



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

ПРАКТИЧЕСКАЯ РАБОТА №3

по дисциплине «Проектирование и разработка серверных частей интернет-ресурсов»

Студент группы ИКБО-21-23

Муравьев А.О.

(подпись студента)

Руководитель практической работы

Благирев М.М.

(подпись руководителя)

Работа представлена

«___»_____2025 г.

Допущен к работе

«___»_____2025 г.

Москва 2025

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ	3
ХОД РАБОТЫ	5
ВЫВОД	12
ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ ...	ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.

ЦЕЛЬ РАБОТЫ

В задании предлагается создать сложную серверную конфигурацию, состоящую из связки apache+nginx+C#+База данных. В данной конфигурации предполагается создание как минимум 3 элементов(контейнеров) или использование как основы серверной конфигурации, созданной в практической работе №1. В этой конфигурации предполагается акселерированное проксирование без кэширования. Схематично предполагаемый алгоритм работы изображен на рисунке.

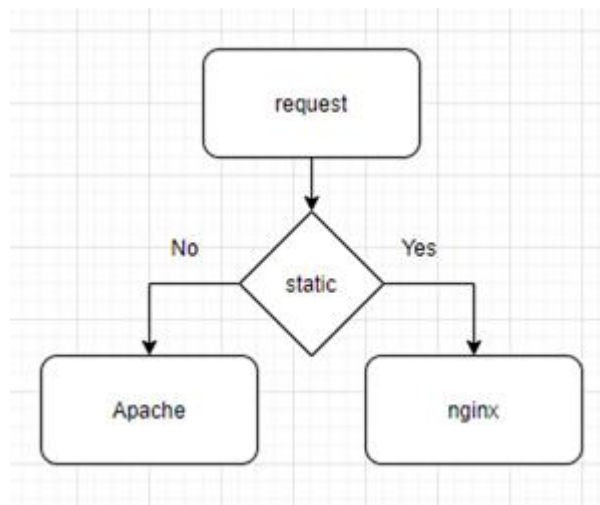


Рисунок 1 – Схематичное изображение предполагаемой функциональности

Предполагается, что сервер nginx будет отображать статический контент, а apache динамический и в связке мы получим быстроедействие и эффективную систему.

Также по необъяснимым обстоятельствам следует изменить root-директории с базовых на другие.

Для доступа к администрированию предусмотреть базовую авторизацию и аутентификацию с применением htaccess и хранением пользователей в БД (без использования htpasswd). Для тестирования данной конфигурации предполагается создать тестовое веб-приложение на тему по варианту: Кофейня.

Тестовое веб-приложение предполагает создание как минимум 2 вебстраниц со статическим контентом и двух веб-страниц с динамическим контентом: взятом из базы данных, например.

ХОД РАБОТЫ

Структура веб-приложения

Веб-приложение будет состоять из пяти страниц. Двух статических (главная страница и страница «О нас»). Двух динамических страниц (страница заказа с меню и страница всех заказов). И одной POST-страницы (страница после отправки заказа).

Реализована предложенная структура веб-приложения (рис. 1).

Nginx обрабатывает запросы, возвращает статические страницы и дёргает Apache при запросе динамики.

Apache устанавливает root-директорию, CGI-директорию, обрабатывает авторизацию на сервере (HTTP Basic Auth + PostgreSQL). Управляет доступом.

CGI-приложение, написанное на C#, обрабатывает GET и POST-запросы. Реализует динамику сайта.

База данных на PostgreSQL хранит в себе данные.

Итого получается 3 контейнера:

1. Nginx,
2. Apache + CGI C#,
3. PostgreSQL.

Реализация веб-приложения

Nginx обрабатывает пути. Если путь состоит из «/» или «/about», то он загружает статические файлы index.html или about.html соответственно (root директорию загружает сам Apache).

В случае столкновения с путями «/menu», «/order» (точное совпадение) или «/admin/» (частичное совпадение) Nginx дёргает Apache и создаёт GET-запрос для CGI-скрипта.

Схематичное представление работы конфигурации Nginx показано на рисунке 2.

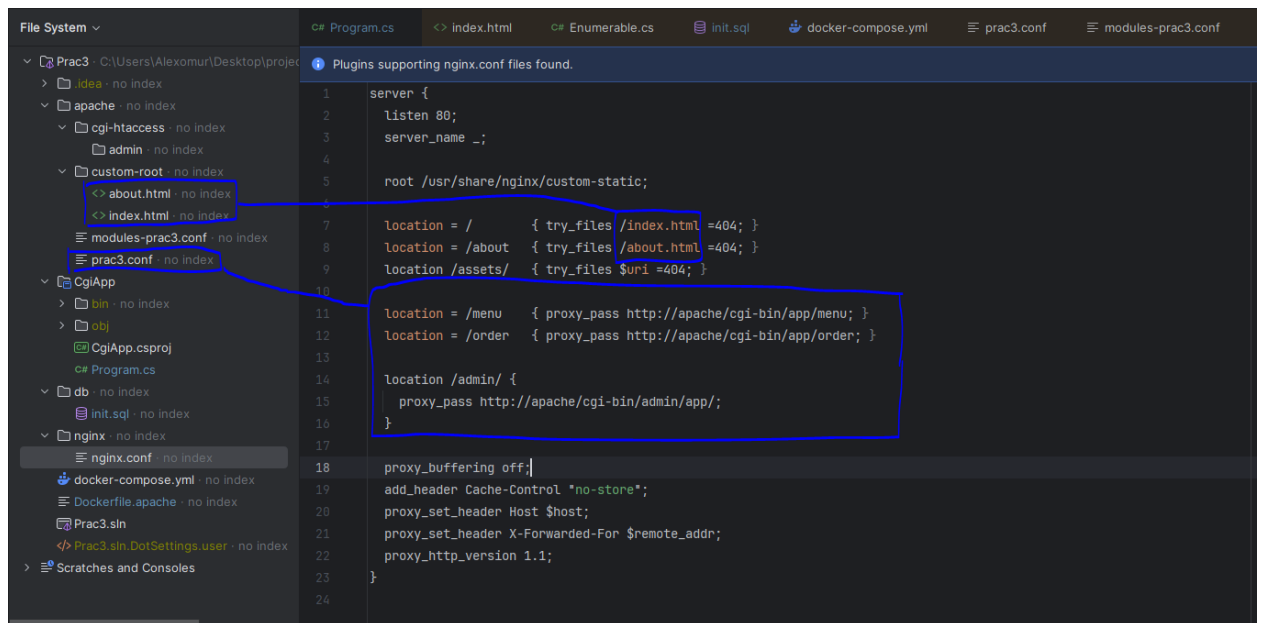


Рисунок 2 – Схематичное представление работы конфигурации Nginx

Конфигурация Apache отвечает за загрузку статической/корневой (root) директории, загрузку CGI, настройки авторизации (и связанной с авторизацией CGI) и загрузку PostgreSQL.

Конфигурация Apache (prac3.conf) представлена на рисунке 3.

```
index.html  C# Enumerable.cs  init.sql  docker-compose.yml  prac3.conf  modules-prac3.conf  about.html
Plugins supporting *.conf files found.
1 DocumentRoot "/var/www/custom-root"
2 <Directory "/var/www/custom-root">
3     Options Indexes FollowSymLinks
4     AllowOverride None
5     Require all granted
6 </Directory>
7 DirectoryIndex index.html
8
9 ScriptAlias /cgi-bin/ "/usr/local/apache2/cgi-bin/"
10 <Directory "/usr/local/apache2/cgi-bin/">
11     Options +ExecCGI
12     AllowOverride None
13     Require all granted
14 </Directory>
15
16 <Directory "/usr/local/apache2/cgi-bin/admin">
17     Options +ExecCGI
18     AllowOverride None
19     AuthType Basic
20     AuthName "Admin Area"
21     AuthBasicProvider dbd
22     AuthDBUserPWQuery "SELECT '{SHA}' || encode(digest(password, 'sha1'), 'base64') FROM users WHERE username = %s"
23     Require valid-user
24 </Directory>
25
26 AcceptPathInfo On
27
28 DBDriver pgsqldb
29 DBDParams "host=db port=5432 user=app password=secret dbname=coffee"
30 DBDMin 1
31 DBDKeep 2
32 DBDMax 10
33 DBDExptime 300
34
35 Header always set X-Powered-By "Apache+CGI+.NET"
```

Рисунок 3 – Конфигурация Apache

CGI C# скрипт запускается Apache. Apache загружает все нужные данные (метод запроса, путь, другие данные о запросе) в переменные окружения.

В C# скрипте необходимо прописать всю логику любых запросов. В том числе секретную.

В данном случае авторизацией полностью занимается Apache + PostgreSQL. Запрос с админским доступом приходит только, если была пройдена авторизация в Apache.

Фрагмент кода (отвечающего за обработку POST-запроса) представлен на рисунке 4.

```
C# Program.cs x  <> index.html  C# Enumerable.cs  init.sql  docker-compose.yml  prac3.conf  modules-prac3.conf  <> about.html  C# CgiApp.csp

73     try
106     else if (method == "POST" && string.Equals(pathInfo, "/order", StringComparison.OrdinalIgnoreCase))
108         IDictionary<string, string> form = await ReadFormUrlEncodedAsync();
109         string customer;
110         string coffeeIdStr;
111         string qtyStr;
112         if (!form.TryGetValue("customer_name", out customer) ||
113             !form.TryGetValue("coffee_id", out coffeeIdStr) ||
114             !form.TryGetValue("qty", out qtyStr))
115         {
116             WriteHeader("400 Bad Request");
117             Console.Write(Html(title: "Ошибка", body: "<p>Некорректные данные заказа.</p>"));
118         }
119     else
120     {
121         int coffeeIdParsed;
122         int qtyParsed;
123         if (!int.TryParse(coffeeIdStr, out coffeeIdParsed) || !int.TryParse(qtyStr, out qtyParsed) || qtyParsed <= 0)
124         {
125             WriteHeader("400 Bad Request");
126             Console.Write(Html(title: "Ошибка", body: "<p>Некорректные данные заказа.</p>"));
127         }
128     else
129     {
130         using (NpgsqlConnection db = await OpenDbAsync())
131         using (NpgsqlCommand cmd = new NpgsqlCommand(cmdText: "insert into orders (customer_name, coffee_id, qty) values (@c, @id, @q)", db))
132         {
133             cmd.Parameters.AddWithValue("c", customer);
134             cmd.Parameters.AddWithValue("id", coffeeIdParsed);
135             cmd.Parameters.AddWithValue("q", qtyParsed);
136             await cmd.ExecuteNonQueryAsync();
137         }
138         WriteHeader();
139         Console.Write(Html(title: "Заказ принят", body: "<p>Спасибо! Заказ записан.</p><p><a href='\"*/menu\"'>Назад к меню</a></p>"));
140     }
141 }
142 }
```

Рисунок 4 – Фрагмент CGI C# скрипта

Для корректной работы программы необходимо заполнить базу данных некой структурой и какими-то данными. Фрагмент init-sql-скрипта представлен на рисунке 5.

```
index.html  C# Enumerable.cs  init.sql x  docker-compose.yml  prac3.conf  modules-prac3.conf  <> about.html  C# C

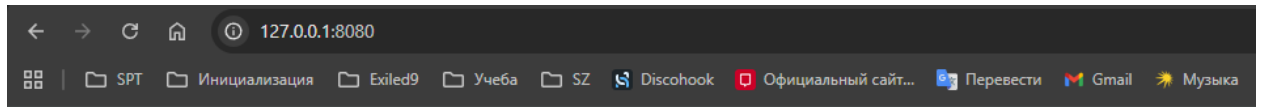
⚠ No data sources are configured to run this SQL and provide advanced code assistance.

15     create table if not exists orders (
18         coffee_id int not null references coffee(id),
19         qty int not null check (qty > 0),
20         created_at timestampz not null default now()
21     );
22
23     insert into users (username, password) values
24     ( username 'admin', password 'admin123')
25     on conflict (username) do update set password = excluded.password;
26
27     insert into coffee (name, price_cents) values
28     ( name 'Эспрессо', price_cents 150),
29     ( name 'Американо', price_cents 200),
30     ( name 'Капучино', price_cents 250),
31     ( name 'Латте', price_cents 300)
32     on conflict do nothing;
```

Рисунок 5 – Фрагмент init-sql-скрипта

Результат работы программы

Результат открытия пути «/» показан на рисунке 6.



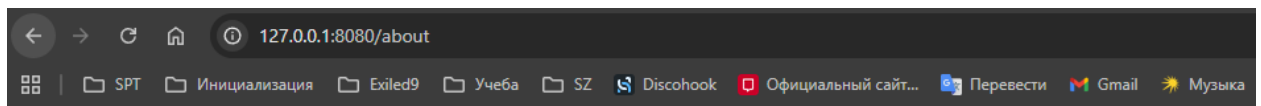
Кофейня

Добро пожаловать! Посмотрите [меню](#) и сделайте заказ.

О нас — [тут](#).

Рисунок 6 – Результат открытия пути «/»

На рисунке 7 показан результат открытия пути «/about».



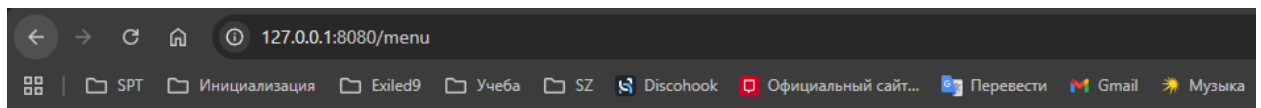
О кофейне

Наша маленькая кофейня с большим сердцем. Свежеобжаренный кофе и уют.

[На главную](#)

Рисунок 7 – Результат открытия пути «/about»

На рисунке 8 представлено открытие пути /menu (динамическая страница).



Меню кофейни

ID	Напиток	Цена
1	Эспрессо	1.50 Р
2	Американо	2.00 Р
3	Капучино	2.50 Р
4	Латте	3.00 Р

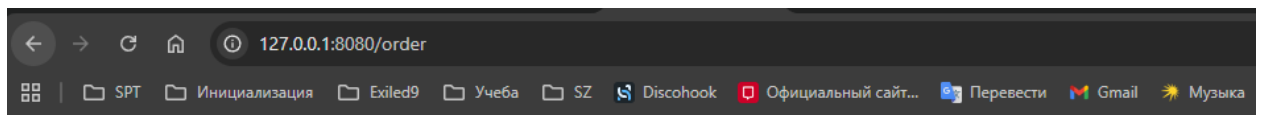
Сделать заказ

Имя:
ID кофе:
Кол-во:

[На главную](#)

Рисунок 8 – Результат открытия пути «/menu»

На рисунке 9 представлен результат отправки POST-запроса с информацией о заказанном кофе.



Заказ принят

Спасибо! Заказ записан.

[Назад к меню](#)

[На главную](#)

Рисунок 9 – Результат отправки POST-запроса

На рисунке 10 представлено окно авторизации при попытке войти в «Admin Area».

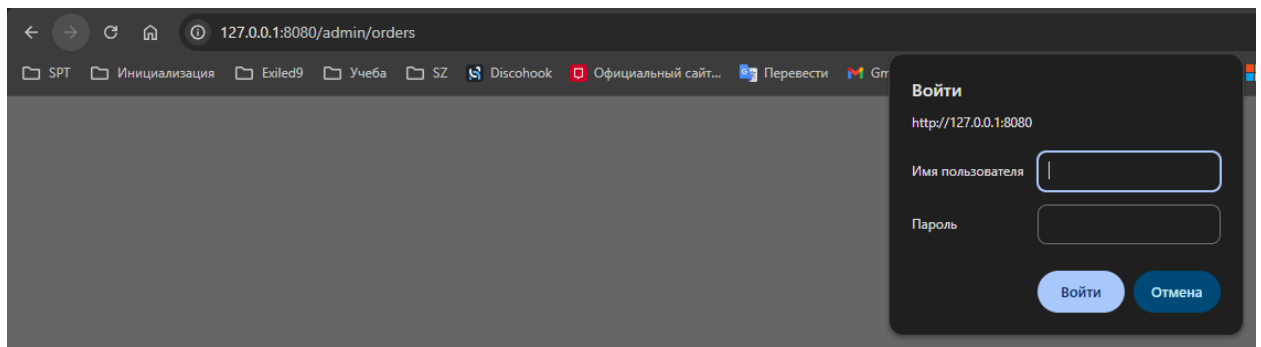
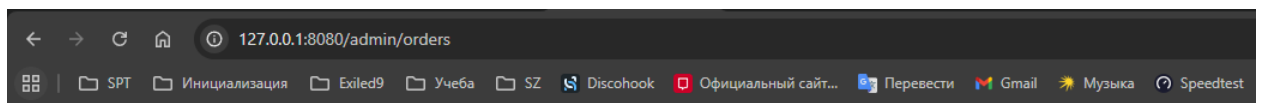


Рисунок 10 – Окно авторизации

На рисунке 11 представлена динамическая страница с последними заказами.



Админ: последние заказы

#	Клиент	Напиток	Кол-во	Когда
4	Муравьев А.О.	Латте	1	2025-09-16 20:36
3	fffff	Американо	1	2025-09-16 18:46
2	Имя 2	Американо	5	2025-09-12 18:07
1	Имя1	Эспрессо	2	2025-09-12 18:07

[На главную](#)

Рисунок 11 – Динамическая страница с данными о последних заказах

ВЫВОД

Таким образом, была разработана сложная серверная конфигурация, состоящая из связки Apache+CGI C#+Nginx+PostgreSQL.

Исходный код проекта расположен по адресу:

<https://github.com/alexomur/MireaBackend/tree/master/Prac3>