МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

кАФЕДРА 33

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  | подпись, дата |  | К.А. Жиданов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЕТ  
О ЛАБОРАТОРНОЙ РАБОТЕ

|  |
| --- |
| Лабораторная работа  «Создание собственного онлайн сервиса» |
| по дисциплине: Технологии и методы программирования |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. № | 3333 |  | подпись, дата |  | П.И. Марченков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

Оглавление

[1.Цель 3](#_Toc199106234)

[2. декомпозиция Функциональной модели IDEF0 3](#_Toc199106235)

[3.Описание функционально временного анализа 6](#_Toc199106236)

[4.Вывод 8](#_Toc199106237)

# **1.Цель работы**

Создать сайта и телеграмм бота с общей базой данных, используя нейронные сети. Сайт представляет из себя выбран планировщик задач с возможностью авторизации.

2. код сайта

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>To-Do List</title>

<style>

body {

font-family: Arial, sans-serif;

}

#todoList {

border-collapse: collapse;

width: 70%;

margin: 0 auto;

}

#todoList th, #todoList td {

border: 1px solid #61b31f;

padding: 8px;

text-align: left;

}

#todoList th {

background-color: #be1010;

}

#todoList th:first-child, #todoList th:last-child {

width: 50%;

}

#todoList th:nth-child(2) {

width: 90%;

}

.add-form {

margin-top: 50px;

width: 70%;

margin: 50px auto;

}

.add-form input[type="text"] {

padding: 8px;

width: 70%;

}

.add-form button {

padding: 400px;

width: 100%;

}

</style>

</head>

<body>

<h2 style="text-align: center;">To-Do List от Петра Марченкова</h2>

<table id="todoList">

<thead>

<tr>

<th>№</th>

<th>Задача</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="listBody">

{{rows}}

</tbody>

</table>

<div class="add-form">

<input type="text" id="newItem" placeholder="Enter new item">

<button onclick="addItem()">Добавить</button>

</div>

<script>

function addItem() {

const input = document.getElementById('newItem');

const text = input.value.trim();

if (!text) return;

fetch('/add-item', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ text })

}).then(() => location.reload());

}

function deleteItem(id) {

fetch(`/delete-item/${id}`, {

method: 'DELETE'

}).then(() => location.reload());

}

function editItem(id) {

const newText = prompt('новая задача:');

if (!newText || newText.trim() === '') return;

fetch(`/edit-item/${id}`, {

method: 'PUT',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ text: newText.trim() })

}).then(() => location.reload());

}

</script>

</body>

</html>

Index.js

const http = require('http');

const fs = require('fs');

const path = require('path');

const mysql = require('mysql2/promise');

const cookieSession = require('cookie-session');

const crypto = require('crypto');

const PORT = process.env.PORT || 3000;

const dbConfig = {

host: process.env.DB\_HOST,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_DATABASE,

};

const pool = mysql.createPool(dbConfig);

async function retrieveListItems() {

const connection = await mysql.createConnection(dbConfig);

const [rows] = await connection.execute('SELECT id, text FROM items');

await connection.end();

return rows;

}

async function getHtmlRows() {

const rows = await retrieveListItems();

return rows.map((item, index) => `

<tr>

<td>${index + 1}</td>

<td>${item.text}</td>

<td>

<button onclick="editItem(${item.id})">редактировать</button>

<button onclick="deleteItem(${item.id})">удалить</button>

</td>

</tr>

`).join('');

}

async function addItemToDb(text) {

const connection = await mysql.createConnection(dbConfig);

const [result] = await connection.execute('INSERT INTO items (text) VALUES (?)', [text]);

await connection.end();

return { id: result.insertId, text };

}

async function deleteItemFromDb(id) {

const connection = await mysql.createConnection(dbConfig);

await connection.execute('DELETE FROM items WHERE id = ?', [id]);

await connection.end();

}

async function updateItemInDb(id, newText) {

const connection = await mysql.createConnection(dbConfig);

await connection.execute('UPDATE items SET text = ? WHERE id = ?', [newText, id]);

await connection.end();

}

const server = http.createServer(async (req, res) => {

const parsedUrl = url.parse(req.url, true);

const cookies = cookie.parse(req.headers.cookie || '');

const isLoggedIn = cookies.auth === 'admin';

// === LOGIN PAGE ===

if (req.method === 'GET' && parsedUrl.pathname === '/login') {

const loginPage = fs.readFileSync(path.join(\_\_dirname, 'login.html'));

res.writeHead(200, { 'Content-Type': 'text/html' });

return res.end(loginPage);

}

if (req.method === 'POST' && parsedUrl.pathname === '/login') {

let body = '';

req.on('data', chunk => body += chunk);

req.on('end', () => {

const params = new URLSearchParams(body);

const username = params.get('username');

const password = params.get('password');

if (username === 'admin' && password === 'admin') {

res.writeHead(302, {

'Set-Cookie': cookie.serialize('auth', 'admin', { httpOnly: true }),

'Location': '/'

});

return res.end();

} else {

res.writeHead(401);

return res.end('Unauthorized');

}

});

return;

}

// === AUTH CHECK ===

if (!isLoggedIn) {

res.writeHead(302, { Location: '/login' });

return res.end();

}

// === MAIN PAGE ===

if (req.method === 'GET' && parsedUrl.pathname === '/') {

const htmlPath = path.join(\_\_dirname, 'index.html');

let html = fs.readFileSync(htmlPath, 'utf8');

const rows = await getHtmlRows();

html = html.replace('{{rows}}', rows);

res.writeHead(200, { 'Content-Type': 'text/html' });

return res.end(html);

}

// === ADD ITEM ===

if (req.method === 'POST' && parsedUrl.pathname === '/add-item') {

let body = '';

req.on('data', chunk => body += chunk);

req.on('end', async () => {

const { text } = JSON.parse(body);

const result = await addItemToDb(text);

res.writeHead(200, { 'Content-Type': 'application/json' });

res.end(JSON.stringify({ success: true, item: result }));

});

return;

}

// === DELETE ITEM ===

if (req.method === 'DELETE' && parsedUrl.pathname.startsWith('/delete-item/')) {

const id = parsedUrl.pathname.split('/').pop();

await deleteItemFromDb(id);

res.writeHead(200, { 'Content-Type': 'application/json' });

return res.end(JSON.stringify({ success: true }));

}

// === EDIT ITEM ===

if (req.method === 'PUT' && parsedUrl.pathname.startsWith('/edit-item/')) {

const id = parsedUrl.pathname.split('/').pop();

let body = '';

req.on('data', chunk => body += chunk);

req.on('end', async () => {

const { text } = JSON.parse(body);

await updateItemInDb(id, text);

res.writeHead(200, { 'Content-Type': 'application/json' });

res.end(JSON.stringify({ success: true }));

});

return;

}

// === STATIC FILES ===

if (req.method === 'GET') {

const filePath = path.join(\_\_dirname, parsedUrl.pathname);

if (fs.existsSync(filePath)) {

const ext = path.extname(filePath).toLowerCase();

const contentType = ext === '.js' ? 'text/javascript' : 'text/plain';

res.writeHead(200, { 'Content-Type': contentType });

return res.end(fs.readFileSync(filePath));

}

}

// === 404 ===

res.writeHead(404);

res.end('Not Found');

});

server.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

Login.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login</title>

</head>

<body>

<h2 style="text-align:center;">Login</h2>

<form method="POST" action="/login" style="text-align:center;">

<input type="text" name="username" placeholder="Username" required />

<input type="password" name="password" placeholder="Password" required />

<button type="submit">Login</button>

</form>

</body>

</html>

Рис. 1. Модель IDEF0

3. Код бота

import asyncio

import logging

import mysql.connector

import random

from aiogram import Bot, Dispatcher, types, F

from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup, ReplyKeyboardMarkup, KeyboardButton

from aiogram.filters import CommandStart, Command

from aiogram.enums import ParseMode

API\_TOKEN = '695946746:AAHI81UwhW1SpSwgKViJ8z3n8Xhfm0tcoQI' # 🔐 Замени на свой токен

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API\_TOKEN)

dp = Dispatcher()

sessions = {}

QUOTES = [

"Действуй! Даже если страшно.",

"Ты справишься — я верю.",

"Одна задача за раз!",

"Маленький шаг — тоже движение.",

]

# DB функции

def db\_conn():

return mysql.connector.connect(

host='localhost',

user='root',

password='root',

database='todolist'

)

def get\_tasks():

conn = db\_conn()

cursor = conn.cursor()

cursor.execute("SELECT id, text FROM items")

tasks = cursor.fetchall()

conn.close()

return tasks

def add\_task(text):

conn = db\_conn()

cursor = conn.cursor()

cursor.execute("INSERT INTO items (text) VALUES (%s)", (text,))

conn.commit()

conn.close()

def delete\_task(task\_id):

conn = db\_conn()

cursor = conn.cursor()

cursor.execute("DELETE FROM items WHERE id = %s", (task\_id,))

conn.commit()

conn.close()

def update\_task(task\_id, new\_text):

conn = db\_conn()

cursor = conn.cursor()

cursor.execute("UPDATE items SET text = %s WHERE id = %s", (new\_text, task\_id))

conn.commit()

conn.close()

# Главное меню

def get\_main\_menu():

return ReplyKeyboardMarkup(

keyboard=[

[KeyboardButton(text="📋 Список задач"), KeyboardButton(text="➕ Добавить задачу")],

[KeyboardButton(text="💬 Цитата"), KeyboardButton(text="📤 Переслать список")]

],

resize\_keyboard=True

)

# /start

@dp.message(CommandStart())

async def start(message: types.Message):

sessions[message.chat.id] = {"step": "login"}

await message.answer("Введите логин:")

# /quote

@dp.message(Command("quote"))

async def quote(message: types.Message):

await message.answer(random.choice(QUOTES))

# /add (альтернатива кнопке)

@dp.message(Command("add"))

async def add(message: types.Message):

chat\_id = message.chat.id

session = sessions.get(chat\_id, {})

if not session.get("authenticated"):

await message.answer("⛔ Сначала авторизуйтесь через /start")

return

session["step"] = "add\_task"

await message.answer("Введите текст новой задачи:")

# /forward

@dp.message(Command("forward"))

async def forward(message: types.Message):

chat\_id = message.chat.id

session = sessions.get(chat\_id, {})

if not session.get("authenticated"):

await message.answer("⛔ Сначала авторизуйтесь через /start")

return

args = message.text.split()

if len(args) < 2:

await message.answer("Использование: /forward <user\_id>")

return

user\_id = int(args[1])

tasks = get\_tasks()

text = "\n".join([f"{t[0]}: {t[1]}" for t in tasks])

try:

await bot.send\_message(chat\_id=user\_id, text=f"📋 Список задач:\n{text}")

await message.answer("✅ Список отправлен.")

except Exception as e:

await message.answer(f"Ошибка: {e}")

# Обработка обычных сообщений

@dp.message(F.text)

async def handle\_text(message: types.Message):

chat\_id = message.chat.id

session = sessions.get(chat\_id, {})

# ⛔ Проверка авторизации

if not session.get("authenticated") and session.get("step") not in ["login", "password"]:

await message.answer("⛔ Сначала авторизуйтесь через /start")

return

# Авторизация

if session.get("step") == "login":

session["login"] = message.text.strip()

session["step"] = "password"

await message.answer("Введите пароль:")

return

if session.get("step") == "password":

session["password"] = message.text.strip()

if session["login"] == "admin" and session["password"] == "admin":

session["authenticated"] = True

session["step"] = "choose\_task"

await message.answer("✅ Успешный вход!", reply\_markup=get\_main\_menu())

else:

await message.answer("❌ Неверный логин или пароль. Попробуйте снова: /start")

sessions.pop(chat\_id, None)

return

# Добавление новой задачи

if session.get("step") == "add\_task":

task\_text = message.text.strip()

add\_task(task\_text)

await message.answer("✅ Задача добавлена!", reply\_markup=get\_main\_menu())

session["step"] = "choose\_task"

return

# Редактирование задачи

if session.get("step") == "edit\_task":

task\_id = session.get("edit\_id")

update\_task(task\_id, message.text.strip())

await message.answer("✅ Задача обновлена.", reply\_markup=get\_main\_menu())

session["step"] = "choose\_task"

return

# Обработка кнопок меню

if message.text == "📋 Список задач":

await show\_tasks\_menu(message)

elif message.text == "➕ Добавить задачу":

session["step"] = "add\_task"

await message.answer("Введите текст новой задачи:")

elif message.text == "💬 Цитата":

await message.answer(random.choice(QUOTES))

elif message.text == "📤 Переслать список":

await message.answer("Используйте команду: /forward <user\_id>")

# Callback-кнопки (редактирование и удаление)

@dp.callback\_query()

async def task\_chosen(callback: types.CallbackQuery):

chat\_id = callback.message.chat.id

session = sessions.get(chat\_id, {})

# ⛔ Проверка авторизации

if not session.get("authenticated"):

await callback.message.answer("⛔ Сначала авторизуйтесь через /start")

return

data = callback.data

if data.startswith("delete\_"):

task\_id = int(data.split("\_")[1])

delete\_task(task\_id)

await callback.message.answer("🗑 Задача удалена.")

await show\_tasks\_menu(callback.message)

elif data.startswith("edit\_"):

task\_id = int(data.split("\_")[1])

session["step"] = "edit\_task"

session["edit\_id"] = task\_id

await callback.message.answer("✏️ Введите новый текст задачи:")

# Показ задач с кнопками

async def show\_tasks\_menu(message):

chat\_id = message.chat.id

session = sessions.get(chat\_id, {})

# ⛔ Проверка авторизации

if not session.get("authenticated"):

await message.answer("⛔ Сначала авторизуйтесь через /start")

return

session["step"] = "choose\_task"

tasks = get\_tasks()

if not tasks:

await message.answer("📭 Список задач пуст.")

return

for task\_id, text in tasks:

keyboard = InlineKeyboardMarkup(

inline\_keyboard=[

[

InlineKeyboardButton(text="✏️ Редактировать", callback\_data=f"edit\_{task\_id}"),

InlineKeyboardButton(text="🗑 Удалить", callback\_data=f"delete\_{task\_id}")

]

]

)

await message.answer(f"<b>{text}</b>", reply\_markup=keyboard, parse\_mode=ParseMode.HTML)

# Запуск

if \_\_name\_\_ == '\_\_main\_\_':

asyncio.run(dp.start\_polling(bot))

4. структура базы данных

**CREATE DATABASE IF NOT EXISTS todolist;**

USE todolist;

CREATE TABLE items (

id INT AUTO\_INCREMENT PRIMARY KEY,

text VARCHAR(255) NOT NULL

);

****5. Обзор возможностей сайта****

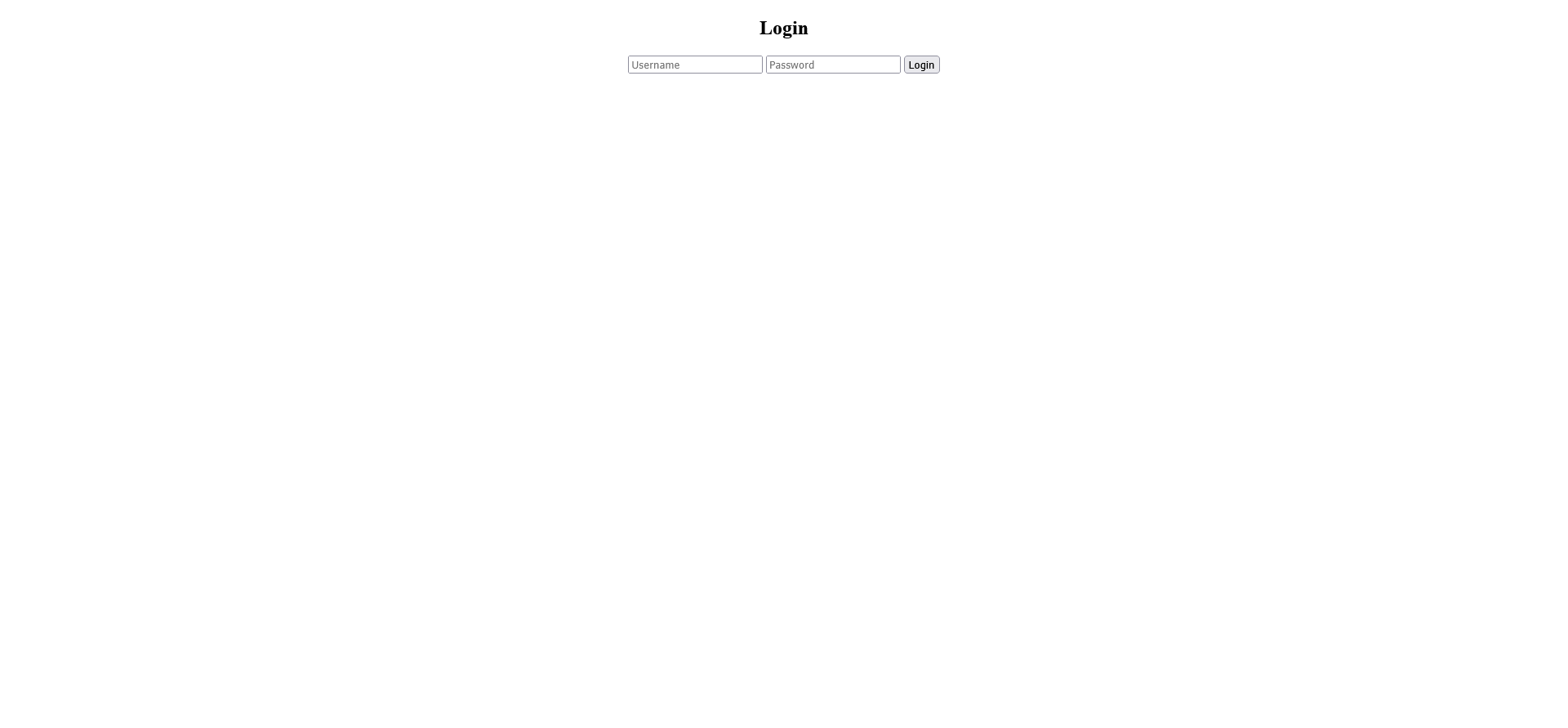


Рис.1-аутентификация

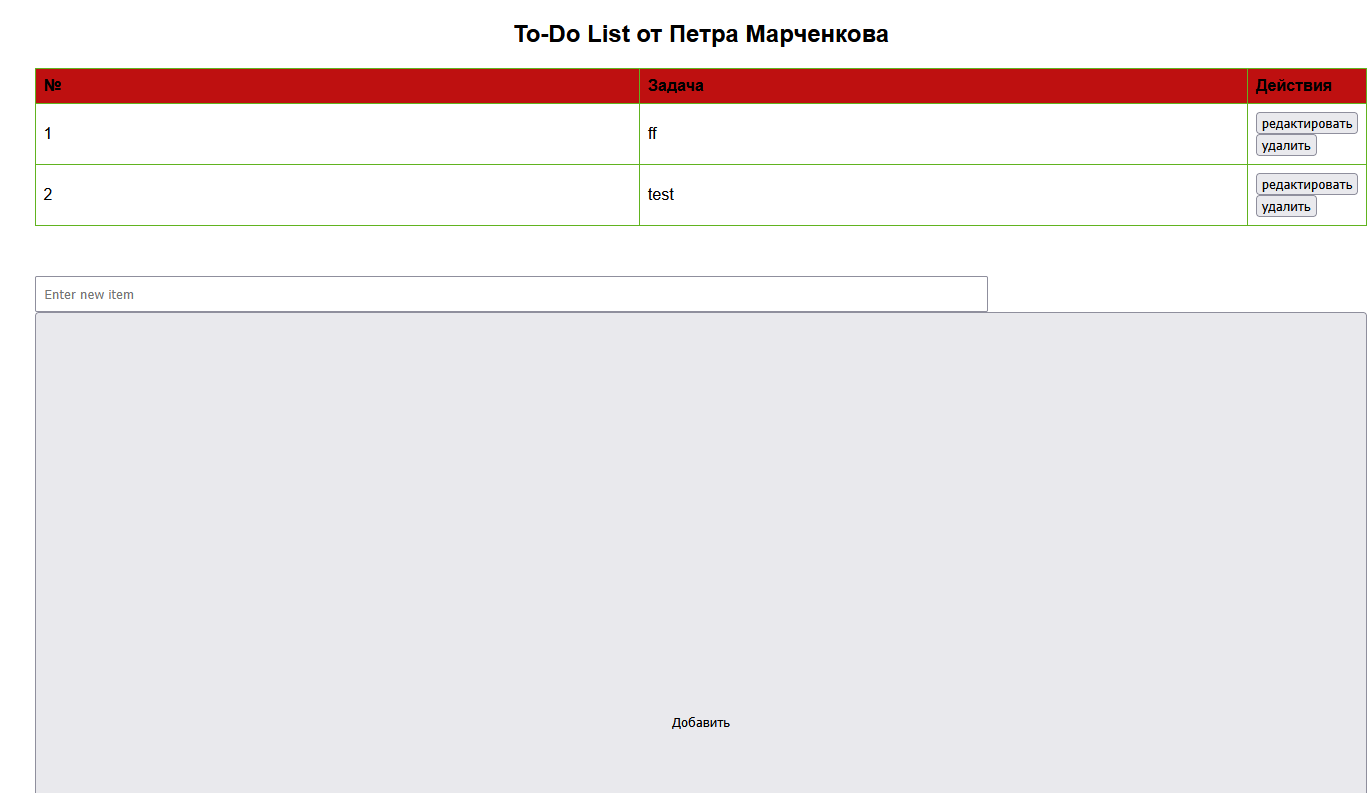


Рис.2-сайт

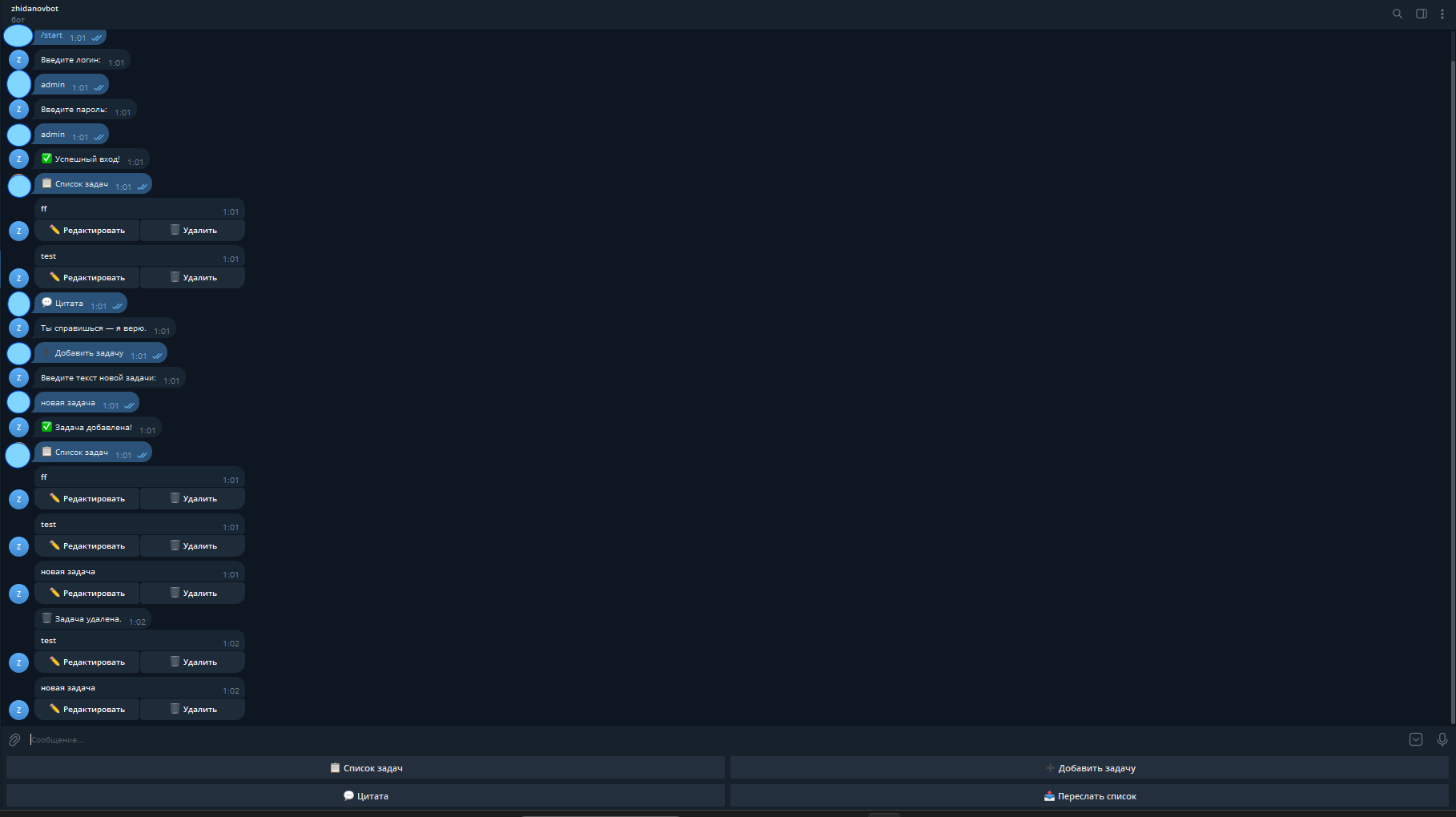


Рис.3-демонстрация работы бота

****6. развертывание сайта и бота****

Я использовал railway для сайта и amvera для бота

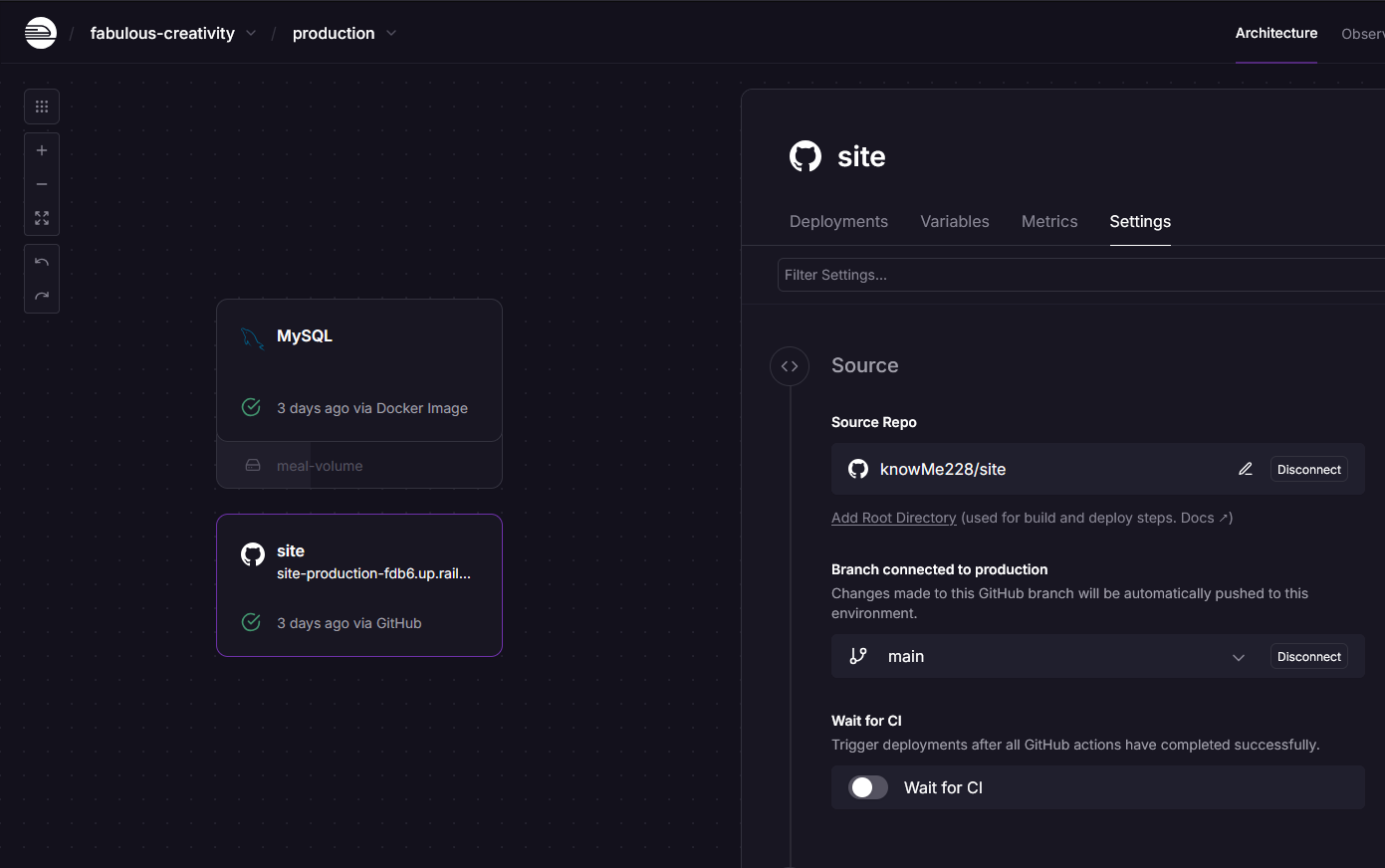


Рис.4-структура хостинга

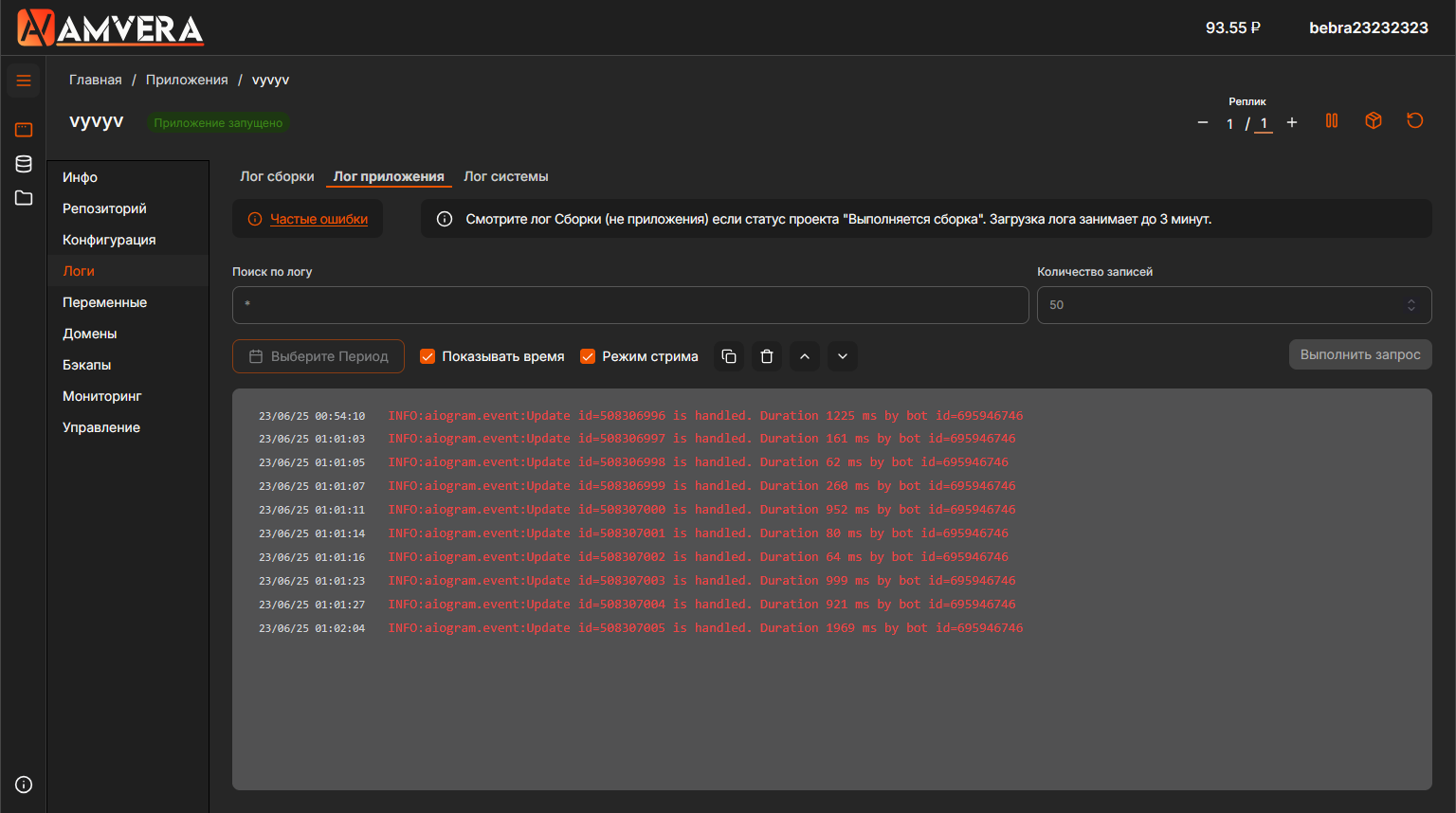


Рис.5-хостинг бота

4. Промпты для получения кода

Для бота

напиши telegram бота на python который будет осуществлять следующие функции вдобавок к работе to-do list сайта:

1 подключаться к mysql предусмотрена авторизация, проходить авторизацию по установленным credentials.

Сайт представляет из себя простой to-do list,в котором можно добавлять,редактировать,удалять items

2 Добавить item в to-do list

3 Редактировать item в to-do list

4 Удалить to-do list

5 показать весь to-do list в наглядном виде

6 переслать другому человеку to-do list

7 еще какие нибудь прикольные функции по твоему усмотрению

отправь весь код,ведь у меня нет рук

8 добавь удобное меню выбора функций бота

Для сайта

Ниже я вставляю начальный код To-Do листа, который я уже начал писать. Пожалуйста, дополни этот код, добавив в него следующие функции:

1. Возможность \*\*добавлять\*\* новую задачу. Она должна отображаться в списке.

2. Возможность \*\*редактировать\*\* уже добавленную задачу (например, по двойному клику или по кнопке «Редактировать»).

3. Возможность \*\*удалять\*\* задачу (например, по нажатию кнопки с крестиком).

4. Возможность авторизации по логину и паролю

Важно:

- Сохрани общий стиль и структуру кода.

- Не переписывай весь проект с нуля — только \*\*дополни\*\* уже имеющийся код.

- Если считаешь нужным, добавь краткие комментарии к новым частям.

- Не меняй существующую верстку и стили, если это не обязательно.

Вот исходный код:

5.Вывод

**В ходе данной лабораторной работы была достигнута цель создания сайта и бота с общей базой данных с использованием нейросетей. Сайт доступен по этой ссылке** <https://site-production-fdb6.up.railway.app/> **, а бот** <https://t.me/EgdjfbdsfagfklulukdgaBot>**.**