|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа №2 | | |
| по дисциплине «Разработка web-приложений и распределенных информационных систем» | | |
| **Создание и работа с Docker-Compose** | | |
|  | | |
|  | Факультет | ПМИ |
| Группа | ПМ-25 |
| Студенты: | Комилов хайруллохон |
|  |  |
|  |  |
|  |  |
| Преподаватель: | Цыгулин Алексей Александрович |
|  |  |
| Новосибирск, 2025 | | |

1. **Цель работы**

Приобретение навыков работы с комплексом контейнеров - Docker Compose, ознакомиться с созданием и компиляцией docker-compose.yml файла, содержащем контейнеры с модернизированной Web-страницей из предыдущей работы, содержащей раздел с комментариями , бекендом приложения и базой данных для хранения комментариев пользователей.

1. **Постановка задачи**

Создать комплекс контейнеров (Docker Compose) и разместить их в репозитории Dockerhub (<https://hub.docker.com/>). При загрузке комплекса должен запуститься сервер, предоставляющий доступ к странице “О себе с комментариями”, при этом комментарии должны сохраняться между запусками программы. В отчёте обязательно указать строку запуска docker-compose.yml файла и прикрепить содержимое этого файла.

1. **Структура программы**

my-portfolio-compose/

├── frontend/

│ ├── public/index.html

│ ├── src/

│ │ ├── styles/

│ │ │ ├── main.css

│ │ │ └── components/

│ │ │ ├── header.css

│ │ │ ├── footer.css

│ │ │ └── comments.css

│ │ └── js/

│ │ ├── main.js

│ │ └── comments.js

│ ├── Dockerfile

│ └── nginx.conf

├── backend/

│ ├── app.js

│ ├── package.json

│ └── Dockerfile

├── database/

│ └── init.sql

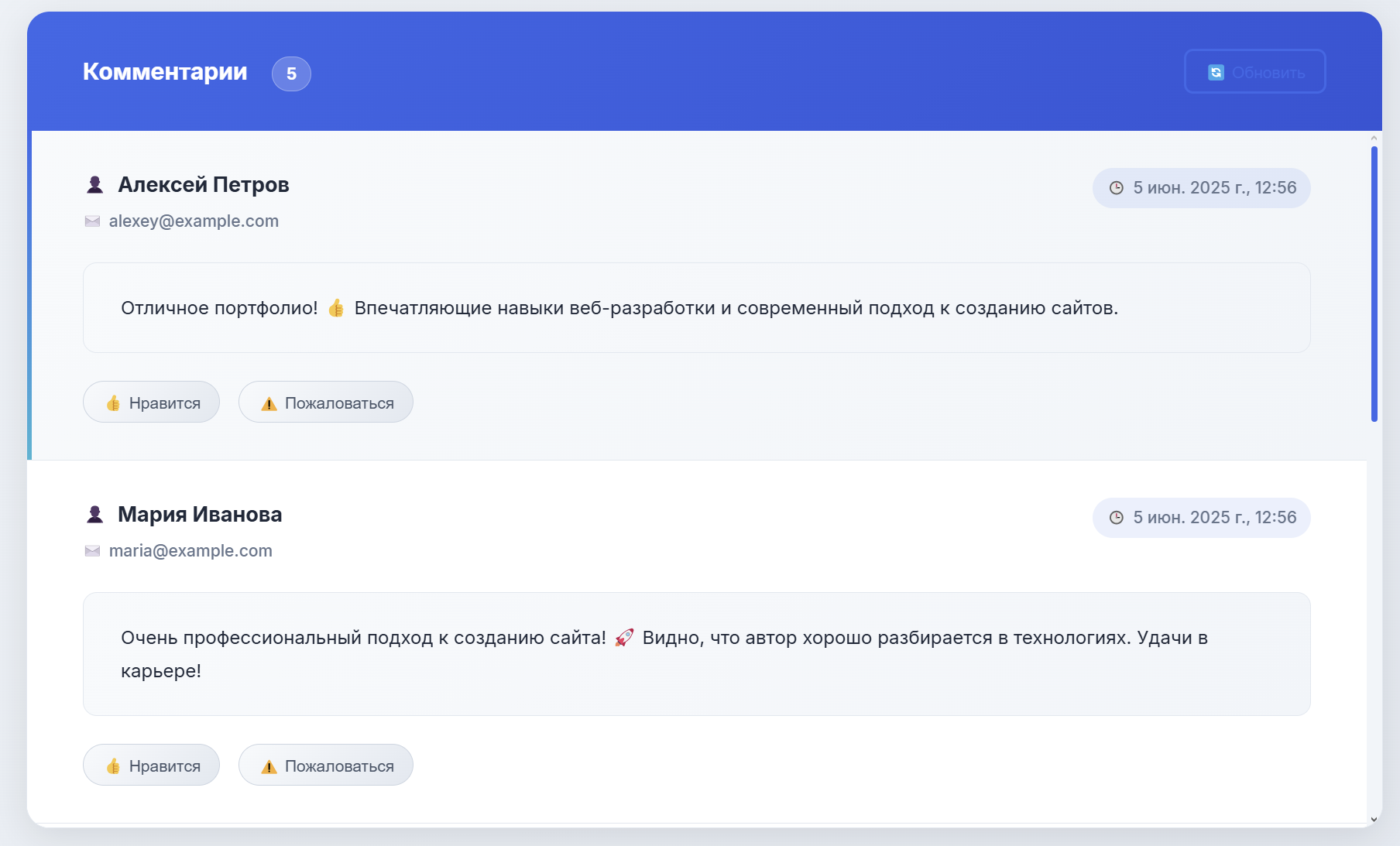
├── docker-compose.yml

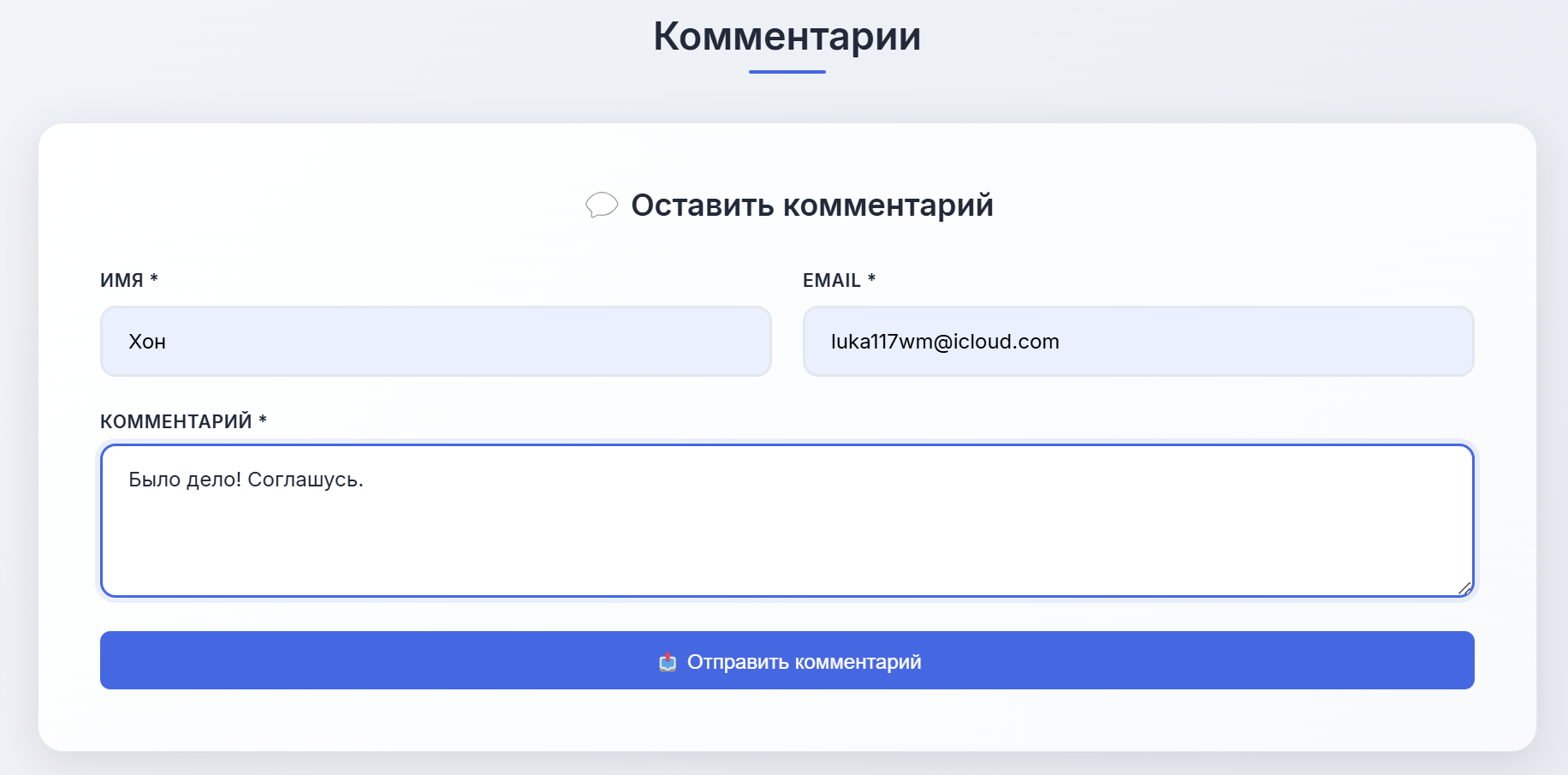
└── README.md

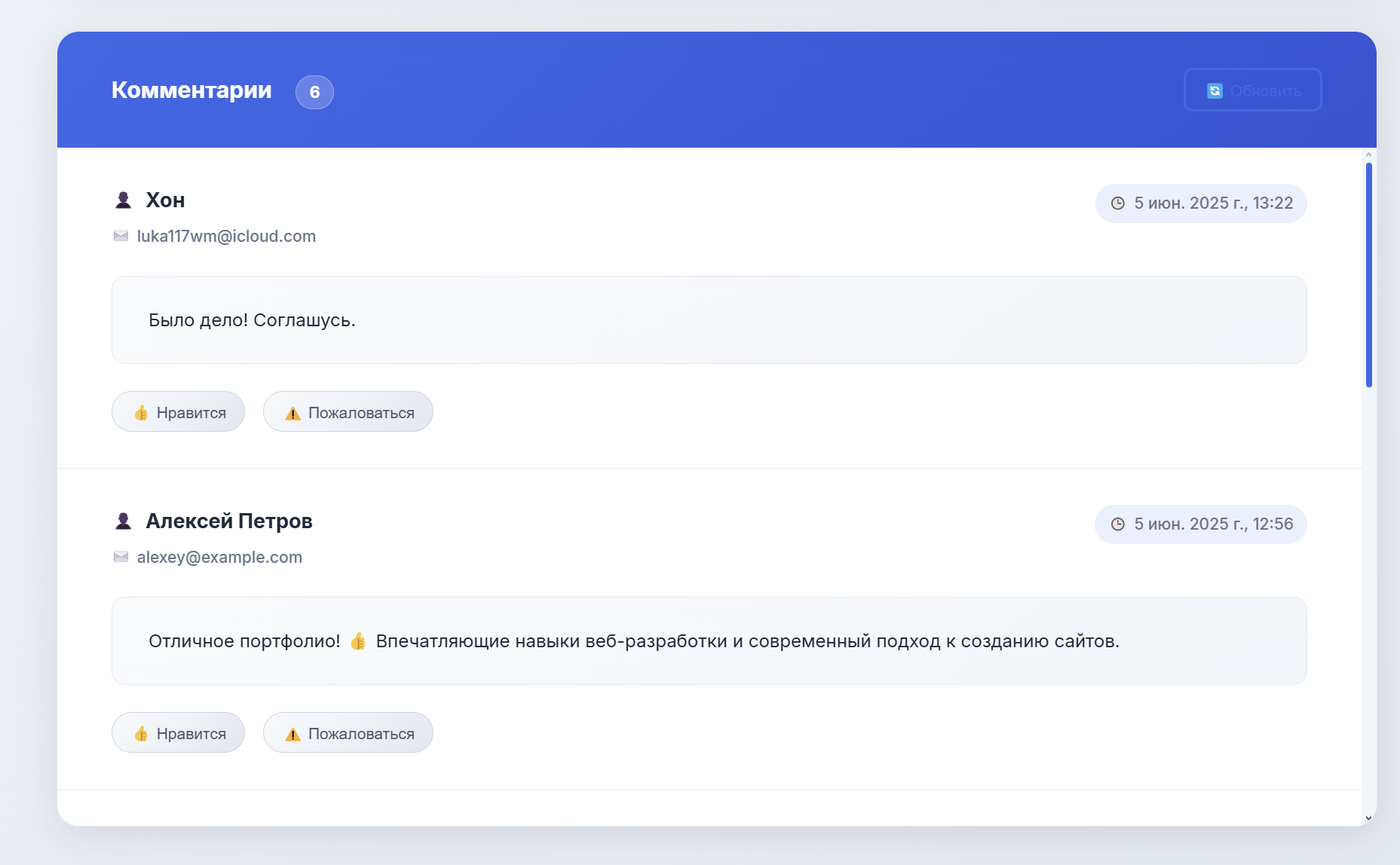
1. **Реализация WEB-страницы**

|  |
| --- |
| **Frontend** |
| index.html |
| <!DOCTYPE html>  <html lang="ru">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <meta name="description" content="Личное портфолио веб-разработчика">      <meta name="keywords" content="веб-разработка, frontend, HTML, CSS, JavaScript">      <meta name="author" content="Your Name">        <title>Портфолио - Веб-разработчик</title>        <!-- Подключение стилей -->      <link rel="stylesheet" href="../src/styles/main.css">      <link rel="stylesheet" href="../src/styles/components/header.css">      <link rel="stylesheet" href="../src/styles/components/about.css">      <link rel="stylesheet" href="../src/styles/components/footer.css">      <link rel="stylesheet" href="../src/styles/components/comments.css">        <!-- Favicon -->      <link rel="icon" href="data:image/svg+xml,<svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 100 100'><text y='.9em' font-size='90'>👨‍💻</text></svg>">  </head>  <body>      <!-- Header -->      <header class="header" id="header">          <nav class="nav">              <div class="nav\_\_brand">                  <h2>MyPortfolio</h2>              </div>              <ul class="nav\_\_menu" id="nav-menu">                  <li class="nav\_\_item">                      <a href="#about" class="nav\_\_link">О себе</a>                  </li>                  <li class="nav\_\_item">                      <a href="#skills" class="nav\_\_link">Навыки</a>                  </li>                  <li class="nav\_\_item">                      <a href="#contact" class="nav\_\_link">Контакты</a>                  </li>              </ul>              <button class="nav\_\_toggle" id="nav-toggle" aria-label="Toggle navigation">                  <span></span>                  <span></span>                  <span></span>              </button>          </nav>      </header>      <!-- Main Content -->      <main class="main">          <!-- Hero Section -->          <section class="hero" id="hero">              <div class="hero\_\_container">                  <div class="hero\_\_content">                      <h1 class="hero\_\_title">                          Привет! Я <span class="hero\_\_name">Junior Web Developer</span>                      </h1>                      <p class="hero\_\_description">                          Изучаю современные технологии веб-разработки и создаю интересные проекты                      </p>                      <div class="hero\_\_actions">                          <a href="#about" class="btn btn--primary">Узнать больше</a>                          <a href="#contact" class="btn btn--secondary">Связаться</a>                      </div>                  </div>              </div>          </section>          <!-- About Section -->          <section class="about" id="about">              <div class="container">                  <h2 class="section\_\_title">О себе</h2>                  <div class="about\_\_content">                      <div class="about\_\_text">                          <p class="about\_\_description">                              Я начинающий веб-разработчик, который увлечен созданием современных                              и функциональных веб-приложений. Постоянно изучаю новые технологии                              и стремлюсь к написанию чистого, поддерживаемого кода.                          </p>                          <p class="about\_\_description">                              В данный момент изучаю контейнеризацию с Docker, что позволяет                              создавать масштабируемые и портативные приложения.                          </p>                      </div>                  </div>              </div>          </section>          <!-- Skills Section -->          <section class="skills" id="skills">              <div class="container">                  <h2 class="section\_\_title">Технические навыки</h2>                  <div class="skills\_\_grid">                      <div class="skill-card">                          <div class="skill-card\_\_icon">🌐</div>                          <h3 class="skill-card\_\_title">Frontend</h3>                          <ul class="skill-card\_\_list">                              <li>HTML5 & CSS3</li>                              <li>JavaScript (ES6+)</li>                              <li>Responsive Design</li>                              <li>CSS Grid & Flexbox</li>                          </ul>                      </div>                        <div class="skill-card">                          <div class="skill-card\_\_icon">⚙️</div>                          <h3 class="skill-card\_\_title">Инструменты</h3>                          <ul class="skill-card\_\_list">                              <li>Git & GitHub</li>                              <li>Docker</li>                              <li>VS Code</li>                              <li>NPM/Yarn</li>                          </ul>                      </div>                        <div class="skill-card">                          <div class="skill-card\_\_icon">📚</div>                          <h3 class="skill-card\_\_title">Изучаю</h3>                          <ul class="skill-card\_\_list">                              <li>React.js</li>                              <li>Node.js</li>                              <li>TypeScript</li>                              <li>PostgreSQL</li>                          </ul>                      </div>                  </div>              </div>          </section>          <!-- Contact Section -->          <section class="contact" id="contact">              <div class="container">                  <h2 class="section\_\_title">Контакты</h2>                  <div class="contact\_\_content">                      <div class="contact\_\_info">                          <div class="contact\_\_item">                              <span class="contact\_\_icon">📧</span>                              <div class="contact\_\_details">                                  <h4>Email</h4>                                  <a href="mailto:your.email@example.com">your.email@example.com</a>                              </div>                          </div>                            <div class="contact\_\_item">                              <span class="contact\_\_icon">🌐</span>                              <div class="contact\_\_details">                                  <h4>GitHub</h4>                                  <a href="https://github.com/yourusername" target="\_blank">github.com/yourusername</a>                              </div>                          </div>                            <div class="contact\_\_item">                              <span class="contact\_\_icon">💼</span>                              <div class="contact\_\_details">                                  <h4>LinkedIn</h4>                                  <a href="https://linkedin.com/in/yourusername" target="\_blank">linkedin.com/in/yourusername</a>                              </div>                          </div>                      </div>                  </div>              </div>          </section>          <!-- Comments Section -->          <section class="comments" id="comments">              <div class="container">                  <h2 class="section\_\_title">Комментарии</h2>                    <!-- Add Comment Form -->                  <div class="comment-form">                      <h3>💬 Оставить комментарий</h3>                      <form id="commentForm" class="comment-form\_\_form">                          <div class="form-row">                              <div class="form-group">                                  <label for="name">Имя \*</label>                                  <input type="text" id="name" name="name" required placeholder="Введите ваше имя">                              </div>                                <div class="form-group">                                  <label for="email">Email \*</label>                                  <input type="email" id="email" name="email" required placeholder="your@email.com">                              </div>                          </div>                            <div class="form-group">                              <label for="comment">Комментарий \*</label>                              <textarea id="comment" name="comment" rows="4" required placeholder="Поделитесь своими мыслями..."></textarea>                          </div>                            <button type="submit" class="btn btn--primary">                              <span class="btn\_\_text">📤 Отправить комментарий</span>                              <span class="btn\_\_spinner" style="display: none;">⏳</span>                          </button>                      </form>                  </div>                  <!-- Comments List -->                  <div class="comments-list">                      <div class="comments-header">                          <h3>Комментарии <span id="commentsCount" class="comments-count">0</span></h3>                          <button id="refreshComments" class="btn btn--secondary btn--small">🔄 Обновить</button>                      </div>                        <div id="commentsList" class="comments-items">                          <!-- Comments will be loaded here -->                      </div>                        <div id="commentsLoading" class="loading" style="display: none;">                          <p>Загрузка комментариев...</p>                      </div>                        <div id="commentsError" class="error-message" style="display: none;">                          <p>Ошибка загрузки комментариев. <button onclick="loadComments()">Повторить</button></p>                      </div>                  </div>              </div>          </section>      </main>      <!-- Footer -->      <footer class="footer">          <div class="container">              <p class="footer\_\_text">                  &copy; 2025 MyPortfolio. Сделано с ❤️ и Docker              </p>          </div>      </footer>      <!-- JavaScript -->      <script src="../src/js/main.js"></script>      <script src="../src/js/comments.js"></script>  </body>  </html> |
| main.js – без изменений |
| comment.js |
| // ===== COMMENTS API CLASS =====  class CommentsAPI {      constructor() {          this.baseURL = window.location.hostname === 'localhost'              ? 'http://localhost:3000/api'              : '/api';      }      async request(endpoint, options = {}) {          try {              const response = await fetch(`${this.baseURL}${endpoint}`, {                  headers: {                      'Content-Type': 'application/json; charset=utf-8',                      'Accept': 'application/json',                      ...options.headers                  },                  ...options              });              if (!response.ok) {                  const errorData = await response.json().catch(() => ({}));                  throw new Error(errorData.message || `HTTP ${response.status}: ${response.statusText}`);              }              return await response.json();          } catch (error) {              console.error('API Request failed:', error);              throw error;          }      }      async getComments() {          return this.request('/comments');      }      async addComment(commentData) {          return this.request('/comments', {              method: 'POST',              body: JSON.stringify(commentData)          });      }      async deleteComment(id) {          return this.request(`/comments/${id}`, {              method: 'DELETE'          });      }      async getCommentsCount() {          return this.request('/comments/count');      }      async healthCheck() {          return this.request('/health');      }  }  // ===== COMMENTS MANAGER CLASS =====  class CommentsManager {      constructor() {          this.api = new CommentsAPI();          this.form = document.getElementById('commentForm');          this.commentsList = document.getElementById('commentsList');          this.commentsCount = document.getElementById('commentsCount');          this.loadingElement = document.getElementById('commentsLoading');          this.errorElement = document.getElementById('commentsError');          this.refreshButton = document.getElementById('refreshComments');            this.init();      }      init() {          this.setupEventListeners();          this.loadComments();          this.updateCommentsCount();            // Auto-refresh every 30 seconds          setInterval(() => this.loadComments(true), 30000);      }      setupEventListeners() {          // Form submission          this.form?.addEventListener('submit', (e) => this.handleSubmit(e));            // Refresh button          this.refreshButton?.addEventListener('click', () => this.loadComments());            // Real-time validation          const inputs = this.form?.querySelectorAll('input, textarea');          inputs?.forEach(input => {              input.addEventListener('blur', () => this.validateField(input));              input.addEventListener('input', () => this.clearValidationMessage(input));          });      }      async handleSubmit(e) {          e.preventDefault();            const formData = new FormData(this.form);          const commentData = {              name: formData.get('name')?.trim(),              email: formData.get('email')?.trim(),              comment: formData.get('comment')?.trim()          };          // Validate form          if (!this.validateForm(commentData)) {              return;          }          const submitButton = this.form.querySelector('button[type="submit"]');          const buttonText = submitButton.querySelector('.btn\_\_text');          const buttonSpinner = submitButton.querySelector('.btn\_\_spinner');          try {              // Show loading state              submitButton.disabled = true;              buttonText.style.display = 'none';              buttonSpinner.style.display = 'inline-block';              const response = await this.api.addComment(commentData);              if (response.success) {                  this.showMessage('Комментарий успешно добавлен!', 'success');                  this.form.reset();                  this.loadComments();                  this.updateCommentsCount();              } else {                  this.showMessage(response.message || 'Ошибка при добавлении комментария', 'error');              }          } catch (error) {              console.error('Error adding comment:', error);              this.showMessage('Ошибка соединения с сервером', 'error');          } finally {              // Hide loading state              submitButton.disabled = false;              buttonText.style.display = 'inline';              buttonSpinner.style.display = 'none';          }      }      validateForm(data) {          let isValid = true;          // Name validation          if (!data.name || data.name.length < 2) {              this.showFieldError('name', 'Имя должно содержать минимум 2 символа');              isValid = false;          }          // Email validation          const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;          if (!data.email || !emailRegex.test(data.email)) {              this.showFieldError('email', 'Введите корректный email адрес');              isValid = false;          }          // Comment validation          if (!data.comment || data.comment.length < 10) {              this.showFieldError('comment', 'Комментарий должен содержать минимум 10 символов');              isValid = false;          }          return isValid;      }      validateField(field) {          const value = field.value.trim();          let message = '';          switch (field.name) {              case 'name':                  if (value.length < 2) message = 'Минимум 2 символа';                  break;              case 'email':                  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;                  if (!emailRegex.test(value)) message = 'Некорректный email';                  break;              case 'comment':                  if (value.length < 10) message = 'Минимум 10 символов';                  break;          }          if (message) {              this.showFieldError(field.name, message);          } else {              this.clearFieldError(field.name);          }      }      showFieldError(fieldName, message) {          const field = this.form.querySelector(`[name="${fieldName}"]`);          const formGroup = field.closest('.form-group');            let errorElement = formGroup.querySelector('.validation-message');          if (!errorElement) {              errorElement = document.createElement('div');              errorElement.className = 'validation-message';              formGroup.appendChild(errorElement);          }            errorElement.textContent = message;          field.classList.add('error');      }      clearFieldError(fieldName) {          const field = this.form.querySelector(`[name="${fieldName}"]`);          const formGroup = field.closest('.form-group');          const errorElement = formGroup.querySelector('.validation-message');            if (errorElement) {              errorElement.remove();          }          field.classList.remove('error');      }      clearValidationMessage(field) {          this.clearFieldError(field.name);      }      async loadComments(silent = false) {          if (!silent) {              this.showLoading();          }          try {              const response = await this.api.getComments();                if (response.success) {                  this.renderComments(response.comments);                  this.hideError();              } else {                  this.showError('Ошибка загрузки комментариев');              }          } catch (error) {              console.error('Error loading comments:', error);              if (!silent) {                  this.showError('Ошибка соединения с сервером');              }          } finally {              this.hideLoading();          }      }      renderComments(comments) {          if (!this.commentsList) return;          if (comments.length === 0) {              this.commentsList.innerHTML = `                  <div class="empty-comments">                      <div class="empty-comments-icon">💬</div>                      <p>Пока нет комментариев. Будьте первым!</p>                  </div>              `;              return;          }          this.commentsList.innerHTML = comments.map(comment =>              this.renderComment(comment)          ).join('');          // Add animation to new comments          const commentElements = this.commentsList.querySelectorAll('.comment-item');          commentElements.forEach((el, index) => {              setTimeout(() => {                  el.style.opacity = '1';                  el.style.transform = 'translateY(0)';              }, index \* 100);          });      }      renderComment(comment) {          const date = new Date(comment.created\_at).toLocaleString('ru-RU', {              year: 'numeric',              month: 'short',              day: 'numeric',              hour: '2-digit',              minute: '2-digit'          });          return `              <div class="comment-item" style="opacity: 0; transform: translateY(20px); transition: all 0.3s ease;">                  <div class="comment-header">                      <div class="comment-author">                          <div class="comment-author-name">${this.escapeHtml(comment.name)}</div>                          <div class="comment-author-email">${this.escapeHtml(comment.email)}</div>                      </div>                      <div class="comment-date">${date}</div>                  </div>                  <p class="comment-text">${this.escapeHtml(comment.comment)}</p>                  <div class="comment-actions">                      <button class="comment-action" onclick="commentsManager.likeComment(${comment.id})">                          👍 Нравится                      </button>                      <button class="comment-action" onclick="commentsManager.reportComment(${comment.id})">                          ⚠️ Пожаловаться                      </button>                  </div>              </div>          `;      }      async updateCommentsCount() {          try {              const response = await this.api.getCommentsCount();              if (response.success && this.commentsCount) {                  this.commentsCount.textContent = response.total;              }          } catch (error) {              console.error('Error updating comments count:', error);          }      }      likeComment(id) {          this.showMessage('Функция "Нравится" будет добавлена в следующей версии!', 'info');      }      reportComment(id) {          if (confirm('Пожаловаться на этот комментарий?')) {              this.showMessage('Жалоба отправлена. Спасибо!', 'success');          }      }      showLoading() {          if (this.loadingElement) {              this.loadingElement.style.display = 'block';          }      }      hideLoading() {          if (this.loadingElement) {              this.loadingElement.style.display = 'none';          }      }      showError(message) {          if (this.errorElement) {              this.errorElement.style.display = 'block';              this.errorElement.querySelector('p').textContent = message;          }      }      hideError() {          if (this.errorElement) {              this.errorElement.style.display = 'none';          }      }      showMessage(message, type = 'info') {          const messageElement = document.createElement('div');          messageElement.className = `${type}-message`;          messageElement.textContent = message;          // Insert before comment form          const commentForm = document.querySelector('.comment-form');          if (commentForm) {              commentForm.parentNode.insertBefore(messageElement, commentForm);          }          // Auto-remove after 5 seconds          setTimeout(() => {              messageElement.remove();          }, 5000);      }      escapeHtml(text) {          const div = document.createElement('div');          div.textContent = text;          return div.innerHTML;      }  }  // ===== INITIALIZATION =====  let commentsManager;  document.addEventListener('DOMContentLoaded', () => {      // Initialize comments manager if comments section exists      if (document.getElementById('comments')) {          commentsManager = new CommentsManager();          console.log('💬 Comments system initialized');      }  });  // Global function for manual refresh  window.loadComments = () => {      if (commentsManager) {          commentsManager.loadComments();      }  }; |
| comments.css |
| /\* ===== COMMENTS SECTION ===== \*/  .comments {      padding: var(--spacing-2xl) 0;      background: linear-gradient(135deg, #f1f5f9 0%, #e2e8f0 100%);  }  .comment-form {      background: linear-gradient(135deg, #ffffff 0%, #f8fafc 100%);      padding: var(--spacing-xl);      border-radius: 20px;      box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);      margin-bottom: var(--spacing-xl);      border: 1px solid #e2e8f0;  }  .comment-form h3 {      margin-bottom: var(--spacing-lg);      color: var(--text-color);      font-size: var(--font-size-2xl);      font-weight: 600;      text-align: center;  }  .comment-form\_\_form {      display: grid;      gap: var(--spacing-md);  }  .form-row {      display: grid;      grid-template-columns: 1fr 1fr;      gap: var(--spacing-md);  }  .form-group {      display: flex;      flex-direction: column;  }  .form-group label {      margin-bottom: var(--spacing-xs);      font-weight: 600;      color: var(--text-color);      font-size: var(--font-size-sm);      text-transform: uppercase;      letter-spacing: 0.5px;  }  .form-group input,  .form-group textarea {      padding: 16px 20px;      border: 2px solid #e2e8f0;      border-radius: 12px;      font-size: var(--font-size-base);      font-family: inherit;      transition: all 0.3s ease;      background: rgba(255, 255, 255, 0.9);      color: #1e293b;      backdrop-filter: blur(10px);  }  .form-group input:focus,  .form-group textarea:focus {      outline: none;      border-color: var(--primary-color);      background: rgba(255, 255, 255, 1);      box-shadow: 0 0 0 4px rgba(37, 99, 235, 0.1);      transform: translateY(-2px);  }  .form-group input::placeholder,  .form-group textarea::placeholder {      color: #94a3b8;      font-style: italic;  }  .form-group textarea {      resize: vertical;      min-height: 120px;      font-family: inherit;  }  .btn--small {      padding: 8px 16px;      font-size: var(--font-size-sm);      border-radius: 8px;  }  .btn\_\_spinner {      display: inline-block;      animation: spin 1s linear infinite;  }  @keyframes spin {      from { transform: rotate(0deg); }      to { transform: rotate(360deg); }  }  /\* Comments List - Светлая тема \*/  .comments-list {      background: linear-gradient(135deg, #ffffff 0%, #f8fafc 100%);      border-radius: 20px;      box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);      overflow: hidden;      border: 1px solid #e2e8f0;  }  .comments-header {      display: flex;      justify-content: space-between;      align-items: center;      padding: var(--spacing-lg) var(--spacing-xl);      background: linear-gradient(135deg, var(--primary-color) 0%, #1d4ed8 100%);      color: white;  }  .comments-header h3 {      margin: 0;      color: white;      font-size: var(--font-size-xl);      font-weight: 700;  }  .comments-count {      background: rgba(255, 255, 255, 0.2);      color: white;      padding: 6px 12px;      border-radius: 20px;      font-size: var(--font-size-sm);      font-weight: 700;      margin-left: var(--spacing-sm);      backdrop-filter: blur(10px);      border: 1px solid rgba(255, 255, 255, 0.3);  }  .comments-items {      max-height: 600px;      overflow-y: auto;      scrollbar-width: thin;      scrollbar-color: var(--primary-color) #f1f5f9;      background: #ffffff;  }  .comments-items::-webkit-scrollbar {      width: 8px;  }  .comments-items::-webkit-scrollbar-track {      background: #f1f5f9;  }  .comments-items::-webkit-scrollbar-thumb {      background: var(--primary-color);      border-radius: 4px;  }  .comment-item {      padding: var(--spacing-lg) var(--spacing-xl);      border-bottom: 1px solid #e2e8f0;      transition: all 0.3s ease;      position: relative;      background: #ffffff;  }  .comment-item:last-child {      border-bottom: none;  }  .comment-item:hover {      background: linear-gradient(135deg, #f8fafc 0%, #f1f5f9 100%);      transform: translateX(8px);  }  .comment-item::before {      content: '';      position: absolute;      left: 0;      top: 0;      width: 4px;      height: 100%;      background: linear-gradient(135deg, var(--primary-color) 0%, var(--accent-color) 100%);      opacity: 0;      transition: opacity 0.3s ease;  }  .comment-item:hover::before {      opacity: 1;  }  .comment-header {      display: flex;      justify-content: space-between;      align-items: flex-start;      margin-bottom: var(--spacing-md);  }  .comment-author {      display: flex;      flex-direction: column;      gap: 6px;  }  .comment-author-name {      font-weight: 700;      color: #1e293b;      font-size: var(--font-size-lg);      display: flex;      align-items: center;      gap: 8px;  }  .comment-author-name::before {      content: '👤';      font-size: 16px;  }  .comment-author-email {      font-size: var(--font-size-sm);      color: #64748b;      font-weight: 500;      display: flex;      align-items: center;      gap: 6px;  }  .comment-author-email::before {      content: '✉️';      font-size: 12px;  }  .comment-date {      font-size: var(--font-size-sm);      color: #64748b;      white-space: nowrap;      background: rgba(37, 99, 235, 0.1);      padding: 6px 12px;      border-radius: 20px;      font-weight: 500;      display: flex;      align-items: center;      gap: 6px;  }  .comment-date::before {      content: '🕒';      font-size: 12px;  }  .comment-text {      color: #1e293b;      line-height: 1.8;      margin: 0;      font-size: var(--font-size-base);      background: linear-gradient(135deg, #f8fafc 0%, #f1f5f9 100%);      padding: var(--spacing-md) var(--spacing-lg);      border-radius: 12px;      border-left: 4px solid var(--primary-color);      font-weight: 400;      border: 1px solid #e2e8f0;  }  .comment-actions {      margin-top: var(--spacing-md);      display: flex;      gap: var(--spacing-sm);  }  .comment-action {      background: linear-gradient(135deg, #f8fafc 0%, #e2e8f0 100%);      border: 1px solid #cbd5e1;      color: #475569;      cursor: pointer;      font-size: var(--font-size-sm);      padding: 8px 16px;      border-radius: 20px;      transition: all 0.3s ease;      font-weight: 500;      display: flex;      align-items: center;      gap: 6px;  }  .comment-action:hover {      background: linear-gradient(135deg, var(--primary-color) 0%, #1d4ed8 100%);      color: white;      transform: translateY(-2px);      box-shadow: 0 4px 12px rgba(37, 99, 235, 0.3);  }  .comment-action--delete:hover {      background: linear-gradient(135deg, #ef4444 0%, #dc2626 100%);      border-color: #ef4444;  }  /\* Loading and Error States \*/  .loading {      padding: var(--spacing-xl);      text-align: center;      color: #64748b;      font-size: var(--font-size-lg);      background: #ffffff;  }  .loading::before {      content: '⏳';      display: block;      font-size: 2rem;      margin-bottom: var(--spacing-sm);  }  .error-message {      padding: var(--spacing-lg);      text-align: center;      color: #dc2626;      background: linear-gradient(135deg, #fef2f2 0%, #fee2e2 100%);      border: 2px solid #fecaca;      border-radius: 16px;      margin: var(--spacing-md);  }  .error-message::before {      content: '⚠️';      display: block;      font-size: 2rem;      margin-bottom: var(--spacing-sm);  }  .error-message button {      background: linear-gradient(135deg, #ef4444 0%, #dc2626 100%);      border: none;      color: white;      cursor: pointer;      padding: 8px 16px;      border-radius: 8px;      font-weight: 600;      margin-top: var(--spacing-sm);      transition: all 0.3s ease;  }  .error-message button:hover {      transform: translateY(-2px);      box-shadow: 0 4px 12px rgba(239, 68, 68, 0.3);  }  .success-message {      padding: var(--spacing-lg);      text-align: center;      color: #059669;      background: linear-gradient(135deg, #ecfdf5 0%, #d1fae5 100%);      border: 2px solid #a7f3d0;      border-radius: 16px;      margin-bottom: var(--spacing-md);      font-weight: 600;  }  .success-message::before {      content: '✅';      display: block;      font-size: 2rem;      margin-bottom: var(--spacing-sm);  }  .empty-comments {      padding: var(--spacing-2xl);      text-align: center;      color: #64748b;      background: #ffffff;  }  .empty-comments-icon {      font-size: 4rem;      margin-bottom: var(--spacing-md);      opacity: 0.6;  }  /\* Responsive Design \*/  @media (max-width: 768px) {      .form-row {          grid-template-columns: 1fr;      }        .comments-header {          flex-direction: column;          gap: var(--spacing-sm);          align-items: stretch;          text-align: center;      }        .comment-header {          flex-direction: column;          gap: var(--spacing-xs);      }        .comment-date {          align-self: flex-start;      }        .form-group input,      .form-group textarea {          padding: 12px 16px;          font-size: var(--font-size-sm);      }        .comment-form {          padding: var(--spacing-md);          border-radius: 16px;      }        .comments-list {          border-radius: 16px;      }        .comments-items {          max-height: 400px;      }        .comment-item {          padding: var(--spacing-md);      }        .comment-item:hover {          transform: none;      }  }  /\* Animation for new comments \*/  @keyframes slideInComment {      from {          opacity: 0;          transform: translateY(-20px) scale(0.95);      }      to {          opacity: 1;          transform: translateY(0) scale(1);      }  }  .comment-item.new {      animation: slideInComment 0.5s ease-out;  }  /\* Form validation styles \*/  .form-group input:user-invalid,  .form-group textarea:user-invalid {      border-color: #ef4444;      background: linear-gradient(135deg, #fef2f2 0%, rgba(255, 255, 255, 0.9) 100%);      color: #1e293b;  }  .form-group input:user-valid,  .form-group textarea:user-valid {      border-color: #10b981;      background: linear-gradient(135deg, #ecfdf5 0%, rgba(255, 255, 255, 0.9) 100%);      color: #1e293b;  }  .validation-message {      margin-top: 6px;      font-size: var(--font-size-sm);      color: #ef4444;      font-weight: 500;      display: flex;      align-items: center;      gap: 6px;  }  .validation-message::before {      content: '⚠️';  } |
| **Backend** |
| package.json |
| {    "name": "portfolio-backend",    "version": "1.0.0",    "description": "Backend API for portfolio comments",    "main": "app.js",    "scripts": {      "start": "node app.js",      "dev": "nodemon app.js"    },    "dependencies": {      "express": "^4.18.2",      "cors": "^2.8.5",      "mysql2": "^3.6.0",      "body-parser": "^1.20.2"    },    "keywords": [      "api",      "comments",      "portfolio"    ],    "author": "Khon Komilov",    "license": "MIT"  } |
| app.js |
| const express = require('express');  const cors = require('cors');  const bodyParser = require('body-parser');  const mysql = require('mysql2/promise');  const app = express();  const PORT = process.env.PORT || 3000;  // Middleware  app.use(cors());  app.use(bodyParser.json({ limit: '10mb' }));  app.use(bodyParser.urlencoded({ extended: true, limit: '10mb' }));  // Устанавливаем кодировку для ответов  app.use((req, res, next) => {      res.setHeader('Content-Type', 'application/json; charset=utf-8');      next();  });  // Database configuration  const dbConfig = {      host: process.env.DB\_HOST || 'database',      user: process.env.DB\_USER || 'portfolio\_user',      password: process.env.DB\_PASSWORD || 'portfolio\_password',      database: process.env.DB\_NAME || 'portfolio\_db',      charset: 'utf8mb4',      timezone: '+00:00',      waitForConnections: true,      connectionLimit: 10,      queueLimit: 0  };  let pool;  // Initialize database connection  async function initDatabase() {      try {          pool = mysql.createPool(dbConfig);          console.log('✅ Database connected successfully');            // Test connection          const connection = await pool.getConnection();          await connection.ping();          connection.release();            console.log('✅ Database connection tested successfully');      } catch (error) {          console.error('❌ Database connection failed:', error);          // Retry connection after 5 seconds          setTimeout(initDatabase, 5000);      }  }  // Routes  // Health check  app.get('/api/health', (req, res) => {      res.json({          status: 'OK',          message: 'Portfolio Backend API is running',          timestamp: new Date().toISOString()      });  });  // Get all comments  app.get('/api/comments', async (req, res) => {      try {          const [rows] = await pool.execute(              'SELECT id, name, email, comment, created\_at FROM comments ORDER BY created\_at DESC'          );          res.json({              success: true,              comments: rows          });      } catch (error) {          console.error('Error fetching comments:', error);          res.status(500).json({              success: false,              message: 'Ошибка при получении комментариев'          });      }  });  // Add new comment  app.post('/api/comments', async (req, res) => {      try {          const { name, email, comment } = req.body;            // Validation          if (!name || !email || !comment) {              return res.status(400).json({                  success: false,                  message: 'Все поля обязательны для заполнения'              });          }            // Email validation          const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;          if (!emailRegex.test(email)) {              return res.status(400).json({                  success: false,                  message: 'Неверный формат email'              });          }            // Insert comment          const [result] = await pool.execute(              'INSERT INTO comments (name, email, comment) VALUES (?, ?, ?)',              [name, email, comment]          );            // Get the newly created comment          const [newComment] = await pool.execute(              'SELECT id, name, email, comment, created\_at FROM comments WHERE id = ?',              [result.insertId]          );            res.status(201).json({              success: true,              message: 'Комментарий успешно добавлен',              comment: newComment[0]          });        } catch (error) {          console.error('Error adding comment:', error);          res.status(500).json({              success: false,              message: 'Ошибка при добавлении комментария'          });      }  });  // Delete comment (optional)  app.delete('/api/comments/:id', async (req, res) => {      try {          const { id } = req.params;            const [result] = await pool.execute(              'DELETE FROM comments WHERE id = ?',              [id]          );            if (result.affectedRows === 0) {              return res.status(404).json({                  success: false,                  message: 'Комментарий не найден'              });          }            res.json({              success: true,              message: 'Комментарий удален'          });        } catch (error) {          console.error('Error deleting comment:', error);          res.status(500).json({              success: false,              message: 'Ошибка при удалении комментария'          });      }  });  // Get comments count  app.get('/api/comments/count', async (req, res) => {      try {          const [rows] = await pool.execute(              'SELECT COUNT(\*) as total FROM comments'          );          res.json({              success: true,              total: rows[0].total          });      } catch (error) {          console.error('Error getting comments count:', error);          res.status(500).json({              success: false,              message: 'Ошибка при подсчете комментариев'          });      }  });  // Error handling middleware  app.use((err, req, res, next) => {      console.error('Unhandled error:', err);      res.status(500).json({          success: false,          message: 'Внутренняя ошибка сервера'      });  });  // 404 handler  app.use('\*', (req, res) => {      res.status(404).json({          success: false,          message: 'Эндпоинт не найден'      });  });  // Start server  async function startServer() {      await initDatabase();        app.listen(PORT, '0.0.0.0', () => {          console.log(`🚀 Backend server running on port ${PORT}`);          console.log(`📝 API endpoints:`);          console.log(`   GET  /api/health - Health check`);          console.log(`   GET  /api/comments - Get all comments`);          console.log(`   POST /api/comments - Add new comment`);          console.log(`   GET  /api/comments/count - Get comments count`);          console.log(`   DELETE /api/comments/:id - Delete comment`);      });  }  startServer().catch(console.error); |
| Batabse/init.sql |
| -- Установка кодировки для сессии  SET NAMES utf8mb4;  SET CHARACTER SET utf8mb4;  -- Создание базы данных с правильной кодировкой  CREATE DATABASE IF NOT EXISTS portfolio\_db      CHARACTER SET utf8mb4      COLLATE utf8mb4\_unicode\_ci;  -- Использование базы данных  USE portfolio\_db;  -- Создание пользователя для приложения  CREATE USER IF NOT EXISTS 'portfolio\_user'@'%' IDENTIFIED BY 'portfolio\_password';  GRANT ALL PRIVILEGES ON portfolio\_db.\* TO 'portfolio\_user'@'%';  FLUSH PRIVILEGES;  -- Создание таблицы комментариев с правильной кодировкой  CREATE TABLE IF NOT EXISTS comments (      id INT AUTO\_INCREMENT PRIMARY KEY,      name VARCHAR(100) NOT NULL COMMENT 'Имя автора комментария',      email VARCHAR(150) NOT NULL COMMENT 'Email автора',      comment TEXT NOT NULL COMMENT 'Текст комментария',      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP COMMENT 'Дата создания',      updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP COMMENT 'Дата обновления',        -- Индексы для производительности      INDEX idx\_created\_at (created\_at),      INDEX idx\_email (email)  ) ENGINE=InnoDB    DEFAULT CHARSET=utf8mb4    COLLATE=utf8mb4\_unicode\_ci    COMMENT='Таблица комментариев портфолио';  -- Вставка тестовых данных с русскими комментариями  INSERT INTO comments (name, email, comment) VALUES  ('Алексей Петров', 'alexey@example.com', 'Отличное портфолио! 👍 Впечатляющие навыки веб-разработки и современный подход к созданию сайтов.'),  ('Мария Иванова', 'maria@example.com', 'Очень профессиональный подход к созданию сайта! 🚀 Видно, что автор хорошо разбирается в технологиях. Удачи в карьере!'),  ('Дмитрий Сидоров', 'dmitry@example.com', 'Красивый дизайн и хорошая функциональность! 🎨 Особенно понравилась адаптивность и плавные анимации.'),  ('Анна Козлова', 'anna@example.com', 'Интересный проект! 💡 Видно, что вложено много труда и внимания к деталям. Система комментариев работает отлично!'),  ('Сергей Николаев', 'sergey@example.com', 'Современные технологии и качественная реализация! 🔥 Docker Compose, API, база данных - всё на высшем уровне.');  -- Создание представления для статистики  CREATE VIEW comments\_stats AS  SELECT      COUNT(\*) as total\_comments,      COUNT(DISTINCT email) as unique\_users,      DATE(MAX(created\_at)) as last\_comment\_date,      DATE(MIN(created\_at)) as first\_comment\_date  FROM comments;  -- Процедура для очистки старых комментариев  DELIMITER //  CREATE PROCEDURE CleanOldComments(IN days\_old INT)  BEGIN      DECLARE deleted\_count INT DEFAULT 0;        DELETE FROM comments      WHERE created\_at < DATE\_SUB(NOW(), INTERVAL days\_old DAY);        SET deleted\_count = ROW\_COUNT();        SELECT deleted\_count as deleted\_comments,             CONCAT('Удалено комментариев старше ', days\_old, ' дней') as message;  END //  DELIMITER ;  -- Создание таблицы для логирования действий  CREATE TABLE IF NOT EXISTS comment\_log (      id INT AUTO\_INCREMENT PRIMARY KEY,      action\_type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,      comment\_id INT,      user\_email VARCHAR(150),      action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,      user\_ip VARCHAR(45),      user\_agent TEXT,        INDEX idx\_action\_time (action\_time),      INDEX idx\_action\_type (action\_type)  ) ENGINE=InnoDB    DEFAULT CHARSET=utf8mb4    COLLATE=utf8mb4\_unicode\_ci    COMMENT='Лог действий с комментариями';  -- Триггер для логирования добавления комментариев  DELIMITER //  CREATE TRIGGER comment\_insert\_log  AFTER INSERT ON comments  FOR EACH ROW  BEGIN      INSERT INTO comment\_log (action\_type, comment\_id, user\_email)      VALUES ('INSERT', NEW.id, NEW.email);  END //  DELIMITER ;  -- Триггер для логирования удаления комментариев  DELIMITER //  CREATE TRIGGER comment\_delete\_log  BEFORE DELETE ON comments  FOR EACH ROW  BEGIN      INSERT INTO comment\_log (action\_type, comment\_id, user\_email)      VALUES ('DELETE', OLD.id, OLD.email);  END //  DELIMITER ;  -- Создание функции для подсчета комментариев пользователя  DELIMITER //  CREATE FUNCTION GetUserCommentsCount(user\_email VARCHAR(150))  RETURNS INT  READS SQL DATA  DETERMINISTIC  BEGIN      DECLARE comment\_count INT DEFAULT 0;        SELECT COUNT(\*) INTO comment\_count      FROM comments      WHERE email = user\_email;        RETURN comment\_count;  END //  DELIMITER ;  -- Вывод информации о создании таблиц  SELECT 'База данных portfolio\_db успешно инициализирована!' as status,         COUNT(\*) as initial\_comments\_count  FROM comments; |

1. **Проверка сайта**







1. **Docker файлы и Docker Compose**

|  |
| --- |
| Dockerfile-fronted(Modifited) |
| # Используем официальный образ nginx  FROM nginx:alpine  # Метаданные образа  LABEL maintainer="your.email@example.com"  LABEL version="2.0.0"  LABEL description="Portfolio Frontend with Comments System"  # Удаляем дефолтную страницу nginx  RUN rm -rf /usr/share/nginx/html/\*  # Копируем HTML файл в корень  COPY public/index.html /usr/share/nginx/html/  # Копируем папку src со стилями и скриптами  COPY src/ /usr/share/nginx/html/src/  # Копируем кастомную конфигурацию nginx для проксирования API  COPY nginx.conf /etc/nginx/conf.d/default.conf  # Устанавливаем правильные права доступа  RUN chmod -R 755 /usr/share/nginx/html && \      chown -R nginx:nginx /usr/share/nginx/html  # Экспонируем порт  EXPOSE 80  # Health check  HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \      CMD curl -f http://localhost/ || exit 1  # nginx запускается автоматически |
| Dockerfile-backend |
| FROM node:18-alpine  # Метаданные  LABEL maintainer="luka117wm@icloud.com"  LABEL version="1.0.0"  LABEL description="Portfolio Backend API"  # Создаем пользователя для безопасности  RUN addgroup -g 1001 -S portfolio && \      adduser -S portfolio -u 1001  # Создаем рабочую директорию  WORKDIR /app  # Копируем package files  COPY package\*.json ./  # Устанавливаем зависимости  RUN npm ci --only=production && \      npm cache clean --force  # Копируем исходный код  COPY app.js .  # Меняем владельца файлов  RUN chown -R portfolio:portfolio /app  # Переключаемся на непривилегированного пользователя  USER portfolio  # Экспонируем порт  EXPOSE 3000  # Health check  HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \      CMD node -e "require('http').get('http://localhost:3000/api/health', (res) => { process.exit(res.statusCode === 200 ? 0 : 1) }).on('error', () => process.exit(1))"  # Запускаем приложение  CMD ["node", "app.js"] |
| nginx.conf |
| server {      listen 80;      server\_name localhost;      root /usr/share/nginx/html;      index index.html;      # Security headers      add\_header X-Frame-Options "SAMEORIGIN" always;      add\_header X-XSS-Protection "1; mode=block" always;      add\_header X-Content-Type-Options "nosniff" always;      add\_header Referrer-Policy "no-referrer-when-downgrade" always;      # Gzip compression      gzip on;      gzip\_vary on;      gzip\_min\_length 1024;      gzip\_proxied expired no-cache no-store private auth;      gzip\_types          text/plain          text/css          text/xml          text/javascript          application/javascript          application/xml+rss          application/json;      # API proxy to backend      location /api/ {          proxy\_pass http://backend:3000/api/;          proxy\_http\_version 1.1;          proxy\_set\_header Upgrade $http\_upgrade;          proxy\_set\_header Connection 'upgrade';          proxy\_set\_header Host $host;          proxy\_set\_header X-Real-IP $remote\_addr;          proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;          proxy\_set\_header X-Forwarded-Proto $scheme;          proxy\_cache\_bypass $http\_upgrade;            # CORS headers          add\_header 'Access-Control-Allow-Origin' '\*' always;          add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, DELETE, PUT' always;          add\_header 'Access-Control-Allow-Headers' 'Content-Type, Authorization' always;            # Handle preflight requests          if ($request\_method = 'OPTIONS') {              add\_header 'Access-Control-Allow-Origin' '\*';              add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, DELETE, PUT';              add\_header 'Access-Control-Allow-Headers' 'Content-Type, Authorization';              add\_header 'Access-Control-Max-Age' 1728000;              add\_header 'Content-Type' 'text/plain; charset=utf-8';              add\_header 'Content-Length' 0;              return 204;          }      }      # Cache static assets      location ~\* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {          expires 1y;          add\_header Cache-Control "public, immutable";      }      # Main location      location / {          try\_files $uri $uri/ /index.html;            # Cache HTML for short time          expires 1h;          add\_header Cache-Control "public";      }      # Handle 404 errors      error\_page 404 /index.html;      # Security: deny access to hidden files      location ~ /\. {          deny all;      }  } |
| docker-compose.yml |
| version: '3.8'  services:    # Frontend - веб-страница портфолио    frontend:      build:        context: ./frontend        dockerfile: Dockerfile      container\_name: portfolio-frontend      ports:        - "8080:80"      depends\_on:        - backend      networks:        - portfolio-network      restart: unless-stopped      labels:        - "traefik.enable=true"        - "traefik.http.routers.frontend.rule=Host(`localhost`)"      healthcheck:        test: ["CMD", "curl", "-f", "http://localhost"]        interval: 30s        timeout: 10s        retries: 3        start\_period: 40s    # Backend - API для комментариев    backend:      build:        context: ./backend        dockerfile: Dockerfile      container\_name: portfolio-backend      ports:        - "3000:3000"      environment:        - NODE\_ENV=production        - PORT=3000        - DB\_HOST=database        - DB\_USER=portfolio\_user        - DB\_PASSWORD=portfolio\_password        - DB\_NAME=portfolio\_db      depends\_on:        database:          condition: service\_healthy      networks:        - portfolio-network      restart: unless-stopped      volumes:        - ./logs:/app/logs      labels:        - "traefik.enable=true"        - "traefik.http.routers.backend.rule=Host(`api.localhost`)"    # Database - MySQL для хранения комментариев    database:      image: mysql:8.0      container\_name: portfolio-database      environment:        - MYSQL\_ROOT\_PASSWORD=root\_password        - MYSQL\_DATABASE=portfolio\_db        - MYSQL\_USER=portfolio\_user        - MYSQL\_PASSWORD=portfolio\_password      ports:        - "3306:3306"      volumes:        - db\_data:/var/lib/mysql        - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql:ro        - ./database/my.cnf:/etc/mysql/conf.d/my.cnf:ro      networks:        - portfolio-network      restart: unless-stopped      healthcheck:        test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "portfolio\_user", "-pportfolio\_password"]        interval: 30s        timeout: 10s        retries: 5        start\_period: 30s      command: --default-authentication-plugin=mysql\_native\_password    # Adminer - веб-интерфейс для управления БД (опционально)    adminer:      image: adminer:latest      container\_name: portfolio-adminer      ports:        - "8081:8080"      environment:        - ADMINER\_DEFAULT\_SERVER=database      depends\_on:        - database      networks:        - portfolio-network      restart: unless-stopped    # Redis - кэш для комментариев (опционально)    redis:      image: redis:7-alpine      container\_name: portfolio-redis      ports:        - "6379:6379"      volumes:        - redis\_data:/data      networks:        - portfolio-network      restart: unless-stopped      healthcheck:        test: ["CMD", "redis-cli", "ping"]        interval: 30s        timeout: 3s        retries: 3      command: redis-server --appendonly yes --requirepass redis\_password  # Именованные тома для постоянного хранения данных  volumes:    db\_data:      driver: local      labels:        - "backup.enable=true"    redis\_data:      driver: local    logs:      driver: local  # Сетевая конфигурация  networks:    portfolio-network:      driver: bridge      ipam:        config:          - subnet: 172.20.0.0/16      labels:        - "project=portfolio"        - "environment=development" |

1. **Запускаем docker-compose**

|  |
| --- |
|  |
| Проверка контейнеров |
|  |

1. **Тотальная проверка**

Чтобы вручную не проверить работоспособность каждого порта, напишем скрипт для проверки

|  |
| --- |
| Check-services.sh |
| #!/bin/bash  echo "🔍 Проверка статуса Docker Compose сервисов"  echo "=============================================="  # Цвета для вывода  RED='\033[0;31m'  GREEN='\033[0;32m'  YELLOW='\033[1;33m'  BLUE='\033[0;34m'  NC='\033[0m' # No Color  echo -e "\n${BLUE}📊 Статус контейнеров:${NC}"  docker-compose ps  echo -e "\n${BLUE}🌐 Проверка Frontend (порт 8080):${NC}"  if curl -s -o /dev/null -w "%{http\_code}" http://localhost:8080 | grep -q "200"; then      echo -e "${GREEN}✅ Frontend доступен: http://localhost:8080${NC}"  else      echo -e "${RED}❌ Frontend недоступен${NC}"  fi  echo -e "\n${BLUE}🚀 Проверка Backend API (порт 3000):${NC}"  if curl -s http://localhost:3000/api/health | grep -q "OK"; then      echo -e "${GREEN}✅ Backend API работает: http://localhost:3000/api/health${NC}"        # Проверка количества комментариев      COMMENTS\_COUNT=$(curl -s http://localhost:3000/api/comments/count | grep -o '"total":[0-9]\*' | cut -d':' -f2)      echo -e "${GREEN}💬 Комментариев в базе: $COMMENTS\_COUNT${NC}"  else      echo -e "${RED}❌ Backend API недоступен${NC}"  fi  echo -e "\n${BLUE}🗄️ Проверка базы данных MySQL (порт 3306):${NC}"  if docker-compose exec -T database mysqladmin ping -h localhost -u portfolio\_user -pportfolio\_password --silent; then      echo -e "${GREEN}✅ MySQL база данных работает${NC}"        # Проверка таблиц      TABLES=$(docker-compose exec -T database mysql -u portfolio\_user -pportfolio\_password -D portfolio\_db -e "SHOW TABLES;" --silent | wc -l)      echo -e "${GREEN}📋 Таблиц в базе: $TABLES${NC}"        # Проверка кодировки      CHARSET=$(docker-compose exec -T database mysql -u portfolio\_user -pportfolio\_password -D portfolio\_db -e "SHOW VARIABLES LIKE 'character\_set\_database';" --silent | awk '{print $2}')      echo -e "${GREEN}🔤 Кодировка базы: $CHARSET${NC}"  else      echo -e "${RED}❌ MySQL база данных недоступна${NC}"  fi  echo -e "\n${BLUE}🖥️ Проверка Adminer (порт 8081):${NC}"  if curl -s -o /dev/null -w "%{http\_code}" http://localhost:8081 | grep -q "200"; then      echo -e "${GREEN}✅ Adminer доступен: http://localhost:8081${NC}"      echo -e "${YELLOW}   Логин: portfolio\_user | Пароль: portfolio\_password${NC}"  else      echo -e "${RED}❌ Adminer недоступен${NC}"  fi  echo -e "\n${BLUE}📡 Проверка сетевого взаимодействия:${NC}"  if docker-compose exec -T backend curl -s http://database:3306 > /dev/null 2>&1; then      echo -e "${GREEN}✅ Backend может подключиться к базе данных${NC}"  else      echo -e "${RED}❌ Проблемы с подключением backend к базе${NC}"  fi  echo -e "\n${BLUE}💾 Использование ресурсов:${NC}"  echo "Контейнер          CPU %    Память"  docker stats --no-stream --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}" | grep portfolio  echo -e "\n${BLUE}📦 Информация о томах:${NC}"  docker volume ls | grep portfolio  echo -e "\n${BLUE}🔗 Полезные ссылки:${NC}"  echo -e "${GREEN}Frontend:${NC} http://localhost:8080"  echo -e "${GREEN}Backend API:${NC} http://localhost:3000/api/health"  echo -e "${GREEN}Adminer:${NC} http://localhost:8081"  echo -e "${GREEN}API Комментарии:${NC} http://localhost:3000/api/comments"  echo -e "\n${YELLOW}📝 Команды для отладки:${NC}"  echo "docker-compose logs -f [service\_name]"  echo "docker-compose exec [service\_name] sh"  echo "docker-compose restart [service\_name]"  echo -e "\n${GREEN}🎉 Проверка завершена!${NC}" |
|  |
| Проверка каждого порта |
| <http://localhost:8080> |
|  |
| <http://localhost:3000/api/health> |
|  |
| <http://localhost:8081> |
|  |
| <http://localhost:3000/api/comments> |
|  |

1. **Деплой в DockerHub**

|  |
| --- |
| deploy.sh |
| #!/bin/bash  # Цвета для вывода  RED='\033[0;31m'  GREEN='\033[0;32m'  YELLOW='\033[1;33m'  BLUE='\033[0;34m'  NC='\033[0m' # No Color  # Конфигурация  DOCKER\_USERNAME="luka117wm"  FRONTEND\_IMAGE="$DOCKER\_USERNAME/portfolio-frontend"  BACKEND\_IMAGE="$DOCKER\_USERNAME/portfolio-backend"  VERSION="v2.0"  echo -e "${BLUE}🚀 Начинаем деплой портфолио на Docker Hub${NC}"  echo "=============================================="  # Проверка авторизации в Docker Hub  echo -e "\n${YELLOW}🔐 Проверка авторизации в Docker Hub...${NC}"  # Проверяем файл конфигурации Docker  if [ -f ~/.docker/config.json ] && grep -q "https://index.docker.io/v1/" ~/.docker/config.json; then      echo -e "${GREEN}✅ Авторизация в Docker Hub подтверждена${NC}"      DOCKER\_USERNAME=$(cat ~/.docker/config.json | grep -A 10 "https://index.docker.io/v1/" | grep -o '"username":"[^"]\*"' | cut -d'"' -f4 2>/dev/null || echo "luka117wm")      echo -e "${BLUE}👤 Пользователь: $DOCKER\_USERNAME${NC}"  else      # Альтернативная проверка через docker system info      if docker system info 2>/dev/null | grep -q "Username:"; then          echo -e "${GREEN}✅ Авторизация в Docker Hub подтверждена${NC}"          DOCKER\_USERNAME=$(docker system info 2>/dev/null | grep "Username:" | awk '{print $2}' || echo "luka117wm")      else          echo -e "${YELLOW}⚠️ Не удается определить статус авторизации, но продолжаем...${NC}"          # Используем username из скрипта          if [ "$DOCKER\_USERNAME" = "your-dockerhub-username" ]; then              echo -e "${RED}❌ Необходимо изменить DOCKER\_USERNAME в скрипте!${NC}"              echo -e "${YELLOW}Измените строку: DOCKER\_USERNAME=\"luka117wm\"${NC}"              exit 1          fi      fi  fi  # Остановка текущих контейнеров  echo -e "\n${YELLOW}⏹️ Остановка текущих контейнеров...${NC}"  docker-compose down  # Сборка образов  echo -e "\n${BLUE}🔨 Сборка образов...${NC}"  echo -e "${YELLOW}📦 Сборка Frontend...${NC}"  if docker build -t $FRONTEND\_IMAGE:latest -t $FRONTEND\_IMAGE:$VERSION ./frontend; then      echo -e "${GREEN}✅ Frontend собран успешно${NC}"  else      echo -e "${RED}❌ Ошибка сборки Frontend${NC}"      exit 1  fi  echo -e "${YELLOW}📦 Сборка Backend...${NC}"  if docker build -t $BACKEND\_IMAGE:latest -t $BACKEND\_IMAGE:$VERSION ./backend; then      echo -e "${GREEN}✅ Backend собран успешно${NC}"  else      echo -e "${RED}❌ Ошибка сборки Backend${NC}"      exit 1  fi  # Пуш образов в Docker Hub  echo -e "\n${BLUE}☁️ Загрузка образов в Docker Hub...${NC}"  echo -e "${YELLOW}⬆️ Загрузка Frontend:latest...${NC}"  if docker push $FRONTEND\_IMAGE:latest; then      echo -e "${GREEN}✅ Frontend:latest загружен${NC}"  else      echo -e "${RED}❌ Ошибка загрузки Frontend:latest${NC}"      exit 1  fi  echo -e "${YELLOW}⬆️ Загрузка Frontend:$VERSION...${NC}"  if docker push $FRONTEND\_IMAGE:$VERSION; then      echo -e "${GREEN}✅ Frontend:$VERSION загружен${NC}"  else      echo -e "${RED}❌ Ошибка загрузки Frontend:$VERSION${NC}"      exit 1  fi  echo -e "${YELLOW}⬆️ Загрузка Backend:latest...${NC}"  if docker push $BACKEND\_IMAGE:latest; then      echo -e "${GREEN}✅ Backend:latest загружен${NC}"  else      echo -e "${RED}❌ Ошибка загрузки Backend:latest${NC}"      exit 1  fi  echo -e "${YELLOW}⬆️ Загрузка Backend:$VERSION...${NC}"  if docker push $BACKEND\_IMAGE:$VERSION; then      echo -e "${GREEN}✅ Backend:$VERSION загружен${NC}"  else      echo -e "${RED}❌ Ошибка загрузки Backend:$VERSION${NC}"      exit 1  fi  # Создание docker-compose для продакшена  echo -e "\n${BLUE}📄 Создание production docker-compose.yml...${NC}"  cat > docker-compose.production.yml << EOF  version: '3.8'  services:    frontend:      image: $FRONTEND\_IMAGE:latest      container\_name: portfolio-frontend      ports:        - "8080:80"      depends\_on:        - backend      networks:        - portfolio-network      restart: unless-stopped    backend:      image: $BACKEND\_IMAGE:latest      container\_name: portfolio-backend      ports:        - "3000:3000"      environment:        - NODE\_ENV=production        - PORT=3000        - DB\_HOST=database        - DB\_USER=portfolio\_user        - DB\_PASSWORD=portfolio\_password        - DB\_NAME=portfolio\_db      depends\_on:        - database      networks:        - portfolio-network      restart: unless-stopped    database:      image: mysql:8.0      container\_name: portfolio-database      environment:        - MYSQL\_ROOT\_PASSWORD=root\_password        - MYSQL\_DATABASE=portfolio\_db        - MYSQL\_USER=portfolio\_user        - MYSQL\_PASSWORD=portfolio\_password      ports:        - "3306:3306"      volumes:        - db\_data:/var/lib/mysql        - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql:ro      networks:        - portfolio-network      restart: unless-stopped      command:        - --default-authentication-plugin=mysql\_native\_password        - --character-set-server=utf8mb4        - --collation-server=utf8mb4\_unicode\_ci    adminer:      image: adminer:latest      container\_name: portfolio-adminer      ports:        - "8081:8080"      depends\_on:        - database      networks:        - portfolio-network      restart: unless-stopped  volumes:    db\_data:  networks:    portfolio-network:      driver: bridge  EOF  echo -e "${GREEN}✅ Production docker-compose.yml создан${NC}"  # Тестовый запуск  echo -e "\n${BLUE}🧪 Тестовый запуск с Docker Hub образами...${NC}"  if docker-compose -f docker-compose.production.yml up -d; then      echo -e "${GREEN}✅ Тестовый запуск успешен${NC}"        # Ожидание запуска сервисов      echo -e "${YELLOW}⏳ Ожидание запуска сервисов (30 сек)...${NC}"      sleep 30        # Проверка здоровья      echo -e "${YELLOW}🔍 Проверка работоспособности...${NC}"      if curl -s http://localhost:3000/api/health | grep -q "OK"; then          echo -e "${GREEN}✅ Backend API работает${NC}"      else          echo -e "${RED}❌ Backend API не отвечает${NC}"      fi        if curl -s -o /dev/null -w "%{http\_code}" http://localhost:8080 | grep -q "200"; then          echo -e "${GREEN}✅ Frontend доступен${NC}"      else          echo -e "${RED}❌ Frontend недоступен${NC}"      fi    else      echo -e "${RED}❌ Ошибка тестового запуска${NC}"      exit 1  fi  # Информация о деплое  echo -e "\n${GREEN}🎉 Деплой завершен успешно!${NC}"  echo "=============================================="  echo -e "${BLUE}📦 Образы в Docker Hub:${NC}"  echo "  • $FRONTEND\_IMAGE:latest"  echo "  • $FRONTEND\_IMAGE:$VERSION"  echo "  • $BACKEND\_IMAGE:latest"  echo "  • $BACKEND\_IMAGE:$VERSION"  echo -e "\n${BLUE}🔗 Ссылки для проверки:${NC}"  echo "  • Frontend: http://localhost:8080"  echo "  • Backend API: http://localhost:3000/api/health"  echo "  • Adminer: http://localhost:8081"  echo -e "\n${BLUE}📋 Команды для запуска на любом сервере:${NC}"  echo "  docker-compose -f docker-compose.production.yml up -d"  echo -e "\n${YELLOW}📝 Для отчета используйте эти команды:${NC}"  echo "  docker pull $FRONTEND\_IMAGE:latest"  echo "  docker pull $BACKEND\_IMAGE:latest"  echo "  docker-compose -f docker-compose.production.yml up -d"  echo -e "\n${GREEN}✨ Готово! Ваше портфолио теперь доступно в Docker Hub${NC}" |
|  |