Пример небольшой части руководства программиста для простого веб-приложения, которое представляет собой планировщик задач.

Руководство программиста для веб-приложения "Todo List"

1. Введение

Todo List— это веб-приложение для создания и ведения личных заметок. Пользователи могут регистрироваться, создавать заметки, редактировать их, а так управлять их статусом.

Основные функции:

* регистрация и аутентификация пользователей;
* создание, редактирование и удаление заметок;
* фильтрация и сортировка заметок;
* изменение статуса;
* поиск заметок.

Технологический стек:

* Фронтенд: HTML, CSS,JavaScript,jQurey, Bootstrap;
* Бэкенд: PHP, MySQL;
* Запросы: AJAX;

2. Архитектура приложения

2.1 Общая схема

Клиент отправляет AJAX-запросы из браузера к серверным PHP-скриптам для выполнения задач, таких как аутентификация, взаимодействие с базой данных и управление контентом. Ответы возвращаются в формате JSON, чтобы клиент мог обработать их на своей стороне.

2.2 Взаимодействие компонентов

* Пользователь взаимодействует с интерфейсом через браузер, отправляя запросы.
* Серверная часть обрабатывает запросы с помощью AJAX и взаимодействует с PHP для получения или записи данных.
* PHP обрабатывает запросы и взаимодействует с MySQL.
* Результаты операций возвращаются в виде JSON-ответов или HTML.

2.3 Основные модели данных

signin\_db.php, signup\_db.php обработка регистрации и входа пользователей.

newtask\_bd.php, update-status.php, edit\_db.php, delete\_db.php создание, редактирование, удаление заметки.

connect.php соединение с базой данных

3. Разработка фронтенда

3.1 Установка окружения

Для начала разработки необходимо установить зависимости:

```bash

git clone https://github.com/username/blogmaster.git

cd blogmaster/frontend

npm install

npm start

```

Проект использует React и Redux для управления состоянием.

3.2 Структура проекта

```plaintext

/src

/components

PostList.js Компонент для отображения списка постов

Post.js Компонент отдельного поста

Comment.js Компонент комментария

/redux

store.js Хранилище Redux

actions.js Действия для управления состоянием

reducers.js Редьюсеры для обработки изменений состояния

/pages

HomePage.js Главная страница

PostPage.js Страница отдельного поста

LoginPage.js Страница входа

```

3.3 Взаимодействие с API

Для взаимодействия с бэкендом используется Axios. Пример запроса для получения списка постов:

```javascript

import axios from 'axios';

export const fetchPosts = async () => {

const response = await axios.get('/api/posts');

return response.data;

};

```

3.4 Стилизация

Стилизация компонентов реализована с помощью Bootstrap и кастомных CSS-модулей. В проекте подключен Bootstrap, но для дополнительных правок используются SCSS:

```css

/ src/styles/main.scss /

body {

background-color: f8f9fa;

}

```

4. Разработка бэкенда

4.1 Установка окружения

Для разработки бэкенда нужно настроить Node.js:

```bash

cd blogmaster/backend

npm install

npm run dev

```

Бэкенд работает на Express.js и использует MongoDB через Mongoose.

4.2 Маршруты API

POST /api/auth/login

Авторизация пользователя. Принимает `email` и `password`, возвращает JWT токен.

GET /api/posts

Возвращает список всех постов.

POST /api/posts

Создание нового поста (доступно только для авторизованных пользователей). Пример тела запроса:

```json

{

"title": "Мой первый пост",

"content": "Это контент моего первого поста."

}

```

4.3 Аутентификация

Используется JWT для авторизации. Токен генерируется при входе пользователя и передается в заголовках для всех защищённых маршрутов:

```javascript

const jwt = require('jsonwebtoken');

const generateToken = (user) => {

return jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, { expiresIn: '1h' });

};

```

4.4 Модели данных

Пример модели пользователя (Mongoose):

```javascript

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

email: { type: String, required: true, unique: true },

passwordHash: { type: String, required: true },

username: { type: String, required: true },

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('User', userSchema);

*```*

5. Тестирование

5.1 Юнит-тесты

Для тестирования используется Jest. Пример юнит-теста для функции аутентификации:

```javascript

const { generateToken } = require('../utils/auth');

test('generateToken создает корректный токен', () => {

const token = generateToken({ \_id: '123' });

expect(token).toBeDefined();

});

```

5.2 Интеграционные тесты

Для интеграционного тестирования API используется Supertest:

```javascript

const request = require('supertest');

const app = require('../app');

describe('GET /api/posts', () => {

it('должен вернуть список постов', async () => {

const response = await request(app).get('/api/posts');

expect(response.statusCode).toBe(200);

expect(response.body).toBeInstanceOf(Array);

});

});

```

Этот пример охватывает только основные аспекты документации веб-приложения. В зависимости от особенностей вашего проекта, содержание может быть расширено или адаптировано.