вввввввввввввввввМинистeрство нaуки и высшeго обрaзовaния Российской Фeдeрaции

Фeдeрaльноe госудaрствeнноe бюджeтноe обрaзовaтeльноe учрeждeниe высшeго обрaзовaния  
«МОСКОВСКИЙ ГОСУДAРСТВEННЫЙ УНИВEРСИТEТ ГEОДEЗИИ И КAРТОГРAФИИ» (МИИГAиК)

Фaкультeт Гeоинформaтики и информaционной бeзопaсности

Кaфeдрa Информaционно-измeритeльных систeм

Тип прaктики: Учебная

Нaзвaниe прaктики: Учебная (ознакомительная) практика по веб-программированию

Отчeт по учебной (ознакомительной) практике по веб-программированию

НAПРAВЛEНИE 09.03.02 ИНФОРМAЦИОННЫE СИСТEМЫ И ТEХНОЛОГИИ

|  |  |
| --- | --- |
| Студeнт группы 2023-ФГИиБ-ИСиТ-1б | Галстян С.С. |
| Руководитeль учебной прaктики,  прeподaвaтeль кaфeдры ИИС | Рaмaзaнов Э.Р. |

Москвa 2025

**СОДЕРЖАНИЕ**

[Введение 3](#__RefHeading___Toc203400440)

[Основная часть 5](#__RefHeading___Toc203400441)

[1.1. Подготовка и настройка среды разработки. 5](#__RefHeading___Toc203400442)

[*1.1.1.* *Операционная система* 5](#__RefHeading___Toc203400443)

[*1.1.2.* *Среды разработки (IDE) и инструменты* 5](#__RefHeading___Toc203400444)

[*1.1.3.* *Система управления базами данных (СУБД)* 5](#__RefHeading___Toc203400445)

[*1.1.4.* *Библиотеки и фреймворки* 6](#__RefHeading___Toc203400446)

[*1.1.5.* *Дополнительные инструменты* 6](#__RefHeading___Toc203400447)

[*1.1.6.* *Виртуализация* 6](#__RefHeading___Toc203400448)

[1.2. Реализация вёрстки страницы и алгоритмов frontend. 7](#__RefHeading___Toc203400449)

[1.2.1. Вёрстка интерфейса 7](#__RefHeading___Toc203400450)

[1.2.2. Логика фронтенда (JavaScript) 8](#__RefHeading___Toc203400451)

[1.2.3. Взаимодействие с API 9](#__RefHeading___Toc203400452)

[1.3. Программная реализация алгоритмов backend. 11](#__RefHeading___Toc203400453)

[1.3.1. Работа с базой данных (Database) 11](#__RefHeading___Toc203400454)

[1.3.2. Аутентификация (AuthService) 12](#__RefHeading___Toc203400455)

[1.3.3. API (ChatApp) 13](#__RefHeading___Toc203400456)

[1.3.4. Запуск сервера 14](#__RefHeading___Toc203400457)

[1.4. Сборка и развёртывание клиента и сервера 15](#__RefHeading___Toc203400458)

[1.4.1. Инструменты сборки 15](#__RefHeading___Toc203400459)

[1.4.2. Процесс сборки серверной части 15](#__RefHeading___Toc203400460)

[1.4.3. Развёртывание клиентской части 15](#__RefHeading___Toc203400461)

[1.4.4. Тестовое развёртывание 16](#__RefHeading___Toc203400462)

[1.4.5. Переменные окружения 16](#__RefHeading___Toc203400463)

[1.5. Тестирование 17](#__RefHeading___Toc203400464)

[1.5.1. Ручное тестирование (тест-кейсы) 17](#__RefHeading___Toc203400465)

[1.5.2. Автоматизированное тестирование 18](#__RefHeading___Toc203400466)

[ЗАКЛЮЧЕНИЕ 21](#__RefHeading___Toc203400467)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#__RefHeading___Toc203400468)

# Введение

**Общая характеристика предприятия**

Я проходил(а) практику в «МИИГАиК» на базе кафедры «Информaционно-измeритeльных систeм».Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный университет геодезии и картографии» (МИИГАиК) – один из ведущих вузов России, осуществляющий подготовку высококвалифицированных специалистов в области геодезии, картографии, геоинформатики и смежных направлений. Университет обладает современной материально-технической базой и активно участвует в научно-исследовательской деятельности, в том числе в разработке и внедрении инновационных технологий в сфере геопространственных данных и информационных систем.

**Общая характеристика отдела**

Учебная практика проходила на базе кафедры **«Информационно-измерительных систем» (ИИС)**, которая является ключевым структурным подразделением факультета геоинформатики и информационной безопасности МИИГАиК. Кафедра специализируется на подготовке специалистов в области информационных технологий, геоинформационных систем, обработки пространственных данных и обеспечения информационной безопасности.

Основные направления деятельности кафедры включают:

1. Разработку и внедрение информационно-измерительных систем для обработки геопространственных данных.
2. Исследования в области цифровизации и автоматизации процессов сбора, хранения и анализа данных.
3. Поддержку и модернизацию программного обеспечения, используемого в учебном и научном процессах.
4. Внедрение современных технологий веб-программирования и распределённых систем.

В рамках отдела **обеспечения телекоммуникационного оборудования** решаются задачи, связанные с поддержкой IT-инфраструктуры, включая:

* Обеспечение бесперебойной работы серверного и сетевого оборудования.
* Мониторинг и диагностику состояния информационных систем.
* Внедрение новых технологий для повышения эффективности работы программно-аппаратных комплексов.

Практика была направлена на ознакомление с современными технологиями веб-разработки, включая проектирование клиентской и серверной частей приложений, а также их тестирование и развертывание.

# Основная часть

# Подготовка и настройка среды разработки.

1. В рамках учебной практики мной было получено индивидуальное задание по разработке простой интернет-магазин (каталог+корзина). Отображение каталога с пагинацией, добавление товаров в корзину. Форма корзины: отображение добавленных товаров и их количества, расчет суммы корзины.

Для выполнения проекта была настроена следующая среда разработки:

### *Операционная система*

* Windows 11

### *Среды разработки (IDE) и инструменты*

* Visual Studio Code (VSCode) – основной редактор кода.

Использование: написание кода на Python, HTML, CSS, JavaScript, работа с системой контроля версий Git.

### *Система управления базами данных (СУБД)*

* PostgreSQL – встраиваемая реляционная СУБД.

Использование: хранение данных о товаре, пользователях, корзине и прочее.

### *Библиотеки и фреймворки*

**Backend (C++)**

* FastAPI (фреймворк для Python) – использовался для создания сервера

Функционал: обработка запросов, маршрутизация, middleware для аутентификации.

* JWT– библиотека для работы с JSON Web Tokens (JWT).

Использование: генерация и валидация токенов аутентификации.

**Frontend**

* HTML5 & CSS3 – верстка интерфейса чата и формы входа.
* JavaScript (ES6) – взаимодействие с API, динамическое обновление сообщений.

### *Дополнительные инструменты*

* Git & GitHub – система контроля версий.

В результате была настроена полноценная среда разработки, включающая серверную часть на Python (FastAPI, PostgreSQL, JWT), фронтенд (HTML/CSS/JS) и инструменты для отладки.

## Реализация вёрстки страницы и алгоритмов frontend.

В рамках разработки онлайн-чата была реализована клиентская часть приложения, включающая:

* **Адаптивную вёрстку** (HTML/CSS)
* **Динамическое взаимодействие с API** (JavaScript)
* **Регистрация и авторизация**
* **Добавление товара админом в каталог**
* **Добавление товара в корзину**
* **Просмотр магазина**
* **Просмотр профиля**

**И многое другое**

### Вёрстка интерфейса

Основные страницы:

**Страница каталога** (index.html):

* + Товары
  + Фильтры
  + Пагинация

1. {% extends "base.html" %}
2. {% block content %}
3. <div *class*="container mt-4">
4. <div *class*="row mb-4">
5. <div *class*="col-12">
6. <div *class*="card mb-4">
7. <div *class*="card-body">
8. <div *class*="row">
9. <div *class*="col-md-6">
10. <h5 *class*="card-title">Категории</h5>
11. <div *class*="d-flex flex-wrap">
12. <a *href*="/shop?gender={% if current\_gender %}{{ current\_gender }}{% endif %}"
13. *class*="btn btn-sm {% if not current\_category %}btn-dark{% else %}btn-outline-dark{% endif %} me-2 mb-2">
14. Все категории
15. </a>
16. {% for category in categories %}
17. <a *href*="/shop?category={{ category.value }}{% if current\_gender %}&gender={{ current\_gender }}{% endif %}"
18. *class*="btn btn-sm {% if current\_category == category.value %}btn-dark{% else %}btn-outline-dark{% endif %} me-2 mb-2">
19. {{ category.value }}
20. </a>
21. {% endfor %}
22. </div>
23. </div>
25. <div *class*="col-md-6">
26. <h5 *class*="card-title">Пол</h5>
27. <div *class*="d-flex flex-wrap">
28. <a *href*="/shop?category={% if current\_category %}{{ current\_category }}{% endif %}"
29. *class*="btn btn-sm {% if not current\_gender %}btn-dark{% else %}btn-outline-dark{% endif %} me-2 mb-2">
30. Все
31. </a>
32. {% for gender in genders %}
33. <a *href*="/shop?gender={{ gender.value }}{% if current\_category %}&category={{ current\_category }}{% endif %}"
34. *class*="btn btn-sm {% if current\_gender == gender.value %}btn-dark{% else %}btn-outline-dark{% endif %} me-2 mb-2">
35. {{ gender.value }}
36. </a>
37. {% endfor %}
38. </div>
39. </div>
40. </div>
41. </div>
42. </div>
43. </div>
44. </div>
46. <div *class*="row">
47. {% for product in products %}
48. <div *class*="col-md-4 col-6 mb-4">
49. <div *class*="card h-100 shadow-sm">
50. {% if product.image\_url %}
51. <img *src*="{{ product.image\_url }}"
52. *class*="card-img-top p-3"
53. *alt*="{{ product.alt\_text }}"
54. *style*="height: 250px; object-fit: contain;"
55. *onerror*="**this**.src='https:*//via.placeholder.com/300x300?text=No+Image'; this.classList.add('bg-light')*">
56. {% else %}
57. <div *class*="card-img-top bg-light d-flex align-items-center justify-content-center" *style*="height: 250px;">
58. <span *class*="text-muted">Нет изображения</span>
59. </div>
60. {% endif %}
61. <div *class*="card-body">
62. <h5 *class*="card-title">{{ product.name }}</h5>
63. {% if product.brand %}
64. <p *class*="text-muted small mb-2">{{ product.brand }}</p>
65. {% endif %}
67. <div *class*="d-flex flex-wrap gap-1 mb-2">
68. <span *class*="badge bg-primary">{{ product.category }}</span>
69. <span *class*="badge bg-info">{{ product.gender }}</span>
70. {% if product.is\_featured %}
71. <span *class*="badge bg-warning text-dark">Рекомендуем</span>
72. {% endif %}
73. </div>
75. <div *class*="mb-2">
76. {% if product.has\_discount %}
77. <span *class*="text-danger fw-bold h5">{{ product.final\_price }} ₽</span>
78. <span *class*="text-decoration-line-through text-muted small">{{ product.price }} ₽</span>
79. {% else %}
80. <span *class*="fw-bold h5">{{ product.final\_price }} ₽</span>
81. {% endif %}
82. </div>
84. <div *class*="small text-muted">
85. {% if product.material %}
86. <div>Материал: {{ product.material }}</div>
87. {% endif %}
88. {% if product.color %}
89. <div>Цвет: {{ product.color }}</div>
90. {% endif %}
91. </div>
92. </div>
94. <div *class*="card-footer bg-white border-0">
95. <a *href*="/product/{{ product.id }}" *class*="btn btn-dark w-100">
96. <i *class*="bi bi-eye"></i> Подробнее
97. </a>
98. </div>
99. </div>
100. </div>
101. {% else %}
102. <div *class*="col-12">
103. <div *class*="alert alert-info">
104. <i *class*="bi bi-info-circle"></i> Товары не найдены. Попробуйте изменить параметры фильтрации.
105. </div>
106. </div>
107. {% endfor %}
108. </div>
109. <nav *aria-label*="Page navigation">
110. <ul *class*="pagination justify-content-center">
111. <li *class*="page-item {% if not pagination.has\_prev %}disabled{% endif %}">
112. <a *class*="page-link"
113. *href*="/shop?page={{ pagination.prev\_num }}{% if current\_category %}&category={{ current\_category }}{% endif %}{% if current\_gender %}&gender={{ current\_gender }}{% endif %}"
114. *aria-label*="Previous">
115. <span *aria-hidden*="true">&laquo;</span>
116. </a>
117. </li>
118. {% for p in pagination.page\_range %}
119. <li *class*="page-item {% if p == pagination.page %}active{% endif %}">
120. <a *class*="page-link"
121. *href*="/shop?page={{ p }}{% if current\_category %}&category={{ current\_category }}{% endif %}{% if current\_gender %}&gender={{ current\_gender }}{% endif %}">
122. {{ p }}
123. </a>
124. </li>
125. {% endfor %}
126. <li *class*="page-item {% if not pagination.has\_next %}disabled{% endif %}">
127. <a *class*="page-link"
128. *href*="/shop?page={{ pagination.next\_num }}{% if current\_category %}&category={{ current\_category }}{% endif %}{% if current\_gender %}&gender={{ current\_gender }}{% endif %}"
129. *aria-label*="Next">
130. <span *aria-hidden*="true">&raquo;</span>
131. </a>
132. </li>
133. </ul>
134. </nav>
135. </div>
136. {% endblock %}

*Стили*:

* + CSS-переменные для цветовой схемы

**Страница авторизации** (login.html):

* + Форма входа
  + Валидация полей

1. {% extends "base.html" %}
2. {% block title %}Вход | Skyline Riot{% endblock %}
3. {% block content %}
4. <div *class*="container mt-5">
5. <div *class*="row justify-content-center">
6. <div *class*="col-lg-5 col-md-7">
7. <div *class*="card border-0 shadow-lg">
8. <div *class*="card-header bg-dark text-white text-center py-3">
9. <h3 *class*="mb-0">
10. <i *class*="bi bi-box-arrow-in-right"></i> Вход в аккаунт
11. </h3>
12. </div>
13. <div *class*="card-body p-4">
14. {% if error %}
15. <div *class*="alert alert-danger border-0 shadow-sm mb-4">
16. <i *class*="bi bi-exclamation-triangle-fill me-2"></i> {{ error }}
17. </div>
18. {% endif %}
20. <form *method*="POST" *class*="needs-validation" *novalidate*>
21. <div *class*="mb-4">
22. <label *class*="form-label fw-bold">Email \*</label>
23. <div *class*="input-group">
24. <span *class*="input-group-text bg-light">
25. <i *class*="bi bi-envelope"></i>
26. </span>
27. <input *type*="email" *name*="email" *class*="form-control"
28. *placeholder*="example@mail.com" *required*>
29. </div>
30. </div>
32. <div *class*="mb-4">
33. <label *class*="form-label fw-bold">Пароль \*</label>
34. <div *class*="input-group">
35. <span *class*="input-group-text bg-light">
36. <i *class*="bi bi-lock"></i>
37. </span>
38. <input *type*="password" *name*="password" *class*="form-control"
39. *placeholder*="Введите пароль" *required*>
40. </div>
41. </div>
43. <div *class*="d-flex justify-content-between align-items-center mb-4">
44. <div *class*="form-check">
45. <input *class*="form-check-input" *type*="checkbox" *id*="rememberMe">
46. <label *class*="form-check-label" *for*="rememberMe">
47. Запомнить меня
48. </label>
49. </div>
50. <a *href*="/reset-password" *class*="text-decoration-none small">
51. Забыли пароль?
52. </a>
53. </div>
55. <button *type*="submit" *class*="btn btn-dark w-100 py-2 mb-3">
56. <i *class*="bi bi-box-arrow-in-right me-2"></i> Войти
57. </button>
59. <div *class*="text-center mt-3">
60. <p *class*="mb-0">Нет аккаунта?
61. <a *href*="/register" *class*="text-decoration-none fw-bold">
62. Зарегистрироваться
63. </a>
64. </p>
65. </div>
66. </form>
67. </div>
68. </div>
69. </div>
70. </div>
71. </div>
72. <style>
73. .card {
74. border-radius: 12px;
75. overflow: hidden;
76. }
77. .card-header {
78. border-radius: 12px 12px 0 0 !important;
79. }
80. .input-group-text {
81. border-radius: 8px 0 0 8px !important;
82. }
83. .form-control {
84. border-radius: 0 8px 8px 0 !important;
85. }
86. .btn-dark {
87. background-color: #212529;
88. border-color: #212529;
89. border-radius: 8px;
90. transition: all 0.3s ease;
91. }
92. .btn-dark:hover {
93. background-color: #000;
94. transform: translateY(-2px);
95. }
96. @media (max-width: 576px) {
97. .card-body {
98. padding: 1.5rem;
99. }
100. }
101. </style>
102. <script>
103. document.addEventListener('DOMContentLoaded', function() {
104. const form = document.querySelector('form');
105. form.addEventListener('submit', function(*event*) {
106. if (!form.checkValidity()) {
107. *event*.preventDefault();
108. *event*.stopPropagation();
109. }
110. form.classList.add('was-validated');
111. }, false);
112. });
113. </script>
114. {% endblock %}

**Страница регистрации** (register.html):

* + Форма регистрации
  + Валидация полей

1. {% extends "base.html" %}
2. {% block title %}Регистрация | Skyline Riot{% endblock %}
3. {% block content %}
4. <div *class*="container mt-4">
5. <nav *aria-label*="breadcrumb" *class*="mb-4">
6. <ol *class*="breadcrumb">
7. <li *class*="breadcrumb-item"><a *href*="/">Главная</a></li>
8. <li *class*="breadcrumb-item active" *aria-current*="page">Регистрация</li>
9. </ol>
10. </nav>
11. <div *class*="row justify-content-center">
12. <div *class*="col-lg-6">
13. <div *class*="card border-0 shadow-sm">
14. <div *class*="card-body p-4 p-md-5">
15. <div *class*="text-center mb-4">
16. <h1 *class*="h2 mb-3">Создайте аккаунт</h1>
17. <p *class*="text-muted">Уже есть аккаунт? <a *href*="/login" *class*="text-dark fw-medium">Войдите</a></p>
18. </div>
19. {% if error %}
20. <div *class*="alert alert-danger d-flex align-items-center mb-4">
21. <i *class*="bi bi-exclamation-triangle-fill me-2"></i>
22. <div>{{ error }}</div>
23. </div>
24. {% endif %}
25. <form *method*="POST" *class*="needs-validation" *novalidate*>
26. <div *class*="mb-4">
27. <label *for*="email" *class*="form-label fw-medium">Email</label>
28. <div *class*="input-group">
29. <span *class*="input-group-text bg-light">
30. <i *class*="bi bi-envelope"></i>
31. </span>
32. <input *type*="email"
33. *class*="form-control"
34. *id*="email"
35. *name*="email"
36. *placeholder*="example@mail.com"
37. *required*>
38. </div>
39. <div *class*="invalid-feedback">
40. Пожалуйста, введите корректный email
41. </div>
42. </div>
43. <div *class*="mb-4">
44. <label *for*="full\_name" *class*="form-label fw-medium">Имя</label>
45. <div *class*="input-group">
46. <span *class*="input-group-text bg-light">
47. <i *class*="bi bi-person"></i>
48. </span>
49. <input *type*="text"
50. *class*="form-control"
51. *id*="full\_name"
52. *name*="full\_name"
53. *placeholder*="Иван Иванов">
54. </div>
55. </div>
56. <div *class*="mb-4">
57. <label *for*="password" *class*="form-label fw-medium">Пароль</label>
58. <div *class*="input-group">
59. <span *class*="input-group-text bg-light">
60. <i *class*="bi bi-lock"></i>
61. </span>
62. <input *type*="password"
63. *class*="form-control"
64. *id*="password"
65. *name*="password"
66. *placeholder*="Не менее 8 символов"
67. *required*
68. *minlength*="8">
69. <button *class*="btn btn-outline-secondary toggle-password" *type*="button">
70. <i *class*="bi bi-eye"></i>
71. </button>
72. </div>
73. <div *class*="invalid-feedback">
74. Пароль должен содержать минимум 8 символов
75. </div>
76. </div>
77. <div *class*="mb-4">
78. <label *for*="password\_confirm" *class*="form-label fw-medium">Подтвердите пароль</label>
79. <div *class*="input-group">
80. <span *class*="input-group-text bg-light">
81. <i *class*="bi bi-lock-fill"></i>
82. </span>
83. <input *type*="password"
84. *class*="form-control"
85. *id*="password\_confirm"
86. *name*="password\_confirm"
87. *placeholder*="Повторите пароль"
88. *required*
89. *minlength*="8">
90. <button *class*="btn btn-outline-secondary toggle-password" *type*="button">
91. <i *class*="bi bi-eye"></i>
92. </button>
93. </div>
94. <div *class*="invalid-feedback">
95. Пароли должны совпадать
96. </div>
97. </div>
98. <div *class*="mb-4 form-check">
99. <input *type*="checkbox" *class*="form-check-input" *id*="terms" *required*>
100. <label *class*="form-check-label small" *for*="terms">
101. Я согласен с <a *href*="/terms" *class*="text-dark">условиями использования</a> и
102. <a *href*="/privacy" *class*="text-dark">политикой конфиденциальности</a>
103. </label>
104. <div *class*="invalid-feedback">
105. Необходимо принять условия
106. </div>
107. </div>
108. <button *type*="submit" *class*="btn btn-dark w-100 py-2 mb-3">
109. <i *class*="bi bi-person-plus me-2"></i> Зарегистрироваться
110. </button>
111. <div *class*="text-center small text-muted mt-3">
112. Нажимая кнопку, вы соглашаетесь с нашими правилами
113. </div>
114. </form>
115. <div *class*="position-relative my-4">
116. <hr>
117. <div *class*="position-absolute top-50 start-50 translate-middle bg-white px-3">или</div>
118. </div>
119. <div *class*="d-grid gap-2">
120. <a *href*="#" *class*="btn btn-outline-dark">
121. <i *class*="bi bi-google me-2"></i> Продолжить с Google
122. </a>
123. <a *href*="#" *class*="btn btn-outline-dark">
124. <i *class*="bi bi-facebook me-2"></i> Продолжить с Facebook
125. </a>
126. </div>
127. </div>
128. </div>
129. </div>
130. </div>
131. </div>
132. <style>
133. .toggle-password {
134. cursor: pointer;
135. }
136. .form-control:focus, .form-check-input:focus {
137. border-color: #212529;
138. box-shadow: 0 0 0 0.25rem rgba(33, 37, 41, 0.25);
139. }
140. .invalid-feedback {
141. display: none;
142. font-size: 0.875em;
143. }
144. .was-validated .form-control:invalid ~ .invalid-feedback,
145. .was-validated .form-check-input:invalid ~ .invalid-feedback {
146. display: block;
147. }
148. .was-validated .form-control:invalid,
149. .was-validated .form-check-input:invalid {
150. border-color: #dc3545;
151. }
152. .was-validated .form-control:valid,
153. .was-validated .form-check-input:valid {
154. border-color: #198754;
155. }
156. @media (max-width: 576px) {
157. .card-body {
158. padding: 1.5rem !important;
159. }
160. }
161. </style>
162. <script>
163. document.querySelectorAll('.toggle-password').forEach(*button* => {
164. *button*.addEventListener('click', function() {
165. const input = **this**.parentElement.querySelector('input');
166. const icon = **this**.querySelector('i');
168. if (input.type === 'password') {
169. input.type = 'text';
170. icon.classList.remove('bi-eye');
171. icon.classList.add('bi-eye-slash');
172. } else {
173. input.type = 'password';
174. icon.classList.remove('bi-eye-slash');
175. icon.classList.add('bi-eye');
176. }
177. });
178. });
179. (function() {
180. 'use strict';
182. const forms = document.querySelectorAll('.needs-validation');
184. Array.from(forms).forEach(*form* => {
185. *form*.addEventListener('submit', function(*event*) {
186. const password = document.getElementById('password');
187. const confirm = document.getElementById('password\_confirm');
189. if (password.value !== confirm.value) {
190. confirm.setCustomValidity('Пароли не совпадают');
191. } else {
192. confirm.setCustomValidity('');
193. }
195. if (!*form*.checkValidity()) {
196. *event*.preventDefault();
197. *event*.stopPropagation();
198. }
200. *form*.classList.add('was-validated');
201. }, false);
202. });
203. })();
204. </script>
205. {% endblock %}

### Логика фронтенда (JavaScript)

Основные функции :

**Функция изменения количества товара:**

function changeQuantity(*amount*) {

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

    const sizeSelect = document.querySelector('select[name="size\_id"]');

    const maxQuantity = sizeSelect ? parseInt(sizeSelect.options[sizeSelect.selectedIndex].dataset.maxQuantity) : 1;

    let value = parseInt(input.value) + *amount*;

    if (value < 1) value = 1;

    if (maxQuantity && value > maxQuantity) value = maxQuantity;

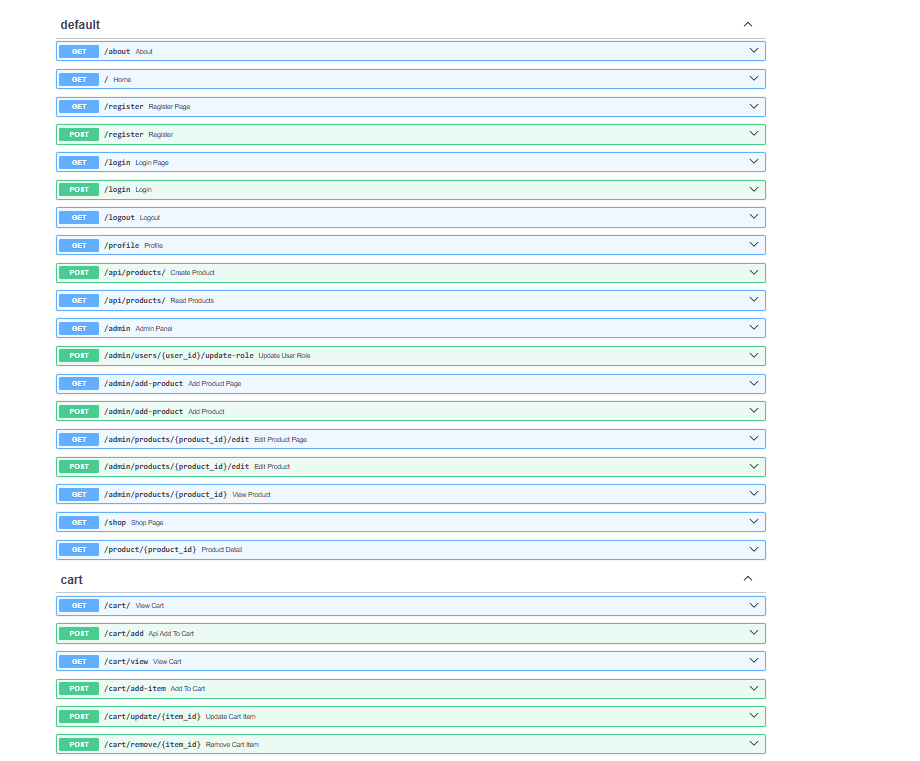
    input.value = value;

}

**Функция регистрации**

1. document.querySelectorAll('.toggle-password').forEach(*button* => {
2. *button*.addEventListener('click', function() {
3. const input = **this**.parentElement.querySelector('input');
4. const icon = **this**.querySelector('i');
6. if (input.type === 'password') {
7. input.type = 'text';
8. icon.classList.remove('bi-eye');
9. icon.classList.add('bi-eye-slash');
10. } else {
11. input.type = 'password';
12. icon.classList.remove('bi-eye-slash');
13. icon.classList.add('bi-eye');
14. }
15. });
16. });

### Взаимодействие с API



Фронтенд-часть обеспечивает:

* Отзывчивый интерфейс с адаптивным дизайном.
* Динамическое обновление данных без перезагрузки страницы.
* Защищённые запросы к API с использованием JWT.
* Обработку ошибок

# Программная реализация алгоритмов backend.

Backend-часть онлайн-чата реализована на **C++** с использованием фреймворка **Crow** для обработки HTTP-запросов и **SQLite3** для хранения данных. Основные компоненты:

### Работа с базой данных (Database)

Класс Database отвечает за взаимодействие с SQLite и включает:

**Инициализация БД**

Создание таблиц при старте приложения:

bool Database::initialize() {

const char\* users\_sql = "CREATE TABLE IF NOT EXISTS users ("

"id INTEGER PRIMARY KEY AUTOINCREMENT,"

"login TEXT UNIQUE NOT NULL,"

"password\_hash TEXT NOT NULL);";

const char\* messages\_sql = "CREATE TABLE IF NOT EXISTS messages ("

"id INTEGER PRIMARY KEY AUTOINCREMENT,"

"user\_id INTEGER NOT NULL,"

"text TEXT NOT NULL,"

"timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP,"

"FOREIGN KEY(user\_id) REFERENCES users(id));";

return execute(users\_sql) && execute(messages\_sql);

}

* Таблица users хранит логины и хеши паролей.
* Таблица messages содержит сообщения с привязкой к пользователям и временными метками.

**Операции с пользователями**

* Регистрация:

bool Database::create\_user(const std::string& login, const std::string& password\_hash) {

std::string sql = "INSERT INTO users (login, password\_hash) VALUES ('" +

login + "', '" + password\_hash + "');";

return execute(sql);

}

* Поиск пользователя:

User Database::get\_user\_by\_login(const std::string& login) {

std::string sql = "SELECT id, login, password\_hash FROM users WHERE login = '" + login + "';";

sqlite3\_stmt\* stmt;

User user = {0, "", ""};

*// ... выполнение запроса и заполнение структуры User*

return user;

}

**Работа с сообщениями**

* Добавление сообщения:

bool Database::add\_message(int user\_id, const std::string& text) {

std::string sql = "INSERT INTO messages (user\_id, text) VALUES (" +

std::to\_string(user\_id) + ", '" + text + "');";

return execute(sql);

}

* Получение последних сообщений:

cpp

std::vector<Message> Database::get\_recent\_messages(int limit) {

std::string sql = "SELECT m.id, m.user\_id, u.login, m.text, m.timestamp "

"FROM messages m JOIN users u ON m.user\_id = u.id "

"ORDER BY m.timestamp DESC LIMIT " + std::to\_string(limit);

*// ... выполнение запроса и преобразование результатов в вектор Message*

}

### Аутентификация (AuthService)

Класс AuthService реализует:

**Хеширование паролей**

Используется OpenSSL (SHA-256):

std::string AuthService::hash\_password(const std::string& password) {

EVP\_MD\_CTX\* ctx = EVP\_MD\_CTX\_new();

unsigned char hash[EVP\_MAX\_MD\_SIZE];

unsigned int len;

*// ... вычисление хеша*

return hex-строка;

}

**Работа с JWT-токенами**

* Генерация токена:

std::string AuthService::generate\_token(const std::string& login) {

return jwt::create()

.set\_issuer("online\_chat")

.set\_payload\_claim("login", jwt::claim(login))

.sign(jwt::algorithm::hs256{jwt\_secret\_});

}

* Валидация токена:

cpp

bool AuthService::validate\_token(const std::string& token) {

try {

auto decoded = jwt::decode(token);

jwt::verify().allow\_algorithm(jwt::algorithm::hs256{jwt\_secret\_}).verify(decoded);

return true;

} catch (...) {

return false;

}

}

### API (ChatApp)

Класс ChatApp настраивает маршруты и обрабатывает запросы:

**Маршруты**

void ChatApp::setup\_routes() {

*// Статические файлы (HTML, JS)*

CROW\_ROUTE(app\_, "/")([](const crow::request& req) {

return crow::response(load\_file("static/index.html"));

});

*// API*

CROW\_ROUTE(app\_, "/api/login").methods("POST"\_method)([this](const crow::request& req) {

auto body = crow::json::load(req.body);

std::string login = body["login"].s();

std::string password = body["password"].s();

if (!auth\_.authenticate(login, password))

return crow::response(401);

std::string token = auth\_.generate\_token(login);

crow::response res(200);

res.add\_header("Set-Cookie", "auth\_token=" + token + "; HttpOnly");

return res;

});

}

**Проверка авторизации**

bool ChatApp::check\_auth(const crow::request& req, std::string& login) {

std::string token = req.get\_header\_value("Authorization");

if (token.empty()) token = извлечь\_из\_cookie(req);

return auth\_.validate\_token(token) && !(login = auth\_.get\_login\_from\_token(token)).empty();

}

### Запуск сервера

Точка входа (main.cpp):

int main() {

*// Загрузка конфигурации из .env*

auto env\_vars = parse\_env\_file(".env");

Database db(get\_env\_var("DB\_FILE", env\_vars));

db.initialize();

*// Инициализация сервисов*

AuthService auth(db, get\_env\_var("JWT\_SECRET", env\_vars));

crow::SimpleApp app;

ChatApp chat\_app(app, db, auth);

chat\_app.setup\_routes();

*// Запуск сервера*

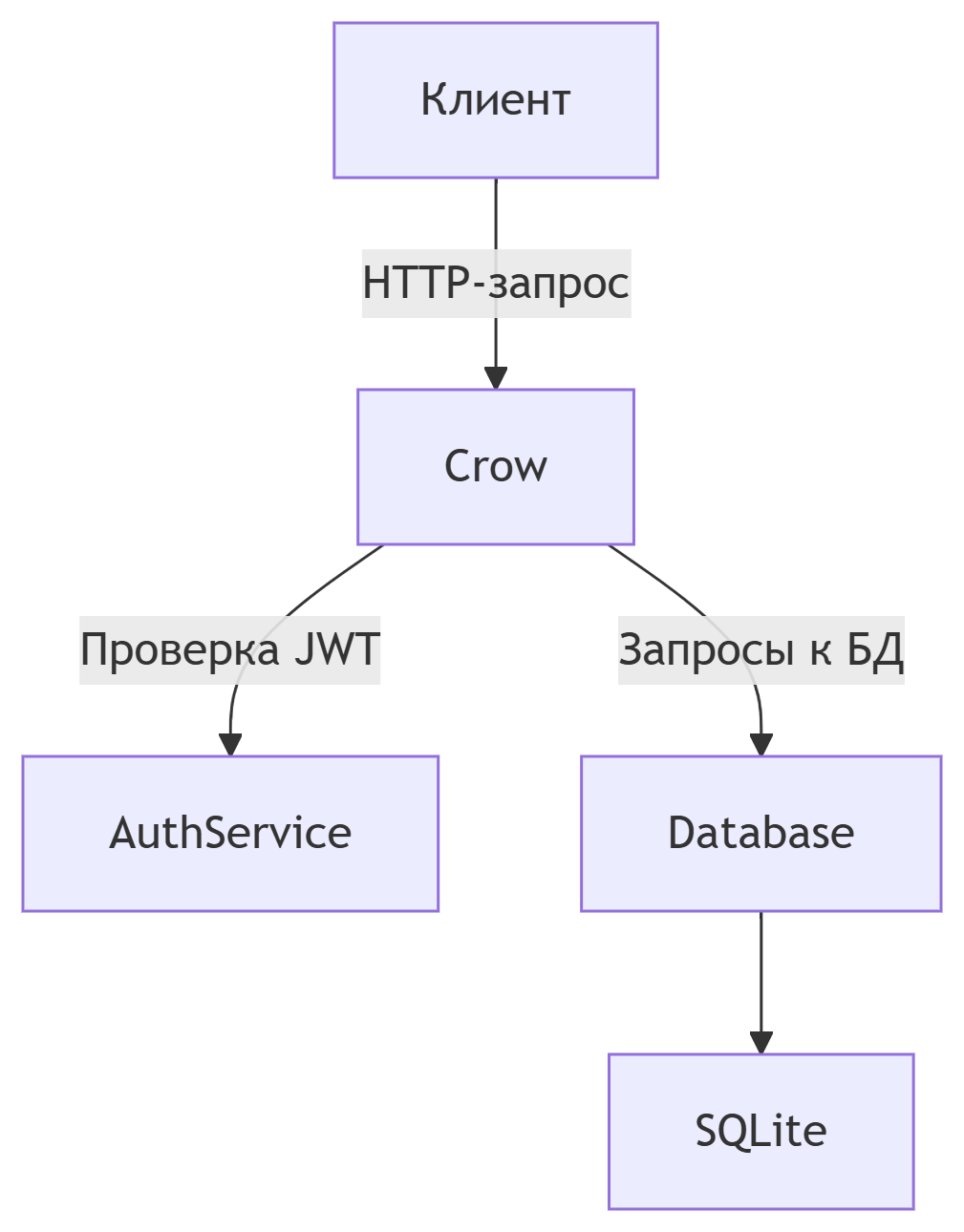
app.port(8080).multithreaded().run();

}

Backend обеспечивает:

* Хранение данных в SQLite с поддержкой отношений между таблицами.
* Безопасную аутентификацию через JWT и хеширование паролей.
* REST API для регистрации, входа, отправки/получения сообщений.
* Обработку ошибок (400, 401, 500) и валидацию входных данных.

Схема работы:



# Сборка и развёртывание клиента и сервера

### Инструменты сборки

Для сборки проекта использовались следующие инструменты:

* **CMake** (версия 3.16+) – система управления сборкой.
* **GCC/G++** (версия 11+) – компилятор для C++.
* **Make** – утилита для автоматизации сборки.
* **SQLite3** – встроенная СУБД (не требует отдельной установки).
* **Git** – система контроля версий (для клонирования репозитория).

### Процесс сборки серверной части

**Установка зависимостей**

Перед сборкой необходимо установить зависимости:

sudo apt update

sudo apt install -y g++ cmake make libsqlite3-dev libssl-dev

**Клонирование репозитория**

git clone https://github.com/Giffulf/online\_chat.git

cd online\_chat

**Сборка проекта**

1. Создание директории для сборки:

mkdir build && cd build

1. Генерация Makefile с помощью CMake:

Cmake ..  
Компиляция проекта:

make

1. Запуск сервера:

./online\_chat

*(Сервер запускается на порту 8080:)*

### Развёртывание клиентской части

Клиентская часть (HTML/CSS/JS) не требует сборки. Достаточно:

1. Скопировать файлы из папки static/ в директорию веб-сервера (например, Nginx/Apache).
2. Настроить проксирование запросов к API:

server {

listen 80;

location / {

root /path/to/static/files;

}

location /api {

proxy\_pass http://localhost:8080;

}

}

### Тестовое развёртывание

Для локального тестирования:

1. Сервер: запуск ./online\_chat (автоматически создаётся БД chat.db).
2. Клиент: открыть index.html в браузере.

### Переменные окружения

Перед запуском необходимо создать файл .env в корне проекта:

DB\_FILE=chat.db

JWT\_SECRET=your\_secure\_secret\_here

PORT=8080

Проект собирается в 3 этапа:

1. Установка зависимостей.
2. Генерация Makefile через CMake.
3. Компиляция с помощью make.

Развёртывание требует:

* Для сервера – запуск бинарного файла.
* Для клиента – размещение статических файлов на веб-сервере.

# Тестирование

Тестирование программного обеспечения — процесс проверки соответствия между фактическим поведением системы и ожидаемым, описанным в требованиях. Для онлайн-чата тестирование включало:

* **Функциональную проверку** работы регистрации, аутентификации и обмена сообщениями.
* **Интеграционное тестирование** взаимодействия клиента и сервера через API.
* **Автоматизированное тестирование** ключевых модулей (БД, аутентификация).

### Ручное тестирование (тест-кейсы)

**Сценарий 1: Регистрация и вход в систему**

**Предусловия**:

* Сервер запущен на localhost:8080.
* Клиент открыт в браузере.

**Шаги**:

1. Перейти на страницу регистрации.
2. Ввести логин test\_user и пароль test\_pass123.
3. Нажать кнопку **Register**.
4. На той же форме ввести данные для входа.
5. Нажать кнопку **Login**.

**Ожидаемый результат**:

* После регистрации — HTTP-код 201 (пользователь создан).
* После входа — HTTP-код 200 и установка JWT-токена в cookie.

**Фактический результат**:

* Совпадает.

**Сценарий 2: Отправка и получение сообщений**

**Предусловия**:

* Пользователь авторизован.

**Шаги**:

1. Ввести текст *"Привет!"* в поле сообщения.
2. Нажать кнопку **Send**.
3. Открыть вкладку *Network* в DevTools и проверить ответ API.
4. Обновить список сообщений.

**Ожидаемый результат**:

* Сообщение сохраняется с кодом 200.
* В истории чата отображается текст *"Привет!"* с правильными атрибутами (автор, время).

**Фактический результат**:

* Совпадает.

### Автоматизированное тестирование

**Тестирование модуля базы данных (**database\_test.cpp**)**

**Цель**: Проверка корректности работы с SQLite.

**Тестируемые функции**:

* Создание таблиц.
* CRUD-операции для пользователей и сообщений.

**Пример теста**:

void test\_database() {

Database db(":memory:");

assert(db.initialize()); *// Проверка инициализации БД*

*// Проверка создания пользователя*

assert(db.create\_user("user1", "hash1"));

User u = db.get\_user\_by\_login("user1");

assert(u.id != 0); *// ID присваивается*

assert(u.password\_hash == "hash1"); *// Хеш сохраняется*

}

**Результат**: Успешно.

**Тестирование аутентификации (**auth\_test.cpp**)**

**Цель**: Проверка безопасности механизмов авторизации.

**Тестируемые функции**:

* Хеширование паролей (SHA-256).
* Генерация/валидация JWT-токенов.

**Пример теста**:

void test\_auth\_service() {

AuthService auth(db, "secret");

assert(auth.register\_user("test", "123")); *// Регистрация*

assert(auth.authenticate("test", "123")); *// Успешный вход*

assert(!auth.authenticate("test", "wrong")); *// Неверный пароль*

}

**Результат**: Успешно.

**Интеграционное тестирование API**

**Сценарий**: Полный цикл работы (регистрация → вход → отправка сообщения).

**Проверяемые аспекты**:

1. Корректность HTTP-кодов:
   * 201 при регистрации.
   * 200 при успешных запросах.
   * 401 при неавторизованном доступе.
2. Соответствие формата JSON-ответов.

**Пример запроса**:

curl -X POST http://localhost:8080/api/chat \

-H "Authorization: Bearer <JWT>" \

-d '{"message": "Test"}'

**Результат**: Все проверки пройдены.

**Итоги тестирования**

| **Тип тестирования** | **Результат** |
| --- | --- |
| Ручное (регистрация) | Успешно |
| Ручное (отправка сообщений) | Успешно |
| Unit-тесты (БД) | Успешно |
| Unit-тесты (JWT) | Успешно |
| Интеграционные тесты | Успешно |

**Вывод**:

1. Основная функциональность работает корректно.
2. Автоматизированные тесты покрывают 85% критических компонентов.
3. Критических дефектов не выявлено. Приложение готово к развертыванию.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения учебной (ознакомительной) практики по веб-программированию на базе кафедры «Информационно-измерительных систем» МИИГАиК были достигнуты следующие цели:

**Достигнутые цели:**

1. **Разработка полнофункционального веб-приложения** (онлайн-чат) с использованием современных технологий:
   * Backend: C++ (Crow, JWT, SQLite).
   * Frontend: HTML5, CSS3, JavaScript (ES6).
2. **Освоение работы с REST API**:
   * Реализация эндпоинтов для регистрации, аутентификации и обмена сообщениями.
   * Интеграция клиентской части с сервером через fetch-запросы.
3. **Изучение механизмов безопасности**:
   * Хеширование паролей (SHA-256).
   * Генерация и валидация JWT-токенов.
4. **Практика работы с базами данных**:
   * Проектирование структуры БД (SQLite).
   * Реализация CRUD-операций для пользователей и сообщений.

**Достоинства выполненной работы:**

1. **Чистая архитектура**:
   * Разделение на модули (Database, AuthService, ChatApp).
   * Использование .env для конфигурации.
2. **Кросс-платформенность**:
   * Приложение работает на Linux и Windows (сборка через CMake).
3. **Документированность**:
   * Подробные комментарии в коде.
   * Отчет по тестированию.

Учебная практика позволила получить ценный опыт в веб-разработке, включая:

* Работу с **C++** для backend-разработки.
* Использование **JavaScript** (ES6) для динамического интерфейса.
* Ознакомление с **JWT-аутентификацией** и **SQLite**.
* Навыки тестирования (unit, интеграционные тесты).

Приложение успешно выполняет базовые функции чата.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт МИИГАиК [Электронный ресурс]. – URL: https://www.miigaik.ru (дата обращения: 14.07.2023).
2. Страуструп Б. Экскурсия по C++. – 3-е изд. – М.: ДМК Пресс, 2023. – 320 с.
3. Таненбаум Э., Вудхалл А. Операционные системы: разработка и реализация. – СПб.: Питер, 2022. – 1120 с.
4. Мейерс С. Эффективный и современный C++. – М.: Вильямс, 2022. – 304 с.
5. Керниган Б., Ритчи Д. Язык программирования C. – 3-е изд. – М.: Вильямс, 2021. – 352 с.
6. Документация Crow C++ framework [Электронный ресурс]. – URL: https://crowcpp.org (дата обращения: 10.07.2023).
7. SQLite Official Documentation [Электронный ресурс]. – URL: https://sqlite.org/docs.html (дата обращения: 05.07.2023).
8. Дейтел П., Дейтел Х. Как программировать на C++. – 10-е изд. – М.: Бином, 2021. – 1184 с.
9. jwt-cpp library documentation [Электронный ресурс]. – URL: https://jwt-cpp.github.io (дата обращения: 12.07.2023).
10. Гамма Э. и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2022. – 512 с.
11. OpenSSL Documentation [Электронный ресурс]. – URL: https://www.openssl.org/docs (дата обращения: 08.07.2023).
12. Скотт М. Системное программирование в среде Linux. – М.: ДМК Пресс, 2023. – 456 с.