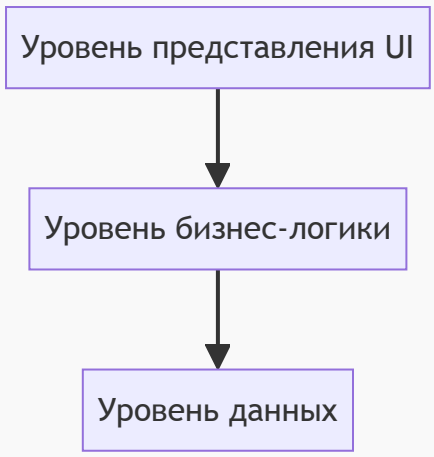
1.1. Анализ предметной области и формирование требований

1.1.2. Методы анализа

- Stakeholder Interview
- Observation
- Document Analysis
- As-Is Study
- BPMN, Use Case, Context Diagram

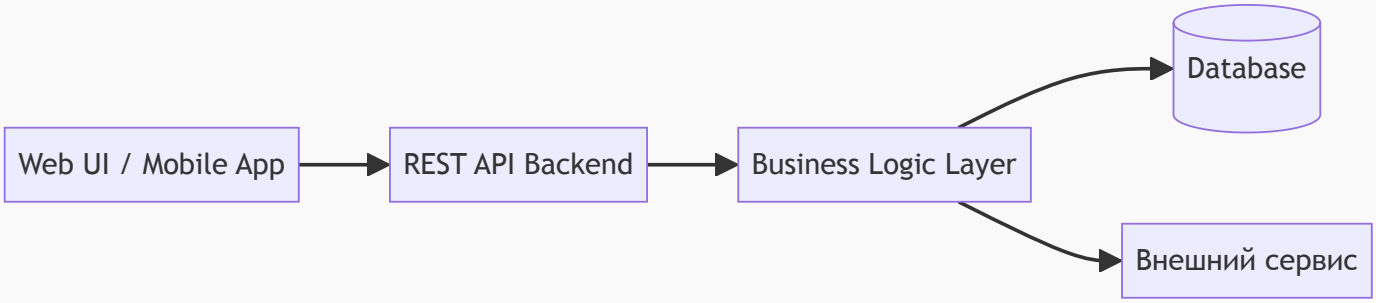
1.2. Основы архитектуры веб-приложений

1.2.2. Трёхуровневая архитектура

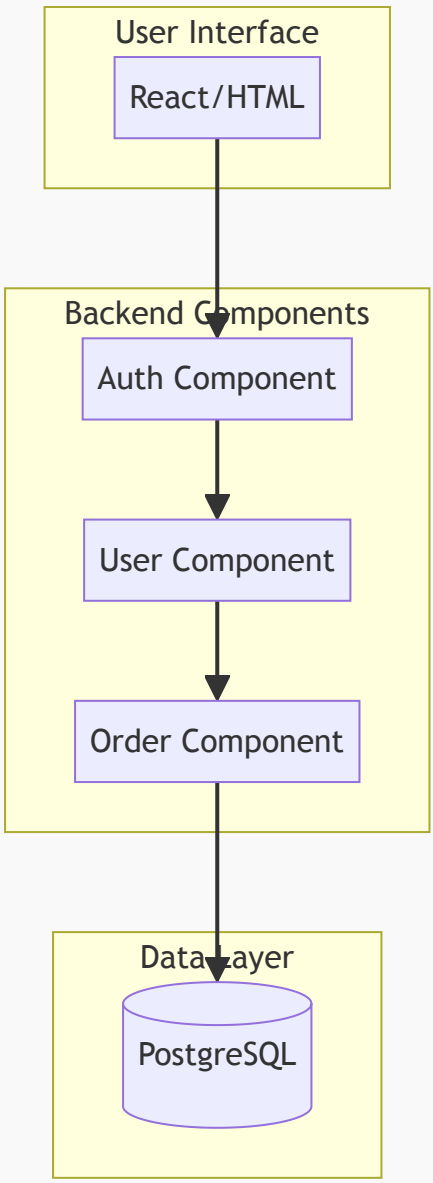


1.3. Архитектурные схемы

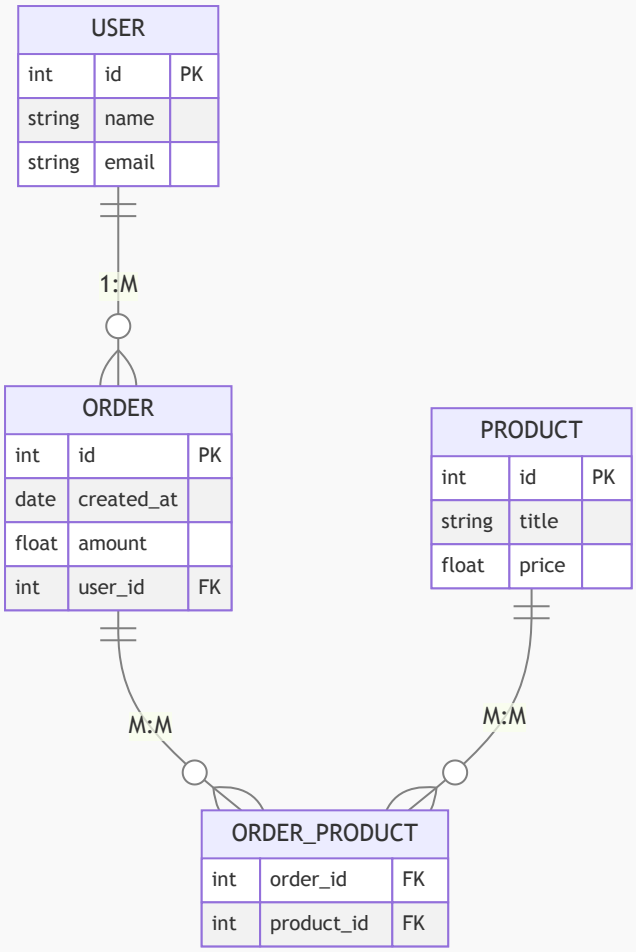
1.3.1. Block-diagram



1.3.2. Component Diagram



1.4. ER-диаграммы



1.5. Планирование разработки и диаграммы Ганта

Диаграмма Ганта — инструмент календарного планирования проекта, позволяющий наглядно отображать сроки, последовательность и зависимости задач в рамках разработки информационных систем.

1.5.1. Что показывает диаграмма Ганта

- перечень задач проекта;
- продолжительность каждой задачи;
- зависимости между задачами;
- загрузку исполнителей;
- критический путь проекта.

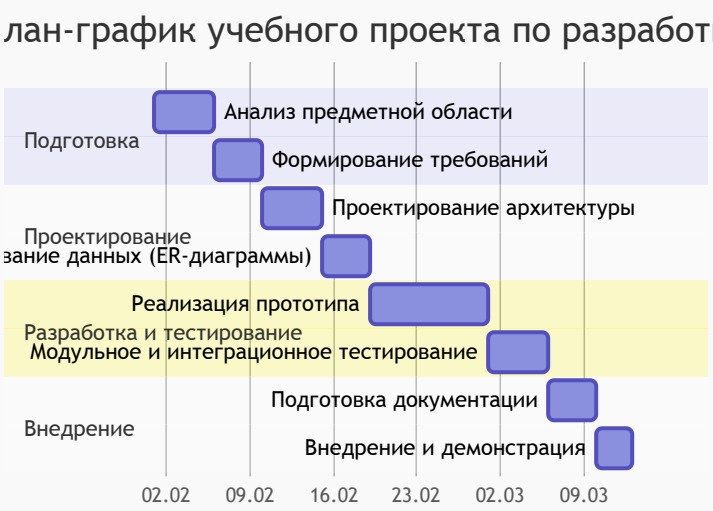
1.5.2. Применение в разработке ИС

Диаграмма Ганта используется для:

- планирования стадий анализа, проектирования, разработки, тестирования и внедрения;
- контроля сроков выполнения работ;
- оценки рисков и узких мест проекта;
- мониторинга прогресса команды на всех этапах.

Диаграммы Ганта подходят как для классического подхода проектного управления (Waterfall), так и для гибридных методологий. В Agile-подходах они применяются для учебных проектов, релизного планирования и визуального контроля сроков спринтов.

Пример диаграммы Ганта для учебного проекта (Mermaid):



Методические рекомендации

Тема: UI/UX: Принципы, инструменты и оптимизация интерфейсов

Навигация по разделам

2. UI/UX: Принципы, инструменты и оптимизация интерфейсов

2.1. Принципы пользовательского интерфейса: иерархия, типографика, визуальные паттерны

Основная задача UI-дизайна — обеспечить визуальную структуру, удобство восприятия и согласованность интерфейса.

2.1.1. Визуальная иерархия

Визуальная иерархия определяет, что пользователь видит первым и как распределяет внимание. Для управления иерархией используются:

- размер элементов;
- цвет и контраст;
- отступы и группировка;
- расположение на экране.

2.1.2. Типографика

Типографика определяет читаемость и удобство восприятия текста. Важно учитывать:

- выбор шрифтов;
- кегль (размер шрифта);
- межстрочные интервалы;

- начертание (обычный, полужирный, курсив);
- длину строки;
- выравнивание текста.

Грамотная типографика снижает когнитивную нагрузку и делает интерфейс профессиональным.

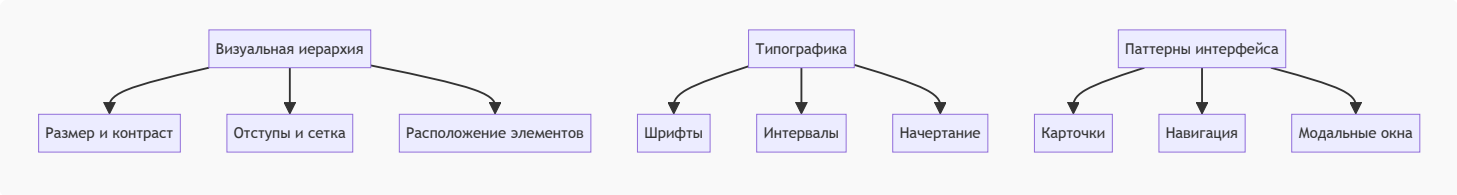
2.1.3. Визуальные паттерны

Визуальные паттерны — знакомые пользователю решения, которые повторяются в разных интерфейсах:

- карточки (cards);
- списки (lists);
- панели навигации (navbars);
- модальные окна;
- хлебные крошки (breadcrumbs);
- табы и аккордеоны.

Использование паттернов уменьшает время обучения работе с интерфейсом и повышает удобство использования.

Mermaid: визуальная структура UI



2.2. Основы UX-проектирования: юзабилити, пользовательские сценарии, адаптивный дизайн

UX-проектирование определяет, как пользователь взаимодействует с системой, насколько понятны шаги достижения цели и комфортен весь путь использования продукта.

2.2.1. Юзабилити

Хорошая система с точки зрения юзабилити:

- понятна и предсказуема;
- минимизирует количество ошибок пользователя;
- уменьшает когнитивную нагрузку;
- позволяет достигать цели без лишних действий;
- предоставляет понятную обратную связь.

2.2.2. Пользовательские сценарии (User Flows)

User Flow описывает путь пользователя к достижению цели (например, оформление заказа в интернет-магазине).

Типовые этапы User Flow:

1. Начальная точка (Home / Login);
2. Выбор действия (поиск товара, переход в каталог);
3. Прохождение шагов (детализация, корзина, ввод данных);
4. Завершение задачи (оплата, подтверждение, результат).

Mermaid: пример пользовательского сценария

2.2.3. Адаптивный дизайн

Адаптивный дизайн позволяет интерфейсу корректно отображаться на разных устройствах и экранах:

- мобильные телефоны;
- планшеты;
- ноутбуки;
- широкоформатные мониторы.

Основные техники адаптивности:

- гибкие сетки (flex, grid);
- относительные единицы измерения (%, rem, vw);
- медиа-запросы;
- responsive images (подбор изображений под размер экрана).

2.3. Работа с векторными редакторами (Figma, Draw.io и аналоги)

Векторные редакторы используются для:

- создания UI-макетов;
- проектирования дизайн-систем;
- разработки интерактивных прототипов;
- совместной работы над проектом;
- построения схем (диаграмм, flowcharts, wireframes);
- экспорта изображений и ресурсов.

Наиболее распространённые инструменты:

- Figma (стандарт индустрии);
- Sketch;
- Adobe XD;
- Draw.io (диаграммы и схемы);
- Lunacy (альтернатива Sketch).

Figma поддерживает:

- создание и использование компонентов;
- авто-layout;
- дизайн-библиотеки;
- прототипирование сценариев;
- дизайн-токены;
- коллаборацию в реальном времени.

2.4. Понимание DOM-структуры и её влияние на производительность

DOM (Document Object Model) — это дерево элементов, из которых состоит веб-страница. Чем глубже и сложнее дерево, тем больше нагрузка на:

- рендеринг;
- перерасчёт стилей;
- перерисовку элементов (repaint/reflow).

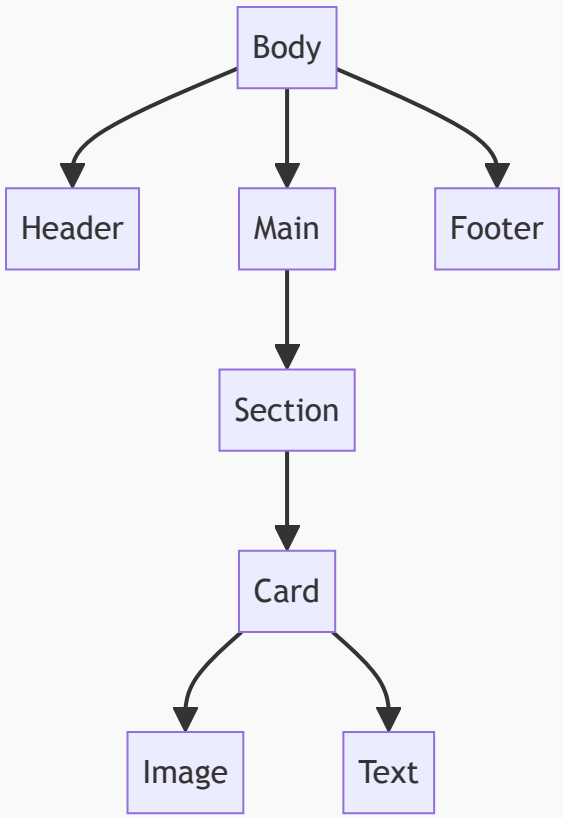
2.4.1. Главные проблемы перегруженного DOM

- чрезмерная вложенность контейнеров;
- большое количество лишних `<div>`;
- использование тяжёлых и сложных CSS-селекторов;
- частые изменения DOM (динамическая перерисовка больших фрагментов).

2.4.2. Рекомендации по оптимизации

- минимизировать вложенность элементов;
- использовать CSS Grid/Flex вместо множества обёрток;
- избегать сложных CSS-селекторов;
- уменьшать количество тяжёлых графических элементов;
- для анимаций использовать `transform` вместо `top/left`.

Mermaid: упрощённая структура DOM



2.5. Принципы микровзаимодействий: анимации, переходы, hover-эффекты

Микровзаимодействия — небольшие отклики интерфейса, которые дают пользователю обратную связь и помогают понять, что система «жива» и реагирует на действия.

2.5.1. Что включают микровзаимодействия

- hover-подсветка элементов;
- анимация при клике;
- плавные переходы состояний (открытие/закрытие блоков);
- индикаторы загрузки;
- подтверждение действий (toast-уведомления, небольшие анимации подтверждения).

2.5.2. Требования к микровзаимодействиям

- быстрые (примерно 150–250 мс);
- лёгкие (не вызывают подвисаний интерфейса);
- ненавязчивые;
- информативные и понятные пользователю.

Грамотно реализованные микровзаимодействия повышают комфорт работы и улучшают общее впечатление от UX.

2.6. Основы оптимизации графики: форматы, сжатие, responsive images

2.6.1. Рекомендуемые форматы

- **WebP** — один из лучших форматов для веба (баланс качества и размера);
- **SVG** — векторная графика (иконки, логотипы, простые иллюстрации);
- **AVIF** — современный формат с ещё более высокой степенью сжатия;
- **PNG** — когда требуется прозрачность и точность деталей;
- **JPEG** — фотографии при умеренном сжатии.

2.6.2. Техники оптимизации

- сжатие изображений (например, TinyPNG, Squoosh);
- выбор оптимального размера изображений под макет;
- ленивая загрузка (атрибут `loading="lazy"`);
- использование спрайтов для SVG-иконок;
- использование CDN для раздачи изображений.

2.6.3. Responsive Images

Использование атрибутов `srcset` и `sizes` позволяет загружать изображение оптимального размера для устройства пользователя.

Пример responsive images (HTML-код)

```
<img
  src="img-800.jpg"
  srcset="img-400.jpg 400w, img-800.jpg 800w, img-1200.jpg 1200w"
  sizes="(max-width: 600px) 400px, 800px"
  loading="lazy"
  alt="Пример изображения для responsive-разметки"
/>
```

Методические рекомендации

Тема: Frontend-разработка

Навигация по разделам

3. Frontend-разработка

3.1. HTML5: структура документа, семантика, оптимизация DOM

HTML5 определяет структуру страницы и влияет на восприятие содержимого браузером, поисковыми системами и пользователем. От качества вёрстки зависят доступность, SEO и производительность интерфейса.

3.1.1. Семантическая структура документа

Пример базового HTML5-шаблона:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Пример HTML5</title>
</head>
<body>
    <header>Заголовок сайта</header>
    <nav>Меню</nav>
    <main>
        <article>
            <h1>Статья</h1>
```

```
            <section>Раздел статьи</section>
        </article>
    </main>
    <footer>Подвал</footer>
</body>
</html>
```

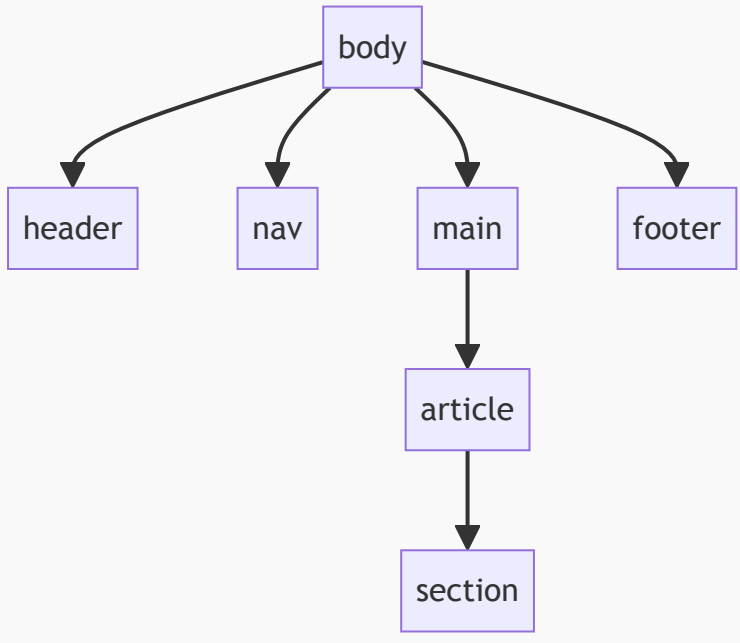
3.1.2. Примеры семантических тегов

- `<header>` — верхняя часть страницы или блока;
- `<nav>` — меню, навигация;
- `<main>` — основной контент страницы;
- `<article>` — самостоятельный фрагмент (статья, пост, новость);
- `<section>` — логический раздел документа;
- `<footer>` — нижний блок страницы или раздела.

3.1.3. Оптимизация DOM

- избегать лишних `<div>` и использовать семантические теги;
- минимизировать вложенность (желательно не более 4–5 уровней);
- объединять элементы, если это возможно;
- использовать списки для однотипных элементов (например, меню);
- избегать повторяющихся структур — применять компонентный подход.

Mermaid: оптимизированная DOM-структура



3.2. CSS3: каскадность, селекторы, оптимизация весов и модульность стилей

CSS управляет визуальным оформлением интерфейса. Правильная структура стилей снижает вес файлов, уменьшает количество конфликтов и упрощает поддержку проекта.

3.2.1. Пример различий специфичности

```
/* Специфичность: низкая */
button {
    color: black;
}

/* Средняя */
nav .menu button {
    color: blue;
}

/* Очень высокая */
#sidebar nav .menu button.active {
    color: red;
}
```

3.2.2. Рекомендации по оптимизации CSS

- использовать методологии (BEM, ITCSS и т. п.);
- избегать вложенности более чем на 3 уровня;
- минимизировать селекторы вида `div div div`;
- выносить повторяющиеся значения в CSS-переменные.

Пример использования CSS-переменной:

```
:root {
    --accent-color: #ff5722;
}

button {
    background: var(--accent-color);
}
```

3.2.3. Пример BEM

```
<div class="card card--wide">
    <h2 class="card__title">Заголовок</h2>
```

```
    <p class="card__text">Описание</p>
</div>
```

3.3. Flexbox и CSS Grid для адаптивной вёрстки

3.3.1. Пример Flexbox

```
.menu {
    display: flex;
    justify-content: space-between;
    align-items: center;
}
```

Flexbox удобно использовать для:

- горизонтальных меню;
- карточек;
- панелей и тулбаров;
- выравнивания элементов по оси.

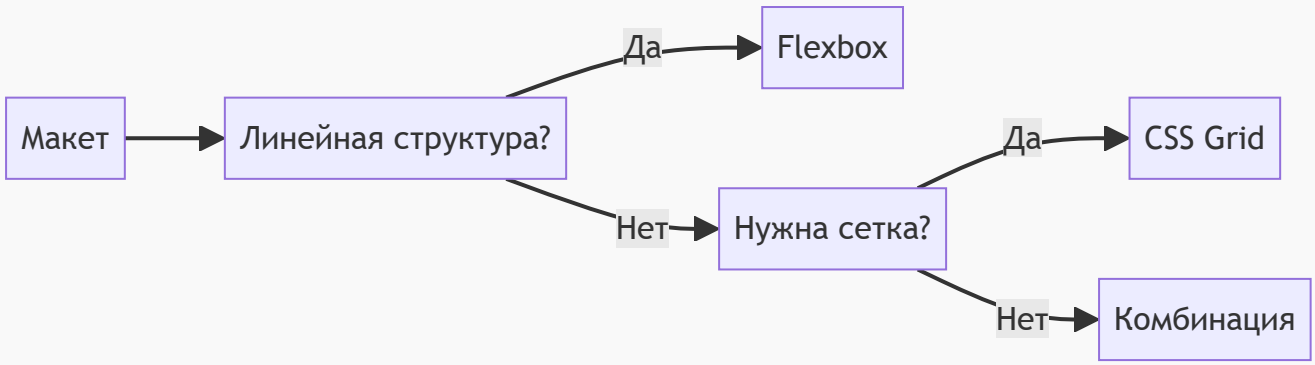
3.3.2. Пример CSS Grid

```
.grid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 20px;
}
```

CSS Grid подходит для:

- двухмерных макетов;
- панелей каталога;
- таблиц и списков товаров;
- сложных адаптивных сеток.

Mermaid: логика использования Flex vs Grid



3.4. JavaScript (ES6+): оптимизация кода, работа с DOM и событиями

JavaScript добавляет интерактивность, динамическую загрузку данных и реактивность интерфейса. Важно использовать современные возможности языка и оптимизировать работу с DOM.

3.4.1. Пример современных возможностей ES6+

```
const apiUrl = "/api/products";

async function loadProducts() {
    const response = await fetch(apiUrl);
    const data = await response.json();
    console.log(data);
}

loadProducts();
```

3.4.2. Работа с DOM

```
const button = document.querySelector(".btn");

button.addEventListener("click", () => {
    console.log("Кнопка нажата");
});
```

3.4.3. Оптимизация событий

```
function debounce(fn, delay) {
    let timeout;
    return (...args) => {
        clearTimeout(timeout);
```

```
        timeout = setTimeout(() => fn(...args), delay);
    };
}

window.addEventListener("resize", debounce(() => {
    console.log("Оптимизированное событие resize");
}, 200));
```

3.4.4. Снижение нагрузки

- минимизация количества обращений к DOM;
- кэширование результатов запросов и DOM-элементов;
- использование делегирования событий (event delegation);
- асинхронная загрузка тяжёлых ресурсов.

3.5. Lazy-loading и динамическая подгрузка ресурсов

3.5.1. Lazy-loading изображений

Пример ленивой загрузки изображения:

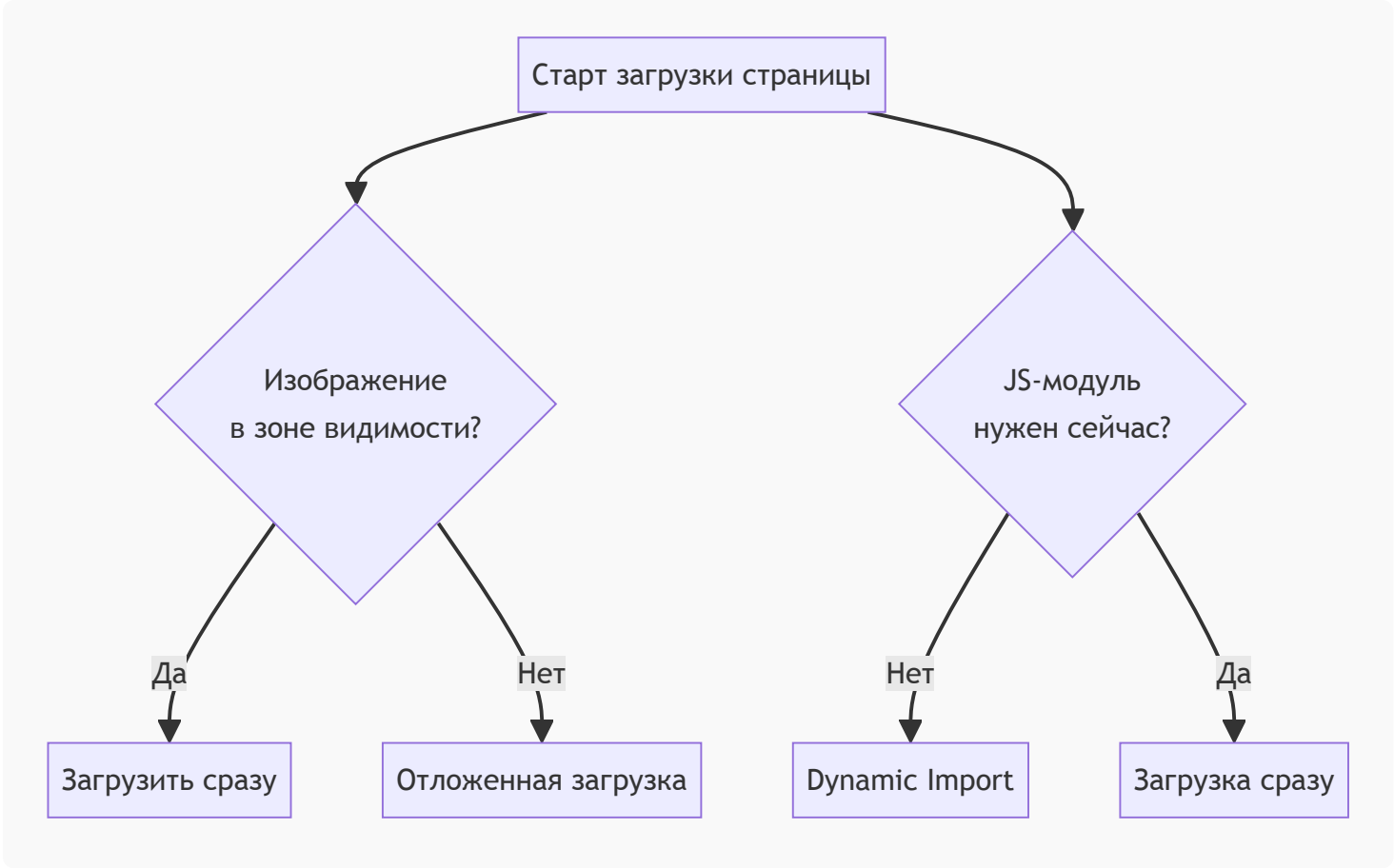
```
<img
  src="small.jpg"
  data-src="large.jpg"
  loading="lazy"
  alt="Пример изображения"
/>
```

3.5.2. Динамический импорт JS-модулей

```
document.querySelector("#open").addEventListener("click", async () => {
    const module = await import("./modal.js");
    module.openModal();
});
```

Эти методы ускоряют загрузку сайта и уменьшают ключевые метрики рендеринга (FCP, LCP).

Mermaid: логика подгрузки ресурсов



3.6. Аудит производительности в Chrome DevTools (Network, Performance, Lighthouse)

Chrome DevTools — основной инструмент анализа качества фронтенда. Позволяет исследовать загрузку ресурсов, работу JavaScript и общую производительность приложения.

3.6.1. Вкладка Network

Network позволяет оценить:

- количество запросов;
- размер загружаемых ресурсов;
- время загрузки и порядок запросов;
- наличие блокирующих файлов (CSS, JS).

3.6.2. Вкладка Performance

Performance показывает:

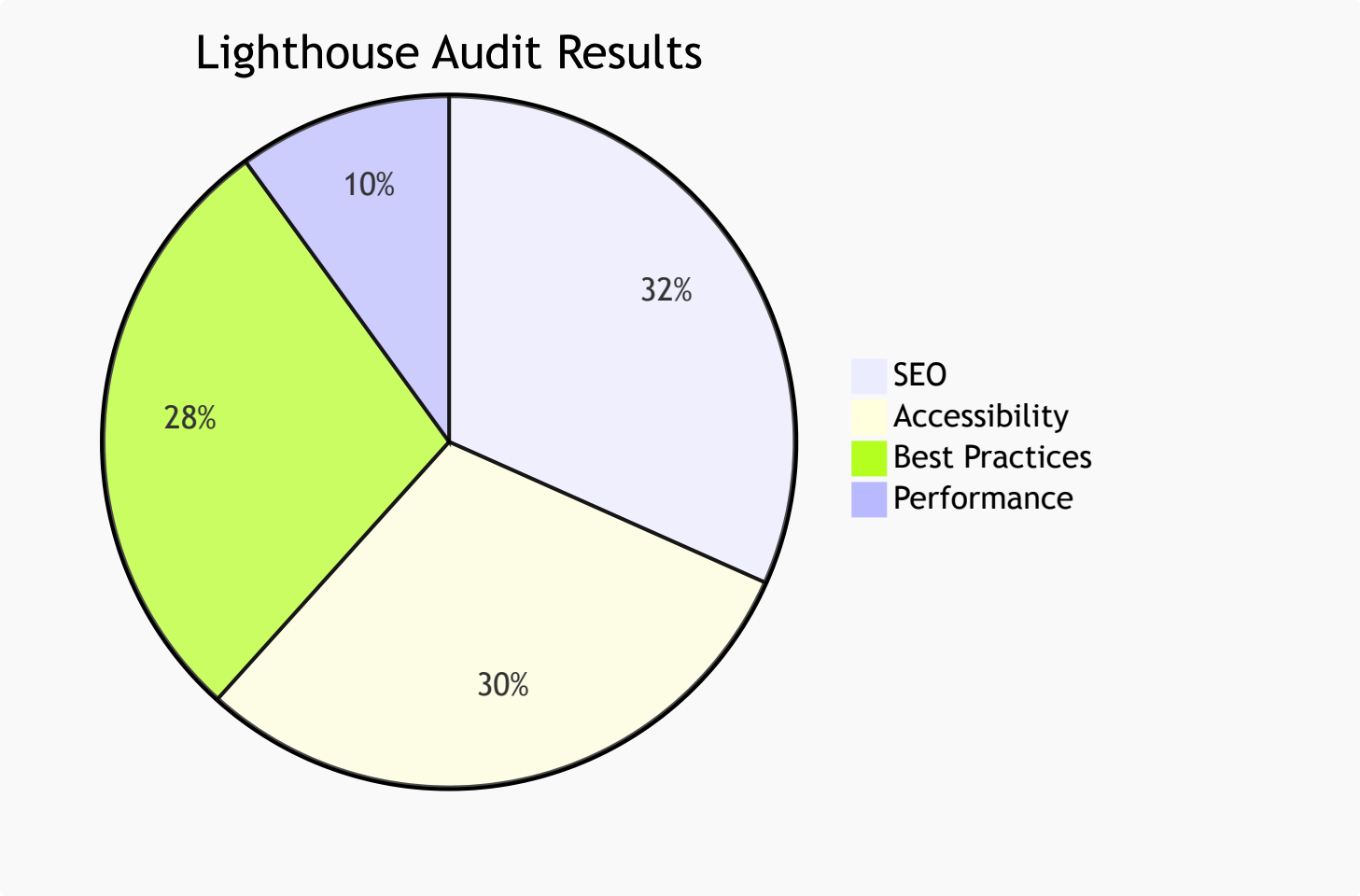
- работу основного JS-потока;
- время рендеринга;
- выполнение событий и обработчиков;
- перерисовки и layout-циклы.

3.6.3. Lighthouse

Lighthouse даёт автоматический аудит по направлениям:

- Performance;
- Accessibility;
- SEO;
- Best Practices;
- Progressive Web App.

Mermaid: пример отчёта Lighthouse (диаграмма)



Методические рекомендации

Тема: 5. Теория: Backend-разработка

Навигация

4.1. Принципы серверной архитектуры и обработка HTTP-запросов

Backend — серверная часть приложения, которая принимает HTTP-запросы, выполняет бизнес-логику, обращается к базе данных и формирует ответ.

4.1.1. Пример REST-маршрута (Node.js / Express)

```
const express = require("express");
const app = express();

app.use(express.json());

app.get("/api/users/:id", (req, res) => {
    const userId = req.params.id;
    res.json({ message: "Получен пользователь", id: userId });
});

app.post("/api/users", (req, res) => {
    const data = req.body;
    res.status(201).json({ created: true, data });
});
```

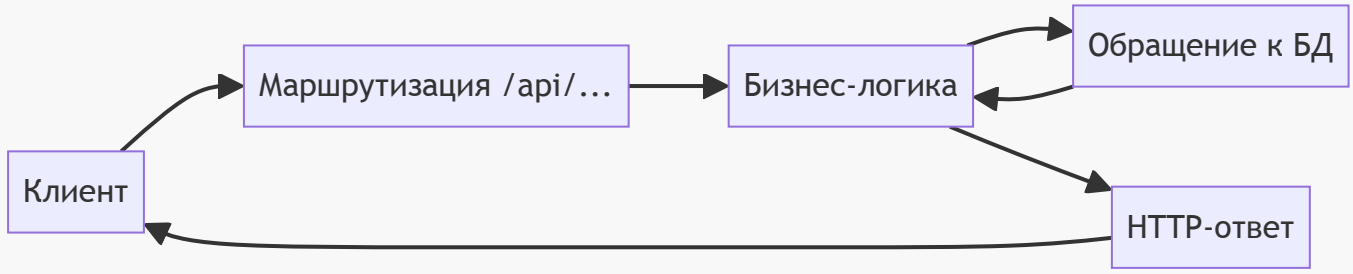
```
app.listen(3000, () => console.log("Server running on port 3000"));
```

4.1.2. Типы HTTP-методов

- **GET** — получение данных
- **POST** — создание ресурса
- **PUT** — полное обновление ресурса
- **PATCH** — частичное обновление
- **DELETE** — удаление

4.1.3. Пример HTTP-ответа

```
{
  "status": "ok",
  "data": { "id": 5, "name": "Alex" }
}
```



4.2. SQL: SELECT, JOIN, агрегирование, индексы

4.2.1. SELECT — выборка данных

```
SELECT id, name, email
FROM users
WHERE active = true;
```

4.2.2. JOIN — объединение таблиц

```
SELECT users.name, orders.total_price
FROM users
JOIN orders ON users.id = orders.user_id;
```

4.2.3. Агрегирующие функции

```
SELECT
    COUNT(*) AS total_orders,
    SUM(total_price) AS revenue,
    AVG(total_price) AS average_check
FROM orders;
```

4.2.4. Индексы

```
CREATE INDEX idx_orders_user_id
ON orders(user_id);
```

Индексы ускоряют выборку, соединения и сортировку, но замедляют INSERT/UPDATE.

4.3. Анализ SQL-запросов с помощью EXPLAIN

EXPLAIN показывает используемые индексы, количество строк, тип соединений и узкие места.

4.3.1. Пример EXPLAIN

```
EXPLAIN
SELECT users.name, orders.total_price
FROM users
JOIN orders ON users.id = orders.user_id
WHERE users.active = true;
```

Упрощённый результат:

```
Nested Loop
 ├─ Filter: (active = true)
 │    └─ Seq Scan on users
 └─ Index Scan using idx_orders_user_id on orders
```

Проблема: Seq Scan — полное сканирование таблицы.

4.3.2. Оптимизация

```
CREATE INDEX idx_users_active ON users(active);
```

4.4. Приёмы борьбы с проблемой N+1

N+1 — 1 запрос + N дополнительных запросов для связанных данных.

4.4.1. Пример N+1 в ORM

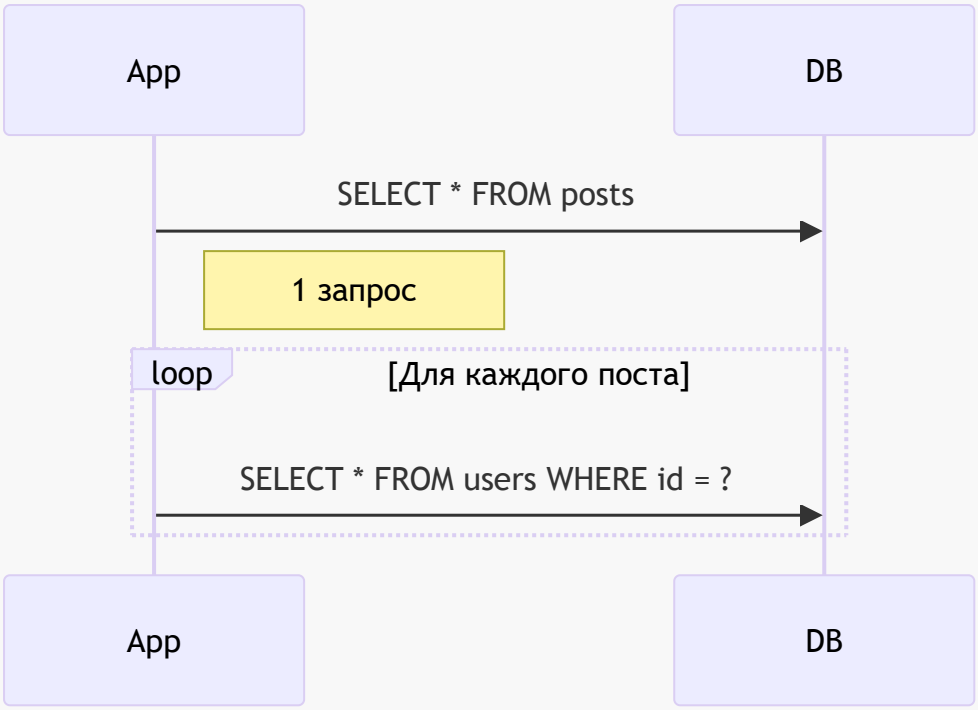
```
const posts = await Post.findAll();

for (const post of posts) {
    const author = await post.getAuthor(); // ещё один запрос
}
```

100 постов → 101 запрос.

4.4.2. Решение: eager loading

```
const posts = await Post.findAll({
    include: ["author"]
});
```

ORM выполнит один JOIN вместо 100 запросов.



4.5. Принципы кэширования

Кэширование ускоряет backend и снижает нагрузку на БД.

4.5.1. Кэширование в памяти (Redis)

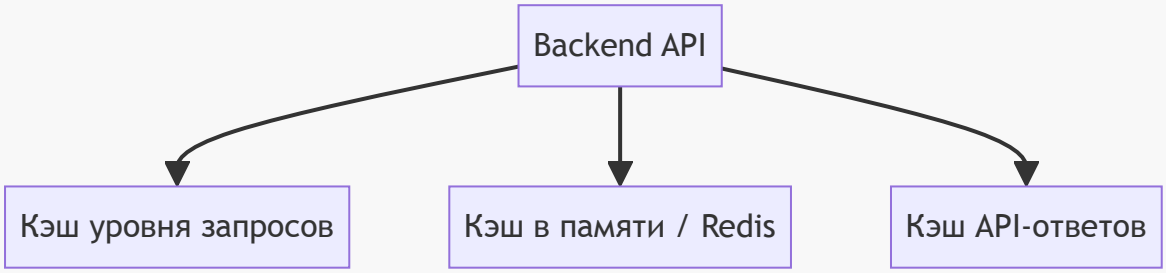
```
const redis = require("redis");
const client = redis.createClient();

app.get("/api/products", async (req, res) => {
    const cache = await client.get("products");

    if (cache) return res.json(JSON.parse(cache));

    const data = await db.getProducts();
    await client.set("products", JSON.stringify(data), { EX: 180 });

    res.json(data);
});
```

4.5.2. Кэширование SQL-запросов

```
await db.cacheQuery("SELECT * FROM categories");
```

4.5.3. Кэширование API-ответов

```
res.set("Cache-Control", "public, max-age=3600");
res.json(data);
```



4.6. Оптимизация структуры таблиц и конфигурации СУБД

4.6.1. Правильные типы данных

- INTEGER vs BIGINT
- VARCHAR(50) вместо TEXT
- BOOLEAN вместо INTEGER

4.6.2. Первичные и внешние ключи

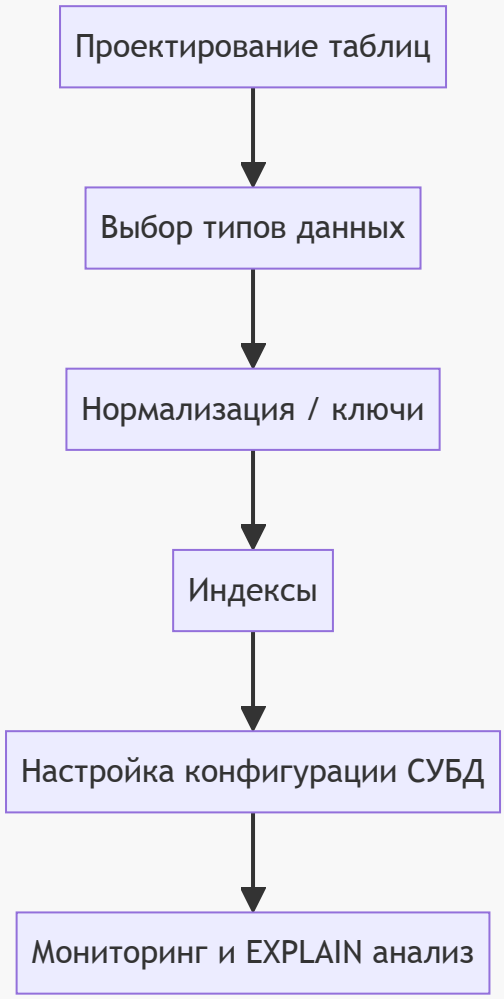
```
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    total_price NUMERIC(10, 2)
);
```

4.6.3. Частичные индексы

```
CREATE INDEX idx_users_active
ON users(active)
WHERE active = true;
```

4.6.4. Настройки СУБД

- pool size
- shared_buffers
- work_mem
- логирование медленных запросов
- autovacuum



Методические рекомендации

Тема: 5. Теория: Инструменты frontend/backend-разработчика

Навигация

5.1. Использование Chrome DevTools

Chrome DevTools — основной инструмент анализа фронтенд-приложений. Он показывает загрузку ресурсов, работу JavaScript, рендеринг и производительность.

5.1.1. Раздел Network — сетевые запросы

Network позволяет:

- видеть количество запросов;
- анализировать статус-коды;
- оценивать вес файлов;
- выявлять блокирующие ресурсы;
- отслеживать работу API.

Пример анализа:

```
GET /api/users      200 OK   45 ms   2.1 KB
GET /styles.css     200 OK   12 ms   14 KB
GET /image.png      200 OK   98 ms   620 KB   ⚠ слишком тяжёлое изображение
```

5.1.2. Раздел Performance — работа JS и рендеринг

Performance показывает:

- long tasks (долгие операции JS);
- Recalculate Style, Layout, Paint;
- обработку событий;
- FPS.

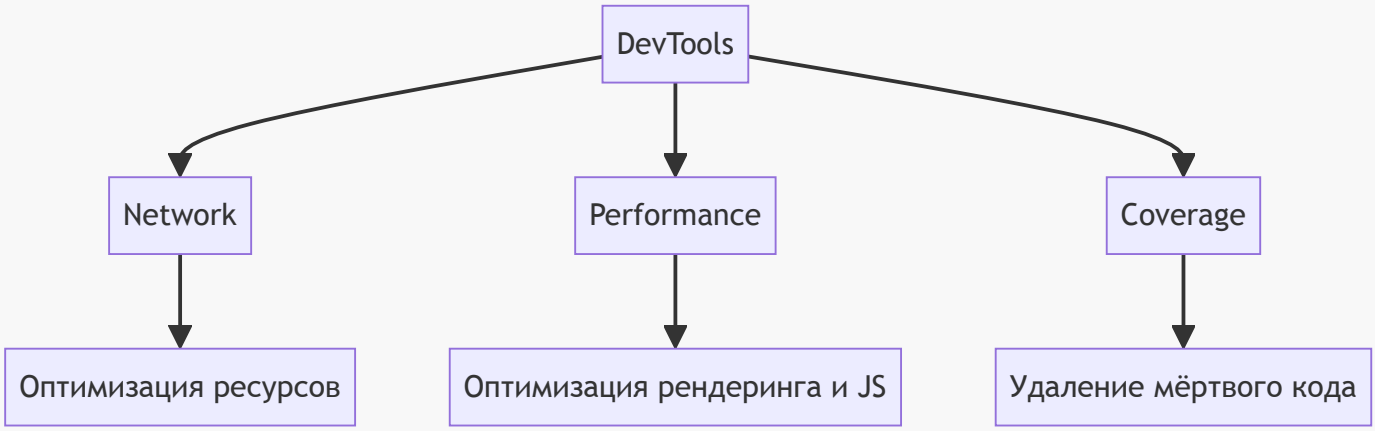
Примеры проблем:

```
"Recalculate Style: 87ms" — слишком сложные селекторы
"Layout: 120ms"           — тяжелая сетка Grid
"Long task: 300ms"        — неоптимальный цикл JS
```

5.1.3. Раздел Coverage — анализ «мёртвого кода»

Coverage показывает, что реально используется.

```
styles.css    10% used
main.js       32% used
framework.js  5% used  ⚠ подгружён весь React, используется только один хук
```



5.2. Использование Figma и Draw.io

Figma — для UI/UX; Draw.io — для диаграмм, архитектурных схем и документации.

5.2.1. Figma — дизайн и прототипирование

Figma используется для:

- создания UI-макетов;
- разработки дизайн-систем;
- прототипов;
- адаптивного дизайна (Auto Layout);
- совместной работы.

```
/UI Kit
    - Colors
    - Typography
    - Components

/Pages
    - Login
    - Dashboard
    - Profile

/Prototypes
```

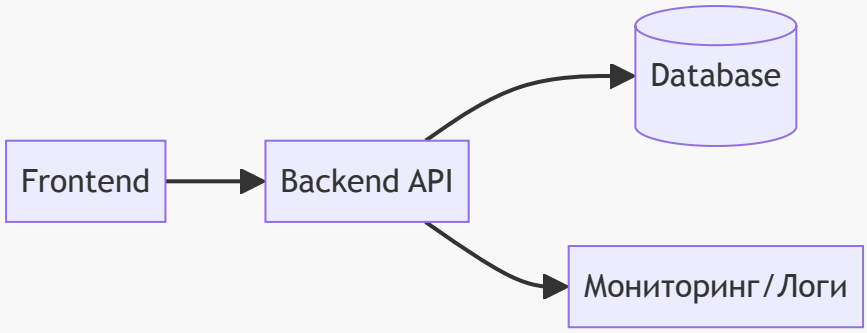
5.2.2. Draw.io — схемы и модели

Используется для:

- flowcharts;
- UML диаграмм;
- ER-моделей;
- архитектурных схем;
- технической документации.

Пример схемы:

```
Client → API Gateway → Auth Service
                     → User Service
                     → Order Service

Database → PostgreSQL
```



5.3. Навыки чтения логов и интерпретации метрик

5.3.1. Уровни логирования

DEBUG	технические детали
INFO	события приложения
WARN	возможные проблемы
ERROR	ошибки
FATAL	критические сбои

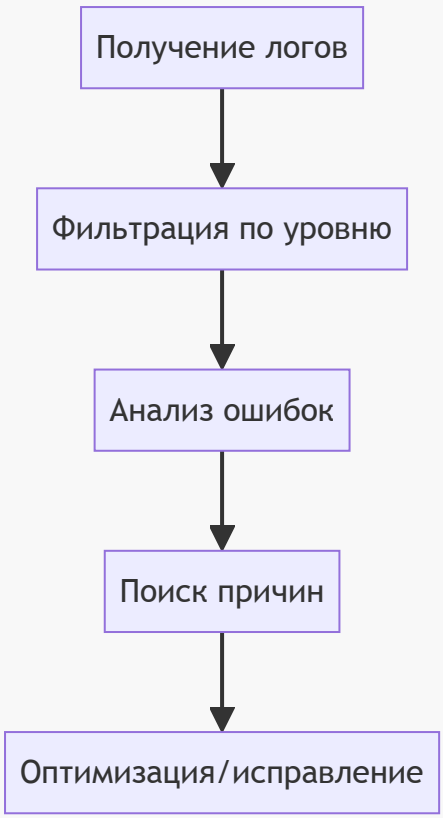
5.3.2. Пример логов (JSON)

```
{
  "level": "ERROR",
  "timestamp": "2025-02-01T12:41:22Z",
  "service": "OrderService",
  "message": "Database timeout",
  "details": {
    "query": "SELECT * FROM orders",
    "duration_ms": 1500
  }
}
```

5.3.3. Основные метрики

- **Response Time** — скорость ответа
- **Throughput (RPS)** — запросы/сек
- **CPU/Memory Usage** — нагрузка
- **Error Rate** — количество ошибок
- **DB Connections** — загрузка БД

- **Cache Hit Ratio** — эффективность кэша



5.4. Работа с Git/GitHub

5.4.1. Основные команды Git

```
git init
git clone <repo>
git checkout -b feature/login
git add .
git commit -m "Implement login"
git push origin feature/login
```

5.4.2. Pull Request — пример описания

```
# Feature: Добавление формы авторизации

## Что сделано
- Создан компонент LoginForm
- Добавлены валидации
- Настроена отправка данных на API
```

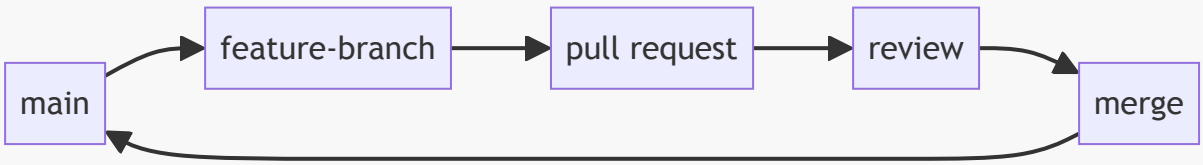
```
## Как тестировать
1. Открыть /login
2. Ввести email + пароль
3. Проверить API-запрос
```

5.4.3. GitHub Issues — документирование задач

```
[BUG] Некорректный заголовок на мобильной версии

Описание: навигационная панель перекрывает контент.
Приоритет: высокий

Шаги для воспроизведения:
1. Открыть страницу на смартфоне
2. Прокрутить вниз

Ожидаемое поведение: контент отображается корректно
```

5.4.4. Git для отчётности

- README.md — документация проекта
- Wiki — расширенная документация
- Issues — список задач
- Pull Requests — история изменений
- Releases — итоговые версии



Методические рекомендации

Тема: 6. Отчётность и аналитика

Навигация по разделам

6. Отчётность и аналитика

6.1. Способы сравнения показателей эффективности

Оценка эффективности веб-приложения основывается на анализе метрик, которые отражают техническое состояние и производительность системы до и после оптимизации. Основные показатели измеряются с помощью Chrome Lighthouse, Chrome DevTools, мониторинга сервера и анализа SQL-запросов.

6.1.1. Типовые метрики web-производительности

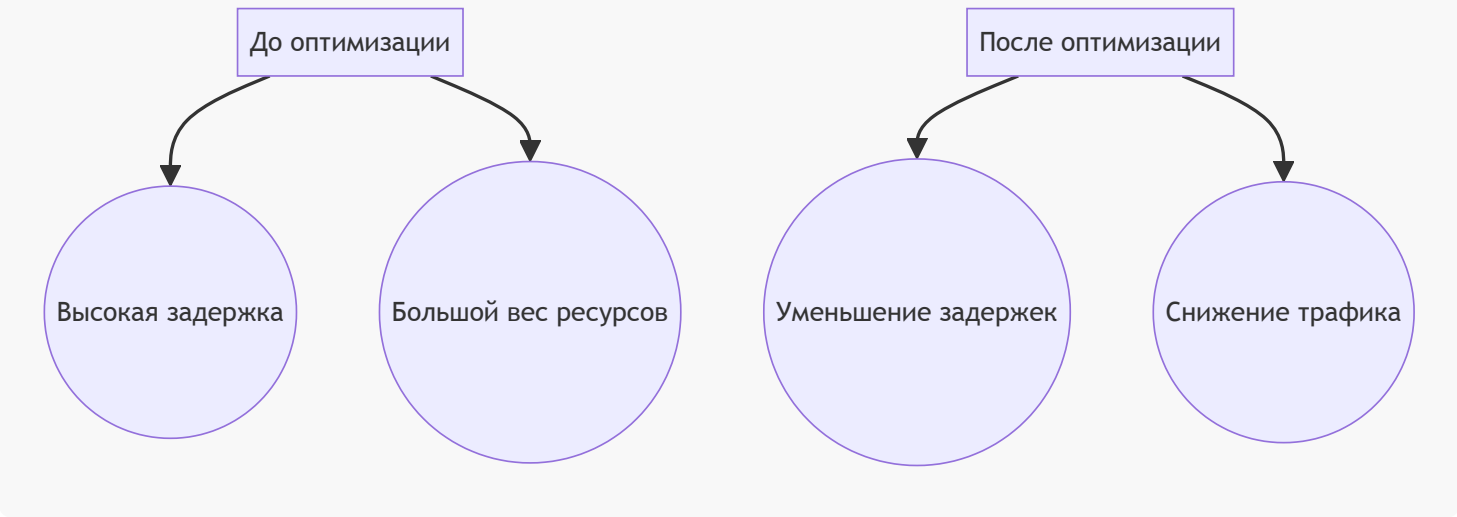
Метрика	Назначение
FCP (First Contentful Paint)	время появления первого контента
LCP (Largest Contentful Paint)	время отображения основного блока страницы
TTI (Time to Interactive)	время до полной интерактивности
Requests count	количество сетевых запросов
Total network weight	общий вес загруженных ресурсов
JS execution time	время выполнения JS-кода
CPU load	нагрузка на процессор
Memory usage	использование оперативной памяти
Server Response Time	задержки API / backend

Метрика	Назначение
SQL latency	время выполнения SQL-запросов

6.1.2. Пример измерений до и после оптимизации

```
FCP:         2.8s  → 1.4s
LCP:         4.2s  → 1.9s
JS Exec:     950ms → 380ms
Network:     3.4 MB → 1.1 MB
SQL-query:   240ms → 40ms (после добавления индекса)
```

6.1.3. Mermaid-диаграмма сравнения результатов



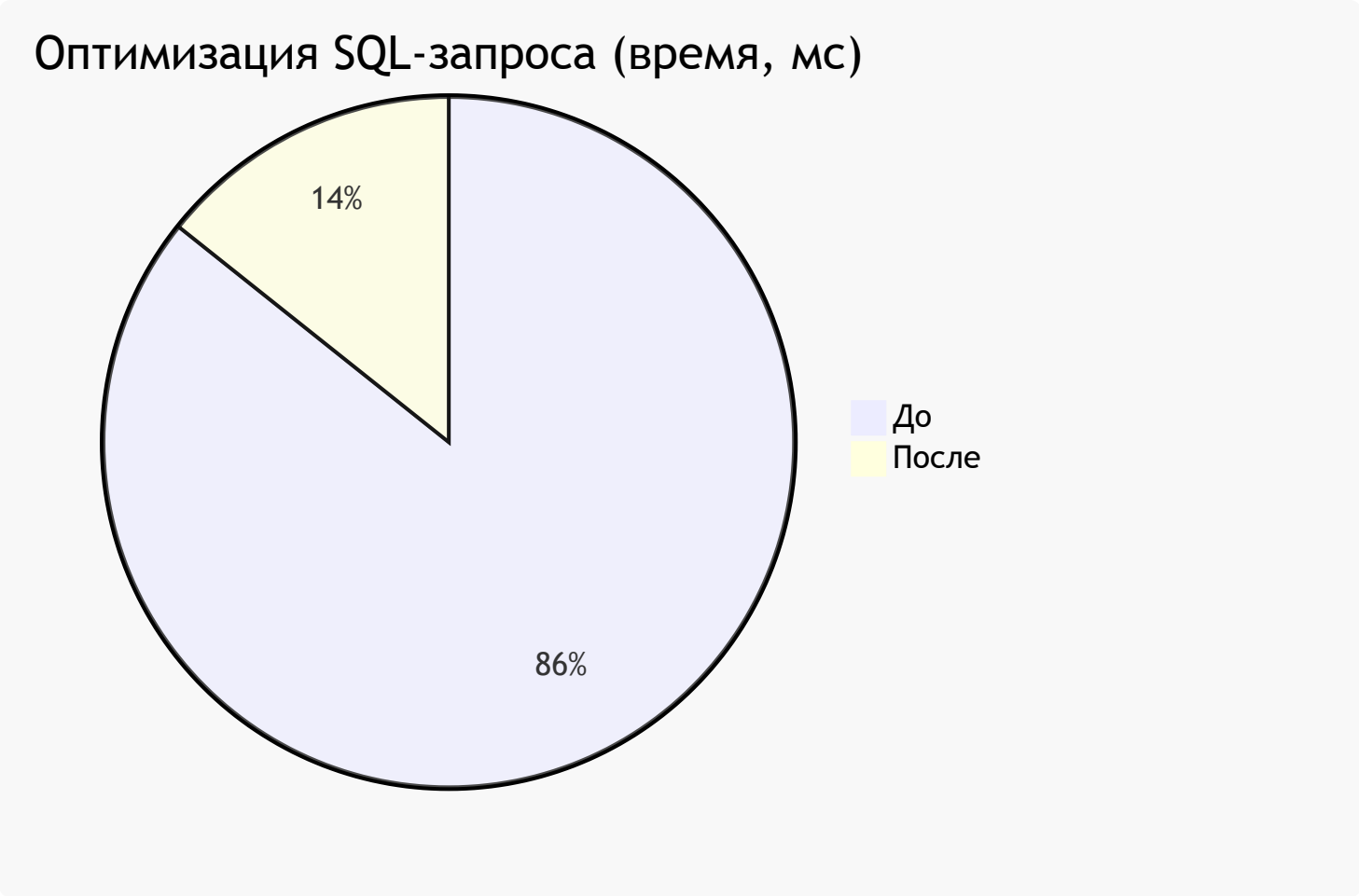
6.2. Подготовка таблиц «до / после оптимизации»

Сравнительные таблицы — один из основных инструментов аналитики. Они позволяют наглядно и объективно показать улучшения в производительности проекта.

6.2.1. Пример таблицы улучшений

Метрика	До	После	Улучшение
FCP	2.8 s	1.4 s	–50%
LCP	4.2 s	1.9 s	–54%
Вес JS	980 KB	320 KB	–67%
SQL-запрос /products	240 ms	40 ms	–83%
Количество запросов	37	21	–43%

6.2.2. Пример графика улучшений (Mermaid Pie)



6.2.3. Шаблон таблицы для отчёта

```
| Метрика | Значение до | Значение после | Разница | Комментарий |
|---------|-------------|----------------|---------|-------------|
|         |             |                |         |             |
|         |             |                |         |             |
```

6.3. Оформление технических отчётов, добавление скриншотов и EXPLAIN-планов

Технический отчёт должен быть структурированным, доказательным и воспроизводимым: по нему другой разработчик должен суметь повторить измерения и убедиться в результатах оптимизации.

6.3.1. Структура хорошего отчёта

1. **Введение**
    - цель оптимизации;
    - краткая характеристика системы.

2. **Методика измерений**
    - инструменты (Lighthouse, Chrome DevTools, PostgreSQL EXPLAIN);
    - условия тестирования (окружение, подключение, нагрузка).
3. **Результаты измерений**
    - таблицы;
    - графики;
    - скриншоты DevTools / Lighthouse.
4. **Что было оптимизировано**
    - какие ресурсы сжаты или объединены;
    - какие SQL-запросы переписаны;
    - какие индексы добавлены.
5. **EXPLAIN-анализ**
    - сравнение планов до и после оптимизации.
6. **Выводы**
    - итоговая оценка эффективности, выводы и рекомендации.

6.3.2. Пример EXPLAIN-плана (до оптимизации)

```
Seq Scan on products  (cost=0.00..1820.00 rows=90000 width=120)
  Filter: (category_id = 5)
```

Проблема: полное сканирование таблицы — отсутствует индекс.

6.3.3. EXPLAIN-план после оптимизации

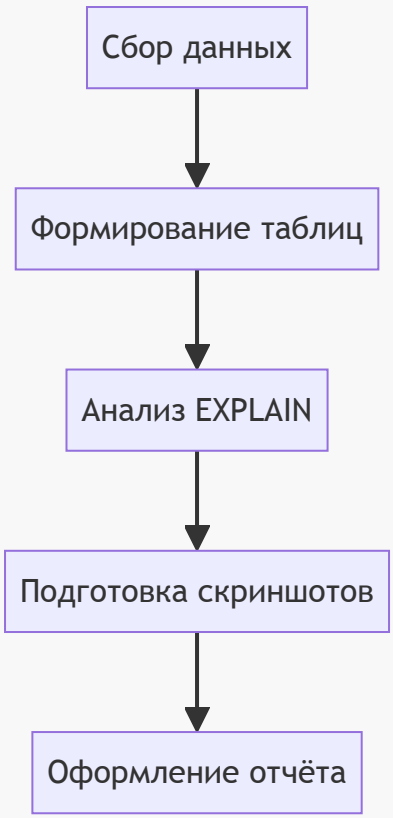
```
Index Scan using idx_products_category_id on products
  (cost=0.43..32.21 rows=120 width=120)
```

Количество обрабатываемых строк снизилось примерно в 700 раз, что приводит к существенному снижению времени выполнения запроса.

6.3.4. Пример вставки скриншота в отчёт

```
![Network Screenshot](screenshots/network-before.png)

На изображении видно, что вес страницы составляет 3.4 MB.
После оптимизации он уменьшился до 1.1 MB.
```

6.3.5. Mermaid-схема процесса подготовки отчёта



6.4. Корректное использование терминологии и академического стиля

Для официальной отчётности важно соблюдать академический стиль: формулировки должны быть точными, терминология — единой, а выводы — обоснованными.

6.4.1. Принципы академического стиля

- точные формулировки: *«SQL-запрос был оптимизирован путём добавления индекса»*;
- отсутствие разговорных выражений: не следует писать *«запрос тупил»*, *«страница лагала»*;
- логичная структура изложения, соблюдение структуры отчёта;
- корректные ссылки на инструменты (Chrome DevTools, PostgreSQL EXPLAIN, Lighthouse 11.1 и др.);
- единая терминологическая база во всём документе;
- объективность: *«Время выполнения уменьшилось на 40%»*, а не просто *«стало лучше»*.

6.4.2. Пример фрагмента отчёта

```
В результате оптимизации SQL-запроса
`SELECT * FROM products WHERE category_id = 5`
время выполнения уменьшилось с 240 мс до 40 мс.

Сокращение достигнуто благодаря созданию индекса:
```

```
CREATE INDEX idx_products_category_id ON products(category_id);

План выполнения изменился с Seq Scan на Index Scan,
что значительно уменьшило количество анализируемых строк.
```

Методические рекомендации

Тема: 7. Выполнение задания: Проектная документация

Навигация по разделам

7. Выполнение задания. Проектная документация

Цель работы. Студент должен разработать комплект проектной документации по своей предметной области, используя структуру и подходы, реализованные в тестовом проекте на предыдущей практике.

В ходе работы студент выполняет:

- анализ предметной области;
- построение архитектурной схемы;
- проектирование структуры базы данных;
- планирование разработки в виде диаграммы Ганта;
- подготовку итогового комплекта проектных материалов.

7.1. Подготовительный этап

Шаг 1. Изучение тестового проекта

Студент должен:

1. Скачать или просмотреть структуру проекта на GitHub:
2. `/docs`
3. `/src`
4. `/db`
5. `README.md`

Особое внимание обратить на то, как оформляются материалы:

- структура документации;

- использование Markdown;
- схемы в Mermaid;
- примеры описания предметной области;
- структура проекта.

Шаг 2. Определение закреплённого варианта предметной области

Студент выбирает или получает тему.

Примеры предметных областей:

- «Учёт заявок обслуживания оборудования»;
- «Продажа билетов»;
- «Бронирование ресурсов»;
- «Студенческий журнал».

Важно: тема должна быть относительно небольшой, но содержать достаточное количество сущностей и процессов.

7.2. Разработка проектной документации

7.2.1. Описание предметной области

Шаг 3. Сбор информации о предметной области

Студент должен подготовить ответы на вопросы:

- кто пользователи системы;
- какие действия они совершают;
- какие объекты существуют;
- какие данные необходимо хранить;
- какие процессы необходимо автоматизировать.

Шаг 4. Оформление описания в структурированном виде

Рекомендуемая структура (по аналогии с тестовым проектом):

Пример структуры документа domain.md

```
# 1. Предметная область

## 1.1. Краткое описание
Система предназначена для ...

## 1.2. Основные участники
```

```
- Пользователь …
- Администратор …

## 1.3. Основные процессы
1) Создание …
2) Обработка …
3) Хранение …
```

Пример (образец для студента):

Предметная область: учёт заявок на ремонт оборудования. Система позволяет принимать заявки, назначать ответственных, отслеживать статусы и формировать отчёты.

7.2.2. Разработка архитектурной схемы

Шаг 5. Определить компоненты системы

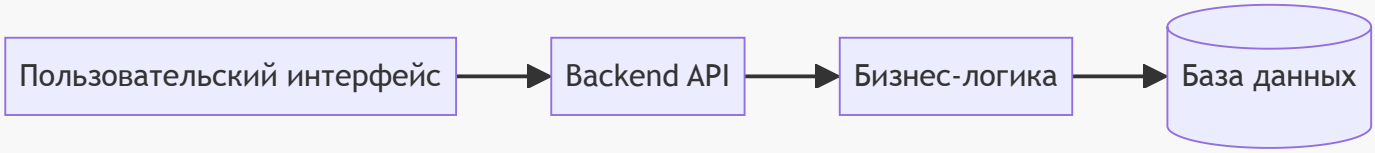
Компоненты могут включать:

- пользовательский интерфейс;
- модуль обработки данных;
- API-сервис;
- базу данных;
- внешние сервисы (при необходимости).

Шаг 6. Составить архитектурную схему

Схема строится на уровне компонентной модели (как в тестовом проекте).

Пример архитектурной схемы (Mermaid)



Шаг 7. Добавить схему в документацию

Файл для размещения архитектурной схемы:

- `docs/architecture.md`

7.2.3. Структура базы данных

Шаг 8. Определить сущности

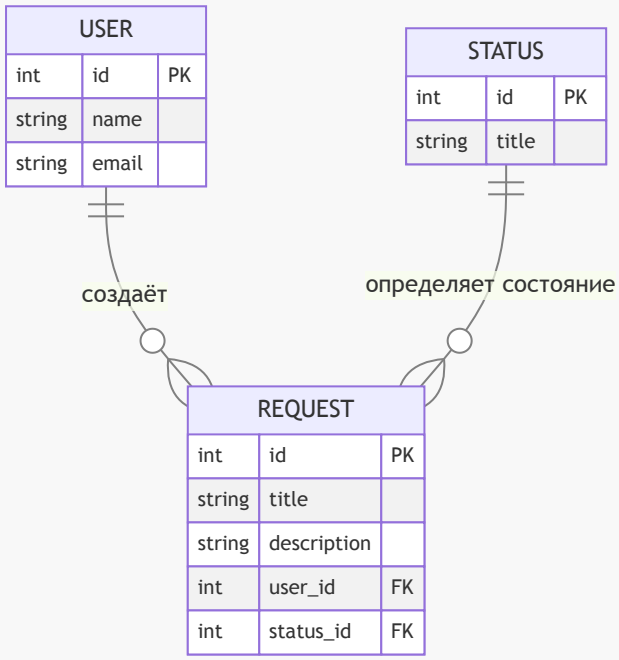
Студент должен перечислить сущности предметной области.

Пример:

- User;
- Request;
- Equipment;
- Status.

Шаг 9. Составить структуру таблиц (ER-диаграмму)

Пример ER-диаграммы (Mermaid ERD)



Шаг 10. Оценить необходимость оптимизации SQL

Если в предметной области предполагаются:

- фильтрация по нескольким полям;
- частые соединения таблиц (JOIN);
- сортировки по разным полям,

студент должен:

- предложить подходящие индексы;
- оценить и при необходимости оптимизировать связи;
- при необходимости скорректировать структуру таблиц.

Шаг 11. Вынести структуру БД в отдельный файл

Структура базы данных должна быть оформлена в виде:

- `db/schema.sql` — SQL-скрипт создания схемы;
- `docs/database.md` — текстовое описание структуры БД с диаграммами.

7.3. Подготовка плана разработки (диаграмма Ганта)

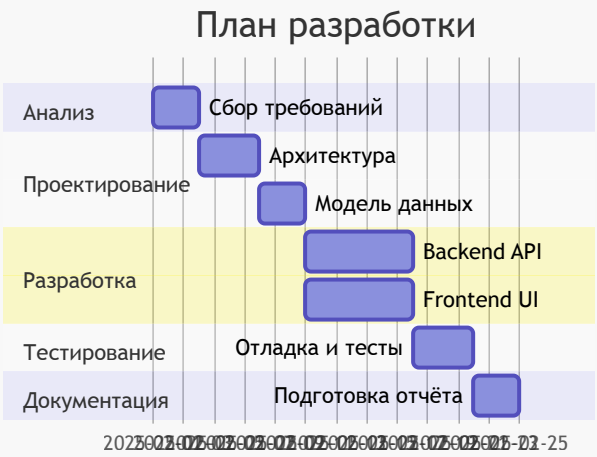
Шаг 12. Разбить разработку на этапы

Этапы должны соответствовать структуре проекта. Пример набора этапов:

1. Анализ требований;
2. Проектирование архитектуры;
3. Проектирование базы данных;
4. Создание чернового приложения;
5. Реализация функциональных модулей;
6. Тестирование;
7. Документирование;
8. Итоговая сборка.

Шаг 13. Создать диаграмму Ганта

Пример диаграммы Ганта (Mermaid)



Шаг 14. Описать диаграмму

Студент должен дать краткое текстовое описание диаграммы Ганта, включая:

- последовательность этапов разработки;
- зависимости между задачами (что начинается после чего);
- ключевые результаты каждого этапа (артефакты, которые должны быть получены).

Рекомендуется оформить диаграмму и её описание в отдельном файле `docs/plan.md`.

7.4. Итоговый комплект проектной документации

Студент должен собрать итоговый комплект проектных материалов в своём репозитории на GitHub, в составе обновлённого проекта.

Рекомендуемая структура репозитория:

```
/docs
   index.md
   domain.md        — описание предметной области
   architecture.md  — архитектурная схема
   database.md      — структура БД
   plan.md          — диаграмма Ганта и её описание

/src
/db
   schema.sql
README.md          — краткое описание проекта
```

Все файлы должны быть согласованы между собой: предметная область, архитектура, база данных и план разработки должны описывать одну и ту же систему и дополнять друг друга.

Методические рекомендации

Тема: 8. Выполнение задания: Дизайн интерфейсов

Навигация по разделам

8. Экспериментально-практическая работа: Дизайн интерфейсов

Цель работы. На основе ранее созданного прототипа информационной системы выполнить модернизацию интерфейса, улучшив его удобство, адаптивность, визуальную структуру и производительность.

Работа выполняется в одном из инструментов:

- Figma;
- Draw.io;
- Whimsical;
- Penpot;
- Adobe XD.

8.1. Подготовительный этап

Шаг 1. Собрать материалы ранее созданного прототипа

Студент должен подготовить:

- скриншоты текущей версии интерфейса или файл Figma/Draw.io;
- результаты анализа UX (если проводились ранее);
- описание логики экранов;

- структуру DOM (при фронтенд-разработке).

Цель шага: определить, что именно нужно оптимизировать.

8.2. Построение нового прототипа с оптимизацией интерфейса

Шаг 2. Создание или обновление основных пользовательских интерфейсов

Студент должен переработать:

- главный экран;
- навигационную область (меню / хедер / сайдбар);
- ключевые страницы: список, карточка, форма, поиск;
- пользовательские сценарии (User Flow → экран).

Требования к интерфейсам

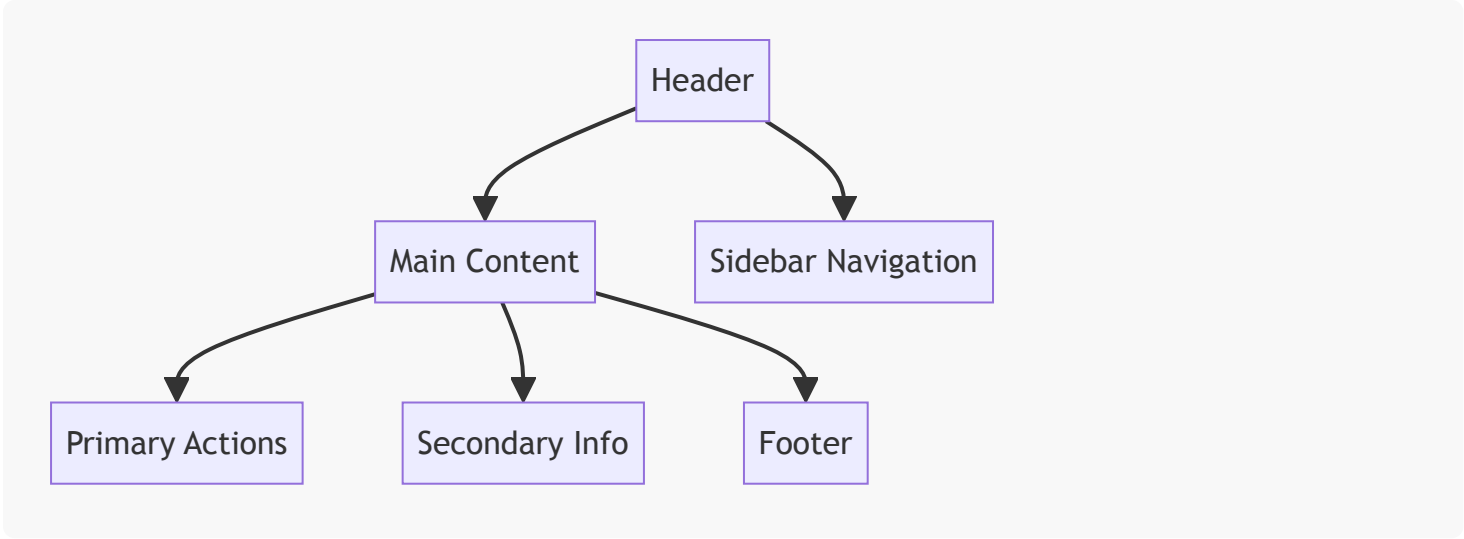
Каждый экран должен содержать:

- логичную визуальную иерархию;
- чистую структуру без лишних элементов;
- единый стиль (цвета, сетка, типографика);
- элементы управления с одинаковым ритмом, отступами и выравниванием.

Пример структуры главного экрана

```
[ Логотип ] [ Меню ] [ Профиль ]

_____________________________________________________

Большой заголовок
Краткое описание

[Основные действия]
[Второстепенные функции]

_____________________________________________________
Футер
```

Пример схемы (Mermaid Wireframe-style)



8.3. Адаптивные версии: Desktop / Tablet / Mobile

Шаг 3. Построить адаптивные макеты основных экранов

Для каждого ключевого экрана — минимум три варианта:

- Desktop (≥1280 px);
- Tablet (768–1024 px);
- Mobile (≤480–600 px).

Правила адаптации

Элемент	Desktop	Tablet	Mobile
Сетка	12 колонок	6–8 колонок	1–4 колонки
Меню	горизонтальное	бургер + укороченное меню	бургер
Карточки	3–4 в ряд	2 в ряд	1 в ряд
Текст	обычный размер	немного крупнее	ещё крупнее и короче

Макеты должны содержать:

- одинаковую логику навигации на всех устройствах;
- корректные отступы и визуальный ритм;
- читабельные элементы и тексты;
- доступные зоны взаимодействия (минимум 44×44 px).

8.4. Оптимизация элементов (упрощение, сокращение DOM)

Шаг 4. Анализ элементов интерфейса на избыточность

Студент должен найти:

- дублирующиеся кнопки;
- слишком глубокие блоки;
- тяжёлые UI-паттерны (сложные таблицы, большое количество вкладок);
- ненужные декоративные контейнеры;
- слишком длинные формы без логической группировки.

Оптимизация DOM

Пример до оптимизации:

```
div.container
  div.wrapper
    div.card
      div.card-header
        h2
```

После оптимизации:

```
section.card
  h2
```

Типовые решения по упрощению интерфейса

- заменить таблицы на карточки при просмотре данных;
- убрать лишние рамки, линии и разделители;
- заменить текстовые ссылки на иконки (при сохранении понятности);
- объединить соседние элементы в единый блок;
- скрыть второстепенную информацию под кнопкой «Подробнее».

8.5. Дизайнерские анимации и микроанимации

Шаг 5. Проработать микровзаимодействия в Figma / After Effects / ProtoPie

Типы микроанимаций

1. **Hover-эффекты**
    - затемнение кнопки;
    - изменение границы;
    - увеличение иконки.

2. **Feedback (обратная связь)**
    - всплывающие подсказки;
    - изменение состояния кнопки «Загружается...».
3. **Переходы между экранами**
    - Slide;
    - Fade;
    - Scale.
4. **Анимация элементов списка**
    - плавное появление элементов;
    - перемещение и сортировка с анимацией.

Пример описания анимации

Кнопка «Сохранить»

```
- Hover: увеличение масштаба до 1.03
- Active: сжатие масштаба до 0.98
- Disabled: 50% прозрачности
- Animation: 120–180 ms, ease-in-out
```

8.6. Оптимизация графики в дизайне

Шаг 6. Подготовить оптимизированные графические ресурсы

В требования входит:

- **Использование современных форматов:**
    - WebP, AVIF — для контентных изображений;
    - SVG — для иконок и интерфейсных элементов.
- **Предварительная компрессия:**
    - TinyPNG, Squoosh, ImageOptim и аналоги.
- **Масштабируемость:**
    - векторные элементы вместо растровых PNG;
    - ретина-варианты (2x, 3x) для сеточных изображений.
- **Responsive Images** (если макет предназначен для дальнейшей вёрстки):

Пример разметки responsive images

```
<img
  src="img-800.jpg"
```

```
  srcset="img-400.jpg 400w, img-800.jpg 800w, img-1600.jpg 1600w"
  sizes="(max-width: 600px) 400px, 800px"
  alt="Example"
/>
```

8.7. Итоговый комплект, который должен предоставить студент

Итоговый комплект включает:

1. описание предыдущей версии интерфейса;
2. перечень и описание выявленных проблем UX/UI;
3. новые экраны (вставить изображения или ссылки на макеты);
4. адаптивные макеты (Desktop / Tablet / Mobile);
5. анимации (GIF / видео — по возможности);
6. список произведённых улучшений с краткими обоснованиями;
7. примеры оптимизации графики (форматы, компрессия, responsive images);
8. ссылка на проект на GitHub со своей готовой работой и приложенной к нему документацией.

8.8. Пример тестового проекта

Для ориентира студентам рекомендуется изучить структуру, оформление документации, подходы к проектированию и пример реализации в готовом тестовом проекте.

Ссылка на тестовый проект:

https://github.com/OlgaKraven/GDRMI.PP.PM.03-testProject.git

Что рекомендуется изучить в тестовом проекте

- структуру каталогов (`/docs`, `/src`, `/db`);
- оформление документации в **Markdown**;
- использование диаграмм **Mermaid** в архитектуре и БД;
- пример проектных материалов (архитектура, предметная область, БД, план работ);
- простую реализацию пользовательского интерфейса;
- структуру кода и модулей — как организована логика проекта.

Зачем нужен пример

Он помогает студентам понять:

- как привести проект к единому стилю оформления;

- как структурировать документацию;
- какие файлы должны входить в итоговый комплект;
- как выполняется минимальная рабочая реализация;
- как оформлять GitHub-репозиторий для защиты практики.

Изучение примера позволяет избежать ошибок и быстрее оформить свою работу в соответствии с требованиями практики.