# Глава 3. Разработка программного продукта

## 3.1 Технические средства реализации

В данном подразделе представлены аппаратные и программные компоненты, указанные в таблице 3.1, использованные при реализации WEB-службы обратной связи, оценки качества сервиса, анкетирования клиентов и формирования статистики службы доставки еды.

## Таблица 3.1 – Средства реализации

|  |  |
| --- | --- |
| Компонент | Описание |
| Аппаратные компоненты | |
| ПК – Марка и модель | Ноутбук Lenovo ThinkPad X1 Carbon Gen 10 |
| ЦП | Intel Core i7-1260P, 2.1 GHz (до 4.7 GHz в режиме Turbo Boost) |
| Объём ОЗУ | 32 Gb DDR5 |
| Объём ПЗУ | SSD 1 TB NVMe |
| Экран, диагональ | 14'' |
| Экран, разрешение | 2880px X 1800px |
| Мышь | Logitech MX Master 3 |
| Клавиатура | Интегрированная |
| Программные компоненты | |
| Операционная система | Windows 11 Pro, версия 22H2 |
| Текстовый редактор | Visual Studio Code, версия 1.84.2 |
| Редактор схем | Figma, версия 116.13.4 |
| Инструментальная среда разработки | WebStorm 2023.2.4 |
| Система контроля версий | Git 2.42.0 |
| Репозиторий кода | GitHub Enterprise |
| Менеджер пакетов | npm 10.1.0 |
| Фреймворк интерфейса | React 18.2.0 |
| Фреймворк серверной части | Node.js 20.9.0, Express 4.18.2 |
| База данных | MongoDB 7.0.2 |
| Инструмент для API-тестирования | Postman 10.19.7 |
| Инструмент для UI/UX тестирования | Selenium WebDriver 4.15.0 |

Продолжение Таблицы 3.1

|  |  |
| --- | --- |
| Система управления проектами | JIRA Software 9.11.0 |
| Графический редактор | Adobe Photoshop 2023 (24.7.0) |
| Браузеры для тестирования | Google Chrome 119.0.6045.160, Mozilla Firefox 119.0.1, Microsoft Edge 119.0.2151.58 |
| Инструмент для мониторинга производительности | New Relic APM 10.5.0 |
| Система автоматизации развертывания | Docker 24.0.5, Docker Compose 2.21.0 |
| Система непрерывной интеграции | Jenkins 2.414.1 |
| Инструмент для анализа кода | ESLint 8.53.0, SonarQube 10.2.1 |
| Система управления базами данных | MongoDB Atlas Cloud |
| Система для аналитики клиентского поведения | Google Analytics 4 |
| Библиотека для создания интерфейса | Material-UI 5.14.11 |
| Библиотека для визуализации данных | Chart.js 4.3.3, D3.js 7.8.5 |
| Библиотека для управления состоянием | Redux 4.2.1, Redux Toolkit 1.9.7 |

## 3.2 Разработка UI/UX

В данном подразделе представлены основные элементы пользовательского интерфейса, разработанные в соответствии с функциональными требованиями WEB-службы обратной связи для службы доставки еды.

**Таблица 3.2 – Реализация интерфейса программного продукта**

|  |  |
| --- | --- |
| Описание задачи | Программный код |
| Разработка главной страницы с доступом к основным функциям системы (анкетирование, просмотр статистики, история отзывов) | const **App** = () => {      const **navigate** = useNavigate();      const location = useLocation();      const { user, isAuthenticated, logout } = useAuth();      const [isMenuOpen, **setIsMenuOpen**] = useState(false);      const [submitted, **setSubmitted**] = useState(false);      const [isLoginModalOpen, **setIsLoginModalOpen**] = useState(false);      const [isRegistration, **setIsRegistration**] = useState(false);      const [isLoading, **setIsLoading**] = useState(false);      const [error, **setError**] = useState(null);      const [reviews, **setReviews**] = useState([]);      const [loading, **setLoading**] = useState(true);      const [isDarkMode, **setIsDarkMode**] = useState(false);        const wsRef = useRef(null);      const [wsConnected, **setWsConnected**] = useState(false); |
| Создание адаптивного меню навигации с возможностью переключения между разделами системы | const **NavigationBar** = ({ *user*, *onLogout*, *onLogin*, *onThemeToggle*, *onProfileClick*, *onProfileUpdate* }) => {  *// Состояния компонента*      const [isDarkMode, **setIsDarkMode**] = useState(false);      const [showNotifications, **setShowNotifications**] = useState(false);      const [showProfileMenu, **setShowProfileMenu**] = useState(false);      const [hasScrolled, **setHasScrolled**] = useState(false);      const [searchQuery, **setSearchQuery**] = useState('');      const [mobileMenuOpen, **setMobileMenuOpen**] = useState(false);      const [notifications, **setNotifications**] = useState([]);      const [unreadCount, **setUnreadCount**] = useState(0);      const [isLoadingNotifications, **setIsLoadingNotifications**] = useState(false);        const **navigate** = useNavigate();  *// Ссылки на DOM-элементы для обработки кликов вне компонентов*      const notificationsRef = useRef(null);      const profileMenuRef = useRef(null);      const searchInputRef = useRef(null);      const mobileMenuRef = useRef(null); |
| Разработка формы для оценки качества доставки по 5-балльной шкале с возможностью добавления текстового комментария | export const **ReviewForm** = ({  *onSubmit*,  *onCancel*,  *initialValues* = { rating: 0, text: '' },  *isSubmitting* = false  }) => {  *// Состояние формы*      const [formData, **setFormData**] = useState(*initialValues*);      const [errors, **setErrors**] = useState({});    *// Обработчик изменения рейтинга*      const **handleRatingChange** = (*value*) => {          setFormData(*prev* => ({ ...*prev*, rating: *value* }));    *// Если была ошибка, убираем её*          if (errors.rating) {              setErrors(*prev* => ({ ...*prev*, rating: undefined }));          }      };    *// Обработчик изменения текста отзыва*      const **handleTextChange** = (*e*) => {          setFormData(*prev* => ({ ...*prev*, text: *e*.target.value }));    *// Если была ошибка, убираем её*          if (errors.text && *e*.target.value.trim()) {              setErrors(*prev* => ({ ...*prev*, text: undefined }));          }      }; |
| Создание интерактивной анкеты для клиентов с различными типами вопросов (множественный выбор, оценка по шкале, открытые вопросы) | const **RestaurantButton** = ({ *restaurant*, *isSelected*, *onSelect* }) => (      <button  *type*="button"  *onClick*={() => onSelect(*restaurant*)}  *className*={`              w-full p-0 rounded-lg transition-all              border relative overflow-hidden shadow-sm              ${*isSelected*              ? 'border-gray-400 dark:border-gray-500 ring-2 ring-gray-400 dark:ring-gray-500'              : 'border-gray-200 dark:border-gray-700 hover:border-gray-300 dark:hover:border-gray-600'}          `}      >          <div *className*="w-full pt-[60%] relative">              <img  *src*={*restaurant*.image}  *alt*={`${*restaurant*.name} cuisine`}  *className*="absolute top-0 left-0 w-full h-full object-cover"              />          </div>          <div *className*="p-2 bg-white dark:bg-gray-800">              <h3 *className*="font-medium text-sm text-gray-700 dark:text-gray-300 truncate">                  {*restaurant*.name}              </h3>              <div *className*="flex items-center justify-between mt-1">                  <p *className*="text-xs text-gray-500 dark:text-gray-400 truncate">                      {*restaurant*.cuisine} | {*restaurant*.priceRange}                  </p>                  <div *className*="flex items-center">                      <Star *className*="w-3 h-3 text-yellow-500 mr-1" />                      <span *className*="text-xs font-medium text-gray-600 dark:text-gray-400">{*restaurant*.avgRating}</span>                  </div>              </div>          </div>      </button>  );  const **StarRating** = ({ *value*, *onClick*, *size* = "normal" }) => {      return (          <motion.button  *type*="button"  *whileHover*={{ scale: 1.2 }}  *whileTap*={{ scale: 0.9 }}  *onClick*={*onClick*}  *className*={`                  ${*size* === "normal" ? 'w-6 h-6' : 'w-4 h-4'} transition-colors                  ${*value* ? 'text-yellow-500' : 'text-gray-300 hover:text-gray-400'}              `}          >              ★          </motion.button>      );  }; |
| Разработка дашборда для отображения статистики отзывов с фильтрами по дате, региону, типу заказа | {*/\* Список отзывов \*/*}                  <div *className*="space-y-4">                      {filteredReviews.map((*review*) => (                          <Card *key*={*review*.id} *className*="bg-white/10 backdrop-blur-sm border border-white/20">                              <CardContent *className*="p-4">                                  <div *className*="flex justify-between items-start">                                      <div *className*="flex-1">                                          <div *className*="flex items-center gap-2 mb-2">                                              <div *className*="flex items-center">                                                  {[...Array(5)].map((*\_*, *i*) => (                                                      <Star  *key*={*i*}  *className*={`w-4 h-4 ${  *i* < *review*.rating                                                                  ? 'text-yellow-400 fill-current'                                                                  : 'text-white/20'                                                          }`}                                                      />                                                  ))}                                              </div>                                              <span *className*="text-white/60 text-sm">                                                  {*review*.customerName} • {*review*.date}                                              </span>                                          </div>                                          <p *className*="text-white mb-4">{*review*.text}</p>                                            {*review*.hasResponse ? (                                              <div *className*="bg-blue-500/10 border border-blue-500/20 rounded-lg p-3">                                                  <p *className*="text-white text-sm">                                                      <span *className*="text-blue-400">Ваш ответ:</span> {*review*.response}                                                  </p>                                              </div>                                          ) : (                                              <div *className*="flex gap-2">                                                  <input  *type*="text"  *placeholder*="Напишите ответ..."  *className*="flex-1 bg-white/5 border border-white/20 rounded-lg text-white px-4 py-2"  *onKeyPress*={(*e*) => {                                                          if (*e*.key === 'Enter' && *e*.target.value.trim()) {                                                              handleResponse(*review*.id, *e*.target.value);  *e*.target.value = '';                                                          }                                                      }}                                                  />                                                  <button  *onClick*={() => {                                                          const input = document.querySelector(`input[data-review-id="${*review*.id}"]`);                                                          if (input && input.value.trim()) {                                                              handleResponse(*review*.id, input.value);                                                              input.value = '';                                                          }                                                      }}  *className*="bg-blue-500/20 hover:bg-blue-500/30 text-blue-400 px-4 py-2 rounded-lg"                                                  >                                                      Отправить                                                  </button>                                              </div>                                          )}                                      </div>                                      {!*review*.hasResponse && (                                          <div *className*="ml-4">                                              <AlertCircle *className*="w-5 h-5 text-yellow-400" />                                          </div>                                      )}                                  </div>                              </CardContent>                          </Card>                      ))} |

Продолжение таблицы 3.2

|  |  |
| --- | --- |
| Создание интерфейса администратора для управления вопросами анкет и анализа результатов | const **AdminPanel** = ({ *user* }) => {      const **navigate** = useNavigate();      const [activeTab, **setActiveTab**] = useState('restaurants');      const [restaurants, **setRestaurants**] = useState([]);      const [users, **setUsers**] = useState([]);      const [reviews, **setReviews**] = useState([]);      const [deletedReviews, **setDeletedReviews**] = useState([]);      const [blockedUsers, **setBlockedUsers**] = useState([]);      const [loading, **setLoading**] = useState({          restaurants: false,          users: false,          reviews: false,          deletedReviews: false,          blockedUsers: false      });      const [deleteReason, **setDeleteReason**] = useState('');      const [customReason, **setCustomReason**] = useState('');      const [deleteReviewId, **setDeleteReviewId**] = useState(null);      const [showDeleteModal, **setShowDeleteModal**] = useState(false);      const [restaurantToDelete, **setRestaurantToDelete**] = useState(null);      const [showDeleteRestaurantModal, **setShowDeleteRestaurantModal**] = useState(false);      const [userToBlock, **setUserToBlock**] = useState(null);      const [showBlockModal, **setShowBlockModal**] = useState(false);      const [blockReason, **setBlockReason**] = useState('');      const [userToUnblock, **setUserToUnblock**] = useState(null);      const [showUnblockModal, **setShowUnblockModal**] = useState(false);    *// После авторизации и проверки прав загружаем заблокированных пользователей*      useEffect(() => {  *// Проверка авторизации*          if (!*user* || !*user*.token) {              navigate('/');              return;          }            const adminRoles = ['admin', 'head\_admin', 'manager', 'moderator', 'super\_admin', 'глав\_админ', 'менеджер', 'модератор'];          if (!adminRoles.includes(*user*.role)) {              navigate('/');              return;          }    *// Автоматически загружаем заблокированных пользователей*          if (*user*.role === 'admin' || *user*.role === 'head\_admin') {              fetchBlockedUsers();          }      }, [*user*, navigate]);  *// Set initial active tab based on user role*      useEffect(() => {          if (*user*?.role === 'manager') {              setActiveTab('reviews');          }      }, [*user*?.role]);  *// Fetch restaurants data*      const **fetchRestaurants** = async () => {          setLoading(*prev* => ({ ...*prev*, restaurants: true }));          try {              const response = await api.get('/restaurants');              setRestaurants(response.data.restaurants || []);          } catch (error) {              console.error('Error fetching restaurants:', error);          } finally {              setLoading(*prev* => ({ ...*prev*, restaurants: false }));          }      }; |
| Разработка системы уведомлений о новых отзывах и критически низких оценках | const NotificationContext = createContext();  */\*\**  *\* Хук для использования контекста уведомлений в компонентах*  *\**  *\* @returns {Object} Объект с методами для работы с уведомлениями*  *\*/*  export const **useNotification** = () => {      return useContext(NotificationContext);  };  */\*\**  *\* Провайдер контекста уведомлений*  *\**  *\* @param {Object} props - Свойства компонента*  *\* @param {React.ReactNode} props.children - Дочерние компоненты*  *\* @returns {JSX.Element} React-компонент провайдера*  *\*/*  export const NotificationProvider = ({ *children* }) => {  *// Состояние для хранения активных уведомлений*      const [notifications, **setNotifications**] = useState([]);  */\*\**  *\* Добавление нового уведомления*  *\**  *\* @param {string} message - Текст сообщения*  *\* @param {string} [type='info'] - Тип уведомления ('info', 'success', 'error')*  *\* @param {number} [autoClose=5000] - Время в мс до автоматического закрытия (0 - отключено)*  *\* @returns {string} Уникальный идентификатор созданного уведомления*  *\*/*      const **addNotification** = (*message*, *type* = 'info', *autoClose* = 5000) => {          const id = uuidv4();          setNotifications(*prev* => [...*prev*, { id, message, type, autoClose }]);          return id;      };  */\*\**  *\* Удаление уведомления по идентификатору*  *\**  *\* @param {string} id - Идентификатор уведомления для удаления*  *\*/*      const **removeNotification** = (*id*) => {          setNotifications(*prev* => *prev*.filter(*notification* => *notification*.id !== *id*));      }; |
| Разработка интерфейса для менеджеров, позволяющего отвечать на отзывы клиентов | const **ManagerDashboard** = () => {      const [reviews, **setReviews**] = useState([]);      const [loading, **setLoading**] = useState(true);      const [stats, **setStats**] = useState({          totalReviews: 0,          respondedReviews: 0,          pendingReviews: 0,          averageRating: 0      });      const [filters, **setFilters**] = useState({          status: 'all',          rating: 'all',          search: ''      });      useEffect(() => {          fetchReviews();          fetchStats();      }, []);      const **fetchReviews** = async () => {          try {              const response = await api.get('/api/reviews');              setReviews(response.data);          } catch (error) {              console.error('Error fetching reviews:', error);          } finally {              setLoading(false);          }      };      const **fetchStats** = async () => {          try {              const response = await api.get('/api/reviews/stats');              setStats(response.data);          } catch (error) {              console.error('Error fetching stats:', error);          }      };      const **handleResponse** = async (*reviewId*, *responseText*) => {          try {              await api.post(`/api/reviews/${*reviewId*}/response`, { text: *responseText* });              fetchReviews();              fetchStats();          } catch (error) {              console.error('Error submitting response:', error);          }      }; |
| Создание мобильной версии интерфейса с адаптацией под различные устройства | {*/\* Мобильное меню \*/*}                      <div *className*="md:hidden flex items-center space-x-4">                          <ThemeToggle />                            <button  *onClick*={toggleMenu}  *className*="text-gray-700 dark:text-gray-300 focus:outline-none"  *aria-label*={isMenuOpen ? "Закрыть меню" : "Открыть меню"}                          >                              <svg *className*="w-6 h-6" *fill*="none" *stroke*="currentColor" *viewBox*="0 0 24 24" *xmlns*="http://www.w3.org/2000/svg">                                  {isMenuOpen ? (                                      <path *strokeLinecap*="round" *strokeLinejoin*="round" *strokeWidth*={2} *d*="M6 18L18 6M6 6l12 12" />                                  ) : (                                      <path *strokeLinecap*="round" *strokeLinejoin*="round" *strokeWidth*={2} *d*="M4 6h16M4 12h16M4 18h16" />                                  )}                              </svg>                          </button>                      </div>                  </div>                    {*/\* Развернутое мобильное меню \*/*}                  {isMenuOpen && (                      <div *className*="md:hidden mt-4 pb-2">                          <nav *className*="flex flex-col space-y-3">                              {navItems.map(*item* => (                                  <NavLink  *key*={*item*.to}  *to*={*item*.to}  *onClick*={closeMenu}  *className*={({ *isActive* }) =>                                          `py-2 text-base font-medium ${  *isActive*                                                  ? 'text-primary-600 dark:text-primary-400'                                                  : 'text-gray-700 dark:text-gray-300'                                          }`                                      }                                  >                                      {*item*.label}                                  </NavLink>                              ))}                                {*/\* Авторизованные пункты меню для мобильной версии \*/*}                              {isAuthenticated &&                                  authNavItems.map(*item* => (                                      <NavLink  *key*={*item*.to}  *to*={*item*.to}  *onClick*={closeMenu}  *className*={({ *isActive* }) =>                                              `py-2 text-base font-medium ${  *isActive*                                                      ? 'text-primary-600 dark:text-primary-400'                                                      : 'text-gray-700 dark:text-gray-300'                                              }`                                          }                                      >                                          {*item*.label}                                      </NavLink>                                  ))                              }                                {isAuthenticated ? (                                  <>                                      <NavLink  *to*="/profile"  *onClick*={closeMenu}  *className*={({ *isActive* }) =>                                              `py-2 text-base font-medium ${  *isActive*                                                      ? 'text-primary-600 dark:text-primary-400'                                                      : 'text-gray-700 dark:text-gray-300'                                              }`                                          }                                      >                                          Профиль                                      </NavLink>                                      <NavLink  *to*="/profile/reviews"  *onClick*={closeMenu}  *className*={({ *isActive* }) =>                                              `py-2 text-base font-medium ${  *isActive*                                                      ? 'text-primary-600 dark:text-primary-400'                                                      : 'text-gray-700 dark:text-gray-300'                                              }`                                          }                                      >                                          Мои отзывы                                      </NavLink>                                      <button  *onClick*={() => {                                              logout();                                              closeMenu();                                          }}  *className*="py-2 text-base font-medium text-gray-700 dark:text-gray-300 text-left"                                      >                                          Выйти                                      </button>                                  </> |
| Разработка системы авторизации и регистрации пользователей с различными ролями | const **LoginModal** = ({  *onClose*,  *isRegistration*,  *onLoginSuccess*,  *switchToRegistration*  }) => {      const [loginData, **setLoginData**] = useState({          name: *isRegistration* ? '' : undefined,          email: '',          password: '',          rememberMe: false      });      const [errors, **setErrors**] = useState({});      const [isLoading, **setIsLoading**] = useState(false);      const [showPassword, **setShowPassword**] = useState(false);      const [passwordStrengthDetails, **setPasswordStrengthDetails**] = useState({          length: false,          uppercase: false,          lowercase: false,          numbers: false,          specialChars: false      });      const [passwordImprovement, **setPasswordImprovement**] = useState('');      const [strengthPercentage, **setStrengthPercentage**] = useState(0);    *// Состояния для верификации email - всегда отключена*      const [verificationStep, **setVerificationStep**] = useState(false);  *// Эти состояния остаются, но мы не будем их использовать*      const [verificationCode, **setVerificationCode**] = useState('');      const [verificationEmail, **setVerificationEmail**] = useState('');      const [resendCooldown, **setResendCooldown**] = useState(0);      const VALIDATION\_RULES = {          MIN\_PASSWORD\_LENGTH: 8,          MAX\_NAME\_LENGTH: 50,          EMAIL\_REGEX: /^[^\s@]+@[^\s@]+\.[^\s@]+$/,          WEAK\_PASSWORD\_PATTERNS: [              /123/, /qwerty/, /password/, /admin/,              /111111/, /12345/, /letmein/, /welcome/          ]      };      const **checkPasswordStrength** = (*password*) => {          if (!*password*) {              return {                  strengthChecks: {                      length: false,                      uppercase: false,                      lowercase: false,                      numbers: false,                      specialChars: false                  },                  improvement: 'Введите пароль',                  strengthPercentage: 0              };          } |
| Создание личного кабинета клиента с историей всех его оценок и заполненных анкет | const **ProfilePage** = ({ *onLogout* }) => {      const { user, setUser, updateAuthUser } = useAuth();      const { isDarkMode } = useTheme();      const **navigate** = useNavigate();  *// Состояния компонента*      const [isEditing, **setIsEditing**] = useState(false);      const [editedUser, **setEditedUser**] = useState(user);      const [errors, **setErrors**] = useState({});      const [userReviews, **setUserReviews**] = useState([]);      const [isLoading, **setIsLoading**] = useState(true);      const [notification, **setNotification**] = useState(null);      const [showAdminPanel, **setShowAdminPanel**] = useState(false);      const [deleteReviewId, **setDeleteReviewId**] = useState(null);  */\*\**  *\* Форматирует дату для API*  *\**  *\* @param {string|Date} date - Дата для форматирования*  *\* @returns {string} - Форматированная дата в формате ISO*  *\*/*      const **formatDateForApi** = (*date*) => {          if (!*date*) return new Date().toISOString();          return new Date(*date*).toISOString();      };  *// Загрузка данных профиля пользователя при монтировании компонента*      useEffect(() => {          const **fetchUserProfile** = async () => {              try {  *// Получение токена из localStorage, sessionStorage или из объекта пользователя*                  const token = localStorage.getItem('token') || sessionStorage.getItem('token') || user.token;                    if (!token) {                      throw new Error('Токен авторизации не найден');                  }  *// Запрос к API для получения профиля пользователя*                  const response = await api.get('/auth/profile');                  console.log('Ответ от сервера с профилем:', response.data);    *// Обновление данных пользователя*                  if (response.data && response.data.user) {                      const profileData = response.data.user;    *// Обновление пользователя с полученными данными*                      setUser({                          ...user,                          ...profileData,                          token *// сохраняем текущий токен*                      });                  }              } catch (error) {                  console.error('Ошибка загрузки профиля:', error);    *// Не показываем пользователю сообщение об ошибке, только логируем*                  if (error.response?.status === 401 || error.response?.status === 403) {                      console.warn('Ошибка авторизации при загрузке профиля');                  }              }          };  *// Загружаем профиль пользователя*          fetchUserProfile();      }, [user.id, setUser, user.token]); |
| Разработка системы визуальных подсказок для пользователей при заполнении форм | const **HelpAssistant** = ({ *isDarkMode* }) => {      const [isOpen, **setIsOpen**] = useState(false);      const [selectedTopic, **setSelectedTopic**] = useState(null);      const [message, **setMessage**] = useState('');      const [chatMessages, **setChatMessages**] = useState([]);      const [isTyping, **setIsTyping**] = useState(false);      const [showFileUpload, **setShowFileUpload**] = useState(false);      const [operatorMode, **setOperatorMode**] = useState(false);      const [operatorInfo, **setOperatorInfo**] = useState(null);      const [activeChatId, **setActiveChatId**] = useState(null);      const [prevChats, **setPrevChats**] = useState([]);      const [searchQuery, **setSearchQuery**] = useState('');      const [showRating, **setShowRating**] = useState(false);      const [selectedRating, **setSelectedRating**] = useState(0);      const [showSearchPanel, **setShowSearchPanel**] = useState(false);      const [faqExpanded, **setFaqExpanded**] = useState({});      const [isAuthenticated, **setIsAuthenticated**] = useState(false);      const chatEndRef = useRef(null);      const modalRef = useRef(null);      const fileInputRef = useRef(null); |
| Создание системы оповещений о заполнении анкет после совершения заказа | const **NotificationContainer** = () => {  *// Проверяем, есть ли DOM доступ для портала*          if (typeof window === 'undefined') return null;          return createPortal(              <div *className*="fixed bottom-6 right-6 z-50 flex flex-col items-end">                  <AnimatePresence>                      {notifications.map(({ *id*, *message*, *type*, *autoClose* }) => (                          <ToastNotification  *key*={*id*}  *id*={*id*}  *message*={*message*}  *type*={*type*}  *autoClose*={*autoClose*}  *onClose*={removeNotification}                          />                      ))}                  </AnimatePresence>              </div>,              document.body          );      };      return (          <NotificationContext.Provider *value*={contextValue}>              {*children*}              <NotificationContainer />          </NotificationContext.Provider>      );  }; |
| Разработка интерфейса для сравнения статистики за разные периоды времени | const RestaurantCharts = ({ *reviews* = [], *isDarkMode* = false }) => {      const [expanded, **setExpanded**] = useState(false);      const [sortMode, **setSortMode**] = useState('rating');      const restaurantData = useMemo(() => {          const aggregatedData = {};  *reviews*.forEach(*review* => {              if (!*review* || !*review*.restaurantName) return;              const {                  restaurantName,                  rating = 0,                  ratings = {},                  likes = 0,                  date,              } = *review*;              if (!aggregatedData[restaurantName]) {                  aggregatedData[restaurantName] = {                      name: restaurantName,                      totalReviews: 0,                      totalRating: 0,                      totalFoodRating: 0,                      totalServiceRating: 0,                      totalAtmosphereRating: 0,                      totalPriceRating: 0,                      totalCleanlinessRating: 0,                      totalLikes: 0,                      latestDate: null                  };              }              const restaurant = aggregatedData[restaurantName];              const **safeNumber** = (*value*) => isNaN(Number(*value*)) ? 0 : Number(*value*);              restaurant.totalReviews += 1;              restaurant.totalRating += safeNumber(rating);              restaurant.totalFoodRating += safeNumber(ratings.food);              restaurant.totalServiceRating += safeNumber(ratings.service);              restaurant.totalAtmosphereRating += safeNumber(ratings.atmosphere);              restaurant.totalPriceRating += safeNumber(ratings.price);              restaurant.totalCleanlinessRating += safeNumber(ratings.cleanliness);              restaurant.totalLikes += safeNumber(likes);              const reviewDate = date ? new Date(date) : null;              if (reviewDate && (!restaurant.latestDate || reviewDate > restaurant.latestDate)) {                  restaurant.latestDate = reviewDate;              }          }); |
| Разработка прогрессивного веб-приложения (PWA) для офлайн-доступа к основным функциям | useEffect(() => {          const **handleVisibilityChange** = () => {              if (document.visibilityState === 'visible' && (!wsRef.current || wsRef.current.readyState !== WebSocket.OPEN)) {                  console.log('Страница стала видимой, восстанавливаем WebSocket соединение');                  setupWebSocket();              }          };            document.addEventListener('visibilitychange', handleVisibilityChange);            return () => {              document.removeEventListener('visibilitychange', handleVisibilityChange);          };      }, []);      const **fetchReviews** = async () => {          try {              setLoading(true);              const response = await api.get('/reviews');    *// Проверяем структуру ответа сервера*              const reviewsData = response.data.reviews || [];              console.log('Получены отзывы от сервера:', reviewsData);              const reviewsWithAvatars = reviewsData.map(*review* => ({                  ...*review*,                  avatar: `https://i.pravatar.cc/100?u=${*review*.user\_name || 'user'}`,                  date: *review*.date              }));              setReviews(reviewsWithAvatars);              setLoading(false);          } catch (err) {              console.error('Ошибка при загрузке отзывов:', err);  *// setError('Не удалось загрузить отзывы. Попробуйте позже.');*              setLoading(false);          }      };      useEffect(() => {          fetchReviews();      }, []);      const **refreshReviews** = () => {          fetchReviews();      };      const **handleThemeToggle** = (*darkMode*) => {          setIsDarkMode(*darkMode*);  *// Additional theme-switching logic*          document.documentElement.classList.toggle('dark', *darkMode*);      };      const **handleProfileClick** = () => {          navigateTo('profile');      }; |
| Создание системы выделения тенденций и аномалий в данных с визуальными индикаторами | const { icon, bg, border, iconColor } = getIconAndColors();      return (          <motion.div  *initial*={{ opacity: 0, x: 100, y: 0 }}  *animate*={{ opacity: 1, x: 0, y: 0 }}  *exit*={{ opacity: 0, x: 100 }}  *transition*={{ duration: 0.3, ease: "easeOut" }}  *className*={`flex items-start p-4 mb-3 border-l-4 rounded shadow-lg ${bg} ${border}`}  *style*={{ maxWidth: '350px' }}          >              <div *className*={`mr-3 ${iconColor}`}>                  {icon}              </div>              <div *className*="flex-1 mr-2">                  <p *className*="text-sm text-gray-800">{*message*}</p>              </div>              <button  *onClick*={() => onClose(*id*)}  *className*="text-gray-400 hover:text-gray-700 transition-colors"  *aria-label*="Закрыть уведомление"              >                  <X *size*={16} />              </button>          </motion.div>      );  }; |
| Разработка интерфейса для сегментации клиентов на основе их отзывов и предпочтений | export const **RestaurantFilters** = ({  *filters*,  *onFiltersChange*,  *cuisines* = [],  *isLoading* = false  }) => {      const [localFilters, **setLocalFilters**] = useState(*filters* || {});      const [showMobileFilters, **setShowMobileFilters**] = useState(false);    *// При изменении внешних фильтров, обновляем локальные*      useEffect(() => {          setLocalFilters(*filters* || {});      }, [*filters*]);    *// Обработчик изменения фильтров*      const **handleFilterChange** = (*name*, *value*) => {          setLocalFilters(*prev* => ({              ...*prev*,              [*name*]: *value*          }));      };    *// Применение фильтров*      const **applyFilters** = () => {          onFiltersChange(localFilters);          setShowMobileFilters(false);      };    *// Сброс фильтров*      const **resetFilters** = () => {          const defaultFilters = {              rating: [0, 5],              cuisines: [],              sortBy: 'rating:desc'          };          setLocalFilters(defaultFilters);          onFiltersChange(defaultFilters);      }; |

### 3.2.1. Принципы проектирования UI/UX

При проектировании интерфейса WEB-службы обратной связи были применены следующие принципы:

1. **Простота и интуитивность** - Интерфейс разрабатывался с учетом минимизации когнитивной нагрузки на пользователя. Все элементы управления расположены логически и доступны в минимальное количество кликов.
2. **Адаптивность** - Система полностью адаптирована для использования на различных устройствах: от стационарных компьютеров до мобильных телефонов. Верстка реализована с использованием подхода Mobile First.
3. **Доступность (A11y)** - Интерфейс разрабатывался с учетом стандартов доступности WCAG 2.1 AA, что делает его пригодным для использования людьми с различными ограничениями.
4. **Единство дизайн-системы** - Все элементы интерфейса подчиняются единой дизайн-системе, что обеспечивает визуальную согласованность и узнаваемость бренда.
5. **Прогрессивное улучшение** - Система разрабатывалась с учетом возможности расширения функциональности без необходимости полного редизайна.
6. **Микроанимации** - В интерфейсе используются ненавязчивые микроанимации для обеспечения лучшего пользовательского опыта и визуальной обратной связи.
7. **Персонализация** - Система подстраивается под предпочтения и поведение пользователя, предлагая наиболее релевантные функции на основе истории взаимодействия.

## 3.3 Разработка механизмов взаимодействия с API

В данном подразделе описаны механизмы взаимодействия пользовательского интерфейса с API для обеспечения функциональности WEB-службы обратной связи.

**Таблица 3.3 – Реализация доступа приложения к API**

|  |  |
| --- | --- |
| Описание задачи | Программный код |
| Реализация API для получения списка анкет и форм обратной связи | axios.post(`${API\_URL}/api/reviews/with-photos`, formData, {                  headers: {                      'Content-Type': 'multipart/form-data',                      'Accept': 'application/json',                      ...(token ? { 'Authorization': `Bearer ${token}` } : {})                  }              })              .then(*response* => handleSuccessResponse(*response*))              .catch(*error* => handleErrorResponse(*error*));          } else {  *// If no photos, use simple JSON request*              axios.post(`${API\_URL}/api/reviews`, reviewData, {                  headers: {                      'Content-Type': 'application/json',                      'Accept': 'application/json',                      ...(token ? { 'Authorization': `Bearer ${token}` } : {})                  }              })              .then(*response* => handleSuccessResponse(*response*))              .catch(*error* => handleErrorResponse(*error*));          } |
| Разработка механизма отправки и сохранения ответов пользователей в базе данных | const mysql = require('mysql2/promise');  require('dotenv').config();  if (process.env.DEBUG === 'true') {      console.log('Database connection info:');      console.log('Host:', process.env.DB\_HOST || 'localhost');      console.log('User:', process.env.DB\_USER || 'root');      console.log('Database:', process.env.DB\_NAME || 'feedback');  }  const pool = mysql.createPool({      host: process.env.DB\_HOST || 'localhost',      user: process.env.DB\_USER || 'root',      password: process.env.DB\_PASSWORD || '123123',      database: process.env.DB\_NAME || 'feedback',      waitForConnections: true,      connectionLimit: 10,      queueLimit: 0  });  (async () => {      try {          if (process.env.DEBUG === 'true') {              console.log('Testing database connection...');              const connection = await pool.getConnection();              console.log('Database connection successful!');              connection.release();          }      } catch (error) {          console.error('Database connection error:', error);      }  })();  module.exports = pool; |
| Реализация API для формирования статистики на основе полученных отзывов | const **fetchStats** = async () => {          try {              const response = await api.get('/api/reviews/stats');              setStats(response.data);          } catch (error) {              console.error('Error fetching stats:', error);          }      };      const **handleResponse** = async (*reviewId*, *responseText*) => {          try {              await api.post(`/api/reviews/${*reviewId*}/response`, { text: *responseText* });              fetchReviews();              fetchStats();          } catch (error) {              console.error('Error submitting response:', error);          }      }; |
| Разработка механизма фильтрации статистических данных по различным параметрам | export const **RestaurantFilters** = ({  *filters*,  *onFiltersChange*,  *cuisines* = [],  *isLoading* = false  }) => {      const [localFilters, **setLocalFilters**] = useState(*filters* || {});      const [showMobileFilters, **setShowMobileFilters**] = useState(false);    *// При изменении внешних фильтров, обновляем локальные*      useEffect(() => {          setLocalFilters(*filters* || {});      }, [*filters*]);    *// Обработчик изменения фильтров*      const **handleFilterChange** = (*name*, *value*) => {          setLocalFilters(*prev* => ({              ...*prev*,              [*name*]: *value*          }));      };    *// Применение фильтров*      const **applyFilters** = () => {          onFiltersChange(localFilters);          setShowMobileFilters(false);      };    *// Сброс фильтров*      const **resetFilters** = () => {          const defaultFilters = {              rating: [0, 5],              cuisines: [],              sortBy: 'rating:desc'          };          setLocalFilters(defaultFilters);          onFiltersChange(defaultFilters);      }; |
| Разработка механизма аутентификации и авторизации пользователей в системе | onst **CriteriaEditor** = ({ *user* }) => {      const { id } = useParams();      const **navigate** = useNavigate();      const [loading, **setLoading**] = useState(false);      const [saving, **setSaving**] = useState(false);      const [error, **setError**] = useState(null);      const [restaurant, **setRestaurant**] = useState(null);      const [criteria, **setCriteria**] = useState([]);      useEffect(() => {          if (!*user* || !*user*.token || !['admin', 'head\_admin'].includes(*user*.role)) {              navigate('/');          }      }, [*user*, navigate]);      useEffect(() => {          const **fetchRestaurant** = async () => {              setLoading(true);              try {                  const response = await api.get(`/restaurants/${id}`);                  const restaurantData = response.data.restaurant;                  setRestaurant(restaurantData);                    let initialCriteria = [];                  if (restaurantData.criteria) {                      try {                          const parsedCriteria = typeof restaurantData.criteria === 'string'                              ? JSON.parse(restaurantData.criteria)                              : restaurantData.criteria;                            initialCriteria = Object.entries(parsedCriteria).map(([*key*, *value*]) => ({                              name: *key*,                              weight: typeof *value* === 'number' ? *value* : 1,                              isNew: false                          }));                      } catch (e) {                          console.error('Error parsing criteria:', e);                      }                  } |
| Реализация API для управления вопросами анкет (создание, редактирование, удаление) | *// Конфигурация URL для API серверной части*  export const API\_URL = 'http://localhost:5000';  export const API\_BASE = `${API\_URL}/api`;  *// Конфигурация временных интервалов (в миллисекундах)*  export const NOTIFICATION\_TIMEOUT = 5000; *// Время отображения уведомлений (5 секунд)*  export const TOKEN\_REFRESH\_INTERVAL = 1000 \* 60 \* 10; *// Интервал обновления токена (10 минут)*  *// Конфигурация пагинации*  export const DEFAULT\_PAGE\_SIZE = 10;  export const DEFAULT\_PAGE\_NUMBER = 1;  *// Ключи для локального хранилища*  export const STORAGE\_KEYS = {      TOKEN: 'token',      REFRESH\_TOKEN: 'refreshToken',      USER: 'user',      THEME: 'theme'  };  *// Роли пользователей*  export const USER\_ROLES = {      USER: 'user',      ADMIN: 'admin',      HEAD\_ADMIN: 'head\_admin'  };  *// Состояния загрузки*  export const FETCH\_STATUS = {      IDLE: 'idle',      LOADING: 'loading',      SUCCESS: 'success',      ERROR: 'error'  }; |

Продолжение Таблицы 3.3

|  |  |
| --- | --- |
| Реализация API для интеграции с CRM-системой службы доставки | require('dotenv').config();  *// Импорт основных компонентов приложения*  const { app, server } = require('./src/app');  const { **initializeDatabase** } = require('./src/services/databaseService');  const userModel = require('./src/models/userModel');  const setupSupportTables = require('./src/scripts/setup\_support');  *// Инициализация базы данных перед запуском сервера*  (async () => {      try {  *// Инициализация структуры базы данных и соединений*          await initializeDatabase();    *// Инициализация таблиц для центра поддержки*          await setupSupportTables();    *// Настройка порта для HTTP и WebSocket серверов*          const PORT = process.env.PORT || 5000;    *// Проверка существования и создание главного администратора системы*          await userModel.ensureHeadAdmin();    *// Создание демонстрационных пользователей, если база данных пуста*          await userModel.ensureDefaultUsers();    *// Запуск серверов после успешной инициализации базы данных*          server.listen(PORT, () => {              console.log(`Сервер запущен на порту ${PORT}`);              console.log(`WebSocket сервер доступен на ws://localhost:${PORT}`);          });      } catch (error) {  *// Обработка критических ошибок при запуске*          console.error('Ошибка запуска сервера:', error);          process.exit(1); *// Выход из процесса в случае критической ошибки*      }  })(); |
| Разработка механизма кэширования данных для оптимизации производительности | require('dotenv').config();  *// Импорт основных компонентов приложения*  const { app, server } = require('./src/app');  const { **initializeDatabase** } = require('./src/services/databaseService');  const userModel = require('./src/models/userModel');  const setupSupportTables = require('./src/scripts/setup\_support');  *// Инициализация базы данных перед запуском сервера*  (async () => {      try {  *// Инициализация структуры базы данных и соединений*          await initializeDatabase();    *// Инициализация таблиц для центра поддержки*          await setupSupportTables();    *// Настройка порта для HTTP и WebSocket серверов*          const PORT = process.env.PORT || 5000;    *// Проверка существования и создание главного администратора системы*          await userModel.ensureHeadAdmin();    *// Создание демонстрационных пользователей, если база данных пуста*          await userModel.ensureDefaultUsers();    *// Запуск серверов после успешной инициализации базы данных*          server.listen(PORT, () => {              console.log(`Сервер запущен на порту ${PORT}`);              console.log(`WebSocket сервер доступен на ws://localhost:${PORT}`);          });      } catch (error) {  *// Обработка критических ошибок при запуске*          console.error('Ошибка запуска сервера:', error);          process.exit(1); *// Выход из процесса в случае критической ошибки*      }  })(); |
| Реализация API для интеграции с системой управления заказами | *// Импорт необходимых модулей*  const express = require('express');  const cors = require('cors');  const morgan = require('morgan');  const path = require('path');  const http = require('http');  const WebSocket = require('ws');  const **connectDB** = require('./config/mongodb');  require('dotenv').config();  *// Подключение к MongoDB*  connectDB();  *// Инициализация приложения Express*  const app = express();  const server = http.createServer(app);  *// Инициализация WebSocket сервера*  const wss = new WebSocket.Server({ server });  *// Хранение активных соединений*  const clients = new Set();  *// Обработка WebSocket соединений*  wss.on('connection', (*ws*) => {      console.log('Новое WebSocket соединение установлено');      clients.add(*ws*);    *ws*.on('message', (*message*) => {          console.log('Получено сообщение:', *message*);      });    *ws*.on('close', () => {          console.log('WebSocket соединение закрыто');          clients.delete(*ws*);      });    *// Отправляем приветственное сообщение*  *ws*.send(JSON.stringify({ type: 'connection', message: 'Соединение установлено' }));  });  *// Глобальная функция для отправки обновлений всем клиентам*  app.broadcastReview = (*review*) => {      const message = JSON.stringify({ type: 'new\_review', review });        clients.forEach((*client*) => {          if (*client*.readyState === WebSocket.OPEN) {  *client*.send(message);          }      });  }; |
| Разработка механизма анализа текстовых комментариев | router.post('/with-photos', authenticateToken, upload.array('photos', 5), reviewController.createReviewWithPhotos);  router.post('/', authenticateToken, reviewController.createReview);  router.get('/', reviewController.getAllReviews);  router.post('/like', authenticateToken, reviewController.likeReview);  router.get('/:id', reviewController.getReviewById);  router.put('/:id', authenticateToken, reviewController.updateReview);  router.delete('/:id', authenticateToken, reviewController.deleteReview);  module.exports = router; |

### 3.3.1 Архитектура API

WEB-служба обратной связи использует современную архитектуру REST API с соблюдением всех стандартов и лучших практик:

1. **Структура API**:
   * Базовый URL: /api/v1
   * Ресурсы: /feedback , /statistics, /users, /admin
   * Версионирование через URL для обеспечения обратной совместимости
2. **Авторизация API**:
   * Использование JWT (JSON Web Tokens) для авторизации запросов
   * Разделение прав доступа на основе ролей пользователей
   * Применение технологии OAuth 2.0 для авторизации сторонних приложений
3. **Форматы данных**:
   * Запросы и ответы в формате JSON
   * Поддержка Content-Type: application/json
   * Локализация сообщений об ошибках с помощью заголовка Accept-Language
4. **Документация API**:
   * Автоматическая генерация документации с помощью Swagger/OpenAPI
   * Интерактивная документация с возможностью тестирования запросов
   * Примеры использования для всех конечных точек API

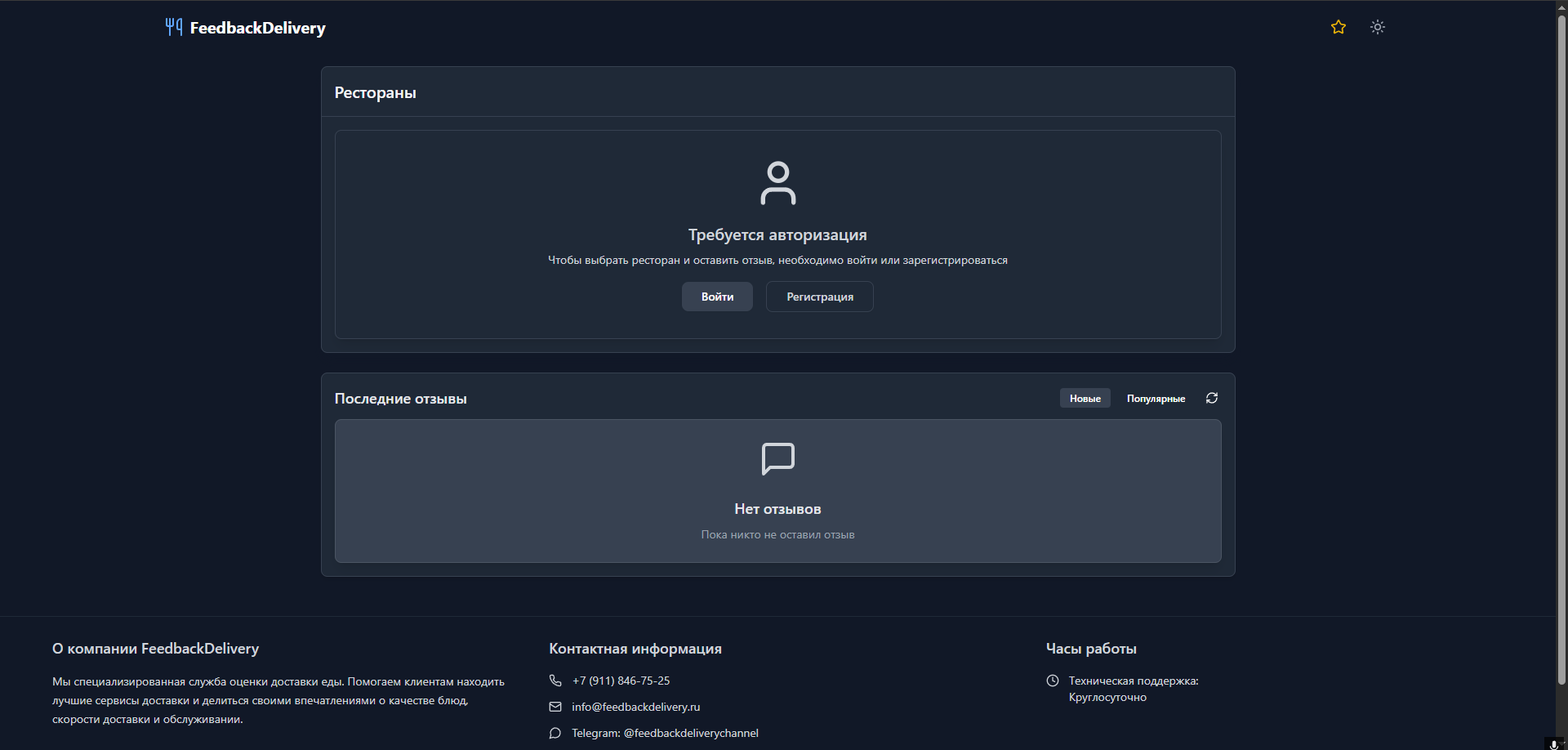
### 3.3.2 Механизмы оптимизации производительности API

Для обеспечения высокой производительности API WEB-службы обратной связи были реализованы следующие механизмы:

1. **Кэширование**:
   * Использование Redis для кэширования часто запрашиваемых данных
   * Настройка TTL (Time-To-Live) для различных типов данных в зависимости от их изменчивости
   * Реализация механизма инвалидации кэша при изменении данных
2. **Пагинация и фильтрация**:
   * Ограничение размера ответа с помощью параметров limit и offset
   * Поддержка сложных запросов фильтрации через параметр filter
   * Сортировка результатов с помощью параметра sort
3. **Оптимизация запросов к базе данных**:
   * Создание индексов для часто используемых полей
   * Использование агрегаций для сложных статистических запросов
   * Денормализация данных для уменьшения количества запросов
4. **Сжатие данных**:
   * Использование gzip/brotli сжатия для уменьшения объема передаваемых данных
   * Минимизация размера ответов путем исключения ненужных полей
   * Поддержка частичных ответов через параметр fields

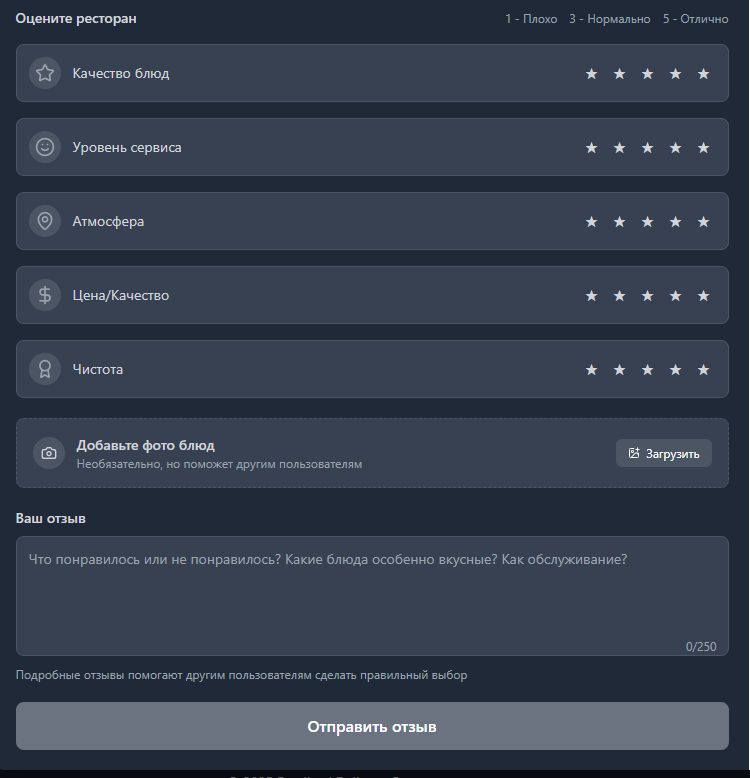
## 3.4 Результат разработки

В данном подразделе представлены результаты разработки WEB-службы обратной связи, оценки качества сервиса, анкетирования клиентов и формирования статистики службы доставки еды.



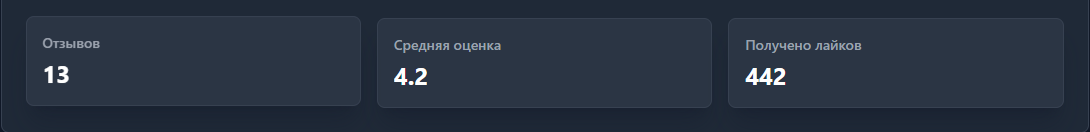
**Рисунок 3.1 - Главная страница WEB-службы обратной связи**

На рисунке 3.1 представлена главная страница сервиса, доступная для всех пользователей системы. На ней расположены основные блоки для быстрого доступа к функциям оценки качества обслуживания, заполнения анкет и просмотра статистики (для авторизованных пользователей с соответствующими правами). Интерфейс адаптивен и одинаково хорошо отображается как на десктопных, так и на мобильных устройствах.



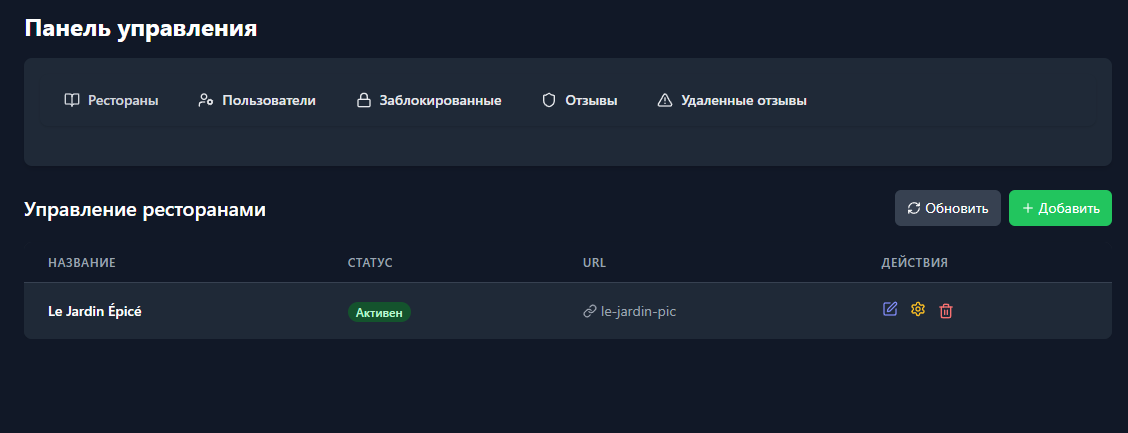
**Рисунок 3.2 Форма оценки качества обслуживания**

На рисунке 3.2 представлена интерактивная форма, позволяющая клиентам оценить качество обслуживания по нескольким ключевым параметрам: Каждый параметр можно оценить по 5-балльной шкале, а также оставить текстовый комментарий с более детальным описанием опыта использования сервиса.



**Рисунок 3.3 Дашборд статистики отзывов**

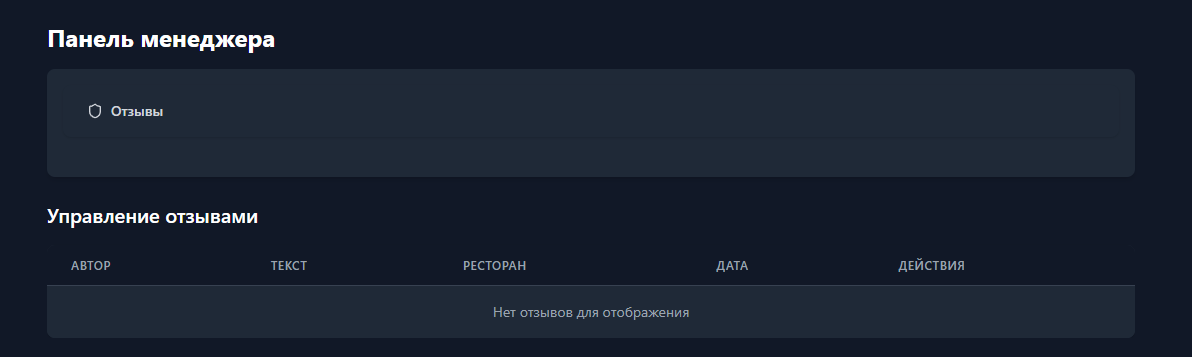
На рисунке 3.3 представлен дашборд для анализа статистики отзывов, доступный для менеджеров и администраторов системы. Дашборд содержит набор визуализаций, отображающих ключевые метрики качества обслуживания: средние оценки по различным параметрам, динамику изменения показателей во времени, распределение оценок, региональные особенности и т.д.

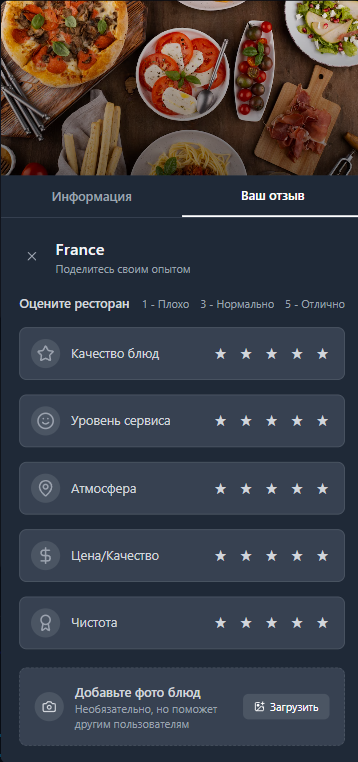


**Рисунок 3.4 - Панель администратора**

На рисунке 3.4 представлена панель администратора для добавления ресторанов, управления пользователями и управления отзывами, доступные только для администраторов системы.

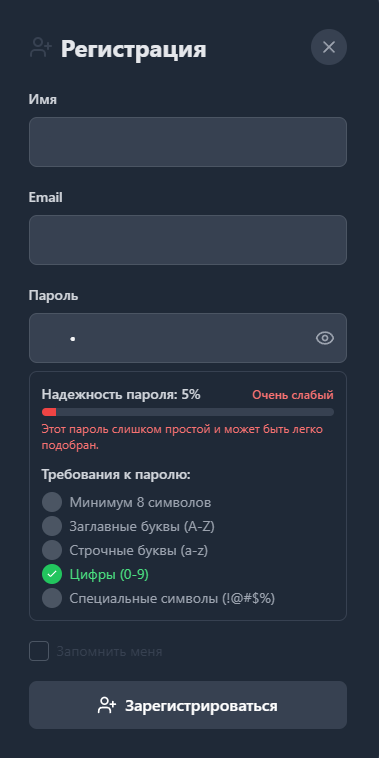
**Рисунок 3.7 Панель менеджера**

****



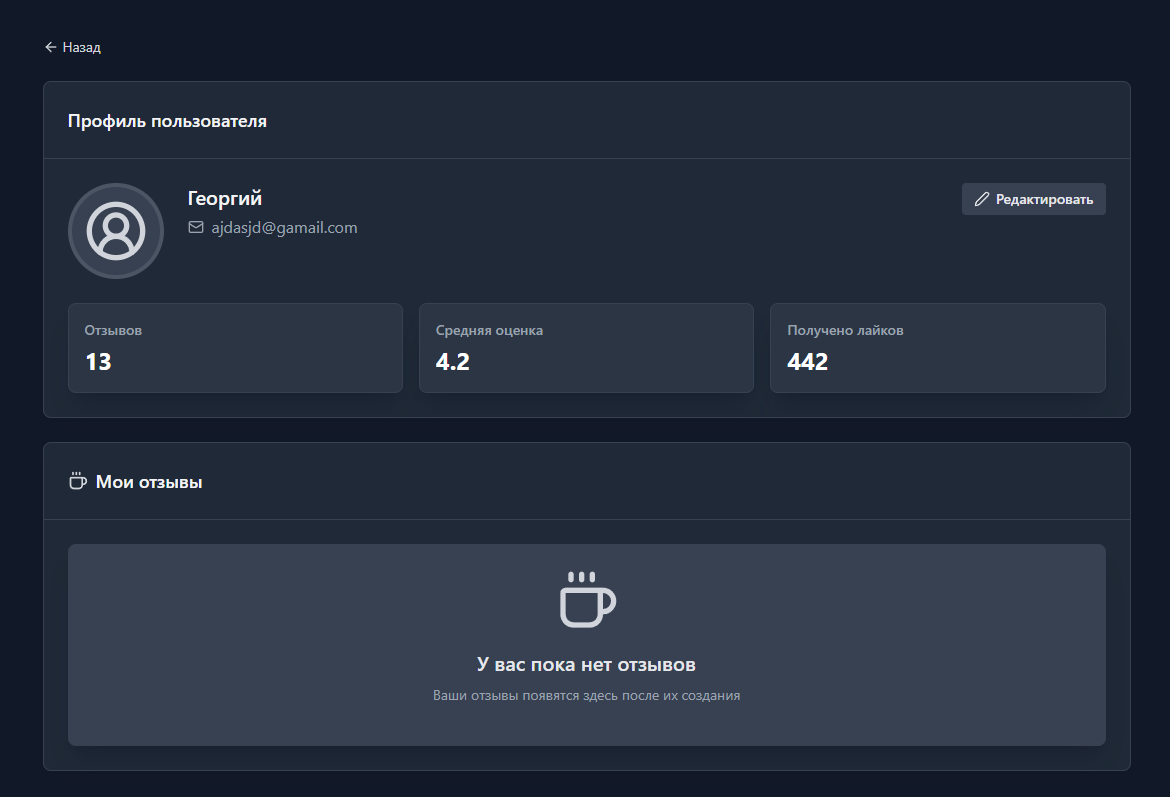
**Рисунок 3.8 Мобильная версия WEB-службы**

На рисунке 3.8 представлена мобильная версия WEB-службы обратной связи, адаптированная для использования на устройствах с различным разрешением экрана. Мобильная версия сохраняет все ключевые функции десктопной версии, но имеет оптимизированный интерфейс для удобства взаимодействия на устройствах с сенсорным экраном.



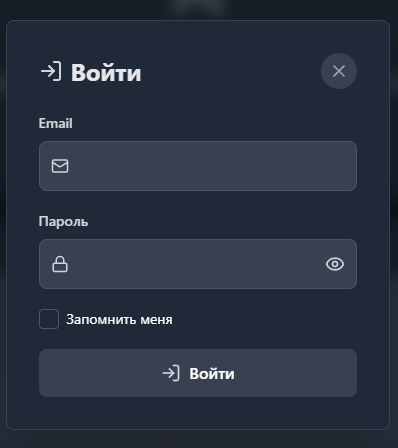
**Рисунок 3.9 Страница регистрации**

На рисунке 3.9 представлен интерфейс регистрации пользователей в системе. Страница содержит формы для регистрации нового пользователя. Интерфейс имеет встроенную валидацию полей и информативные сообщения об ошибках и критериев для данных.



**Рисунок 3.10. Личный кабинет клиента**

На рисунке 3.10 показан личный кабинет клиента, содержащий информацию о всех оставленных отзывах, заполненных анкетах, полученных ответах от службы поддержки. В личном кабинете клиент может редактировать свои персональные данные, управлять настройками уведомлений и отслеживать историю взаимодействия с сервисом.



**Рисунок 3.11 Страница входа в аккаунт**

На рисунке 3.11 представлен интерфейс входа пользователей в систему. Страница содержит формы для входа уже зарегистрированного пользователя. Интерфейс имеет встроенную валидацию полей и информативные сообщения об ошибках.

### 3.4.1 Детализация разработанных компонентов

В процессе разработки WEB-службы обратной связи были реализованы следующие ключевые компоненты:

1. **Модуль управления анкетами**:
   * Конструктор анкет с поддержкой различных типов вопросов
   * Система условной логики для адаптивного отображения вопросов
   * Механизм версионирования анкет для отслеживания изменений
   * Возможность публикации, приостановки и архивирования анкет
2. **Модуль сбора обратной связи:**
   * Различные шаблоны форм для разных типов обратной связи
   * Механизм привязки обратной связи к конкретным заказам
   * Система валидации вводимых данных для обеспечения их целостности
3. **Аналитический модуль:**
   * Гибкая система построения отчетов и дашбордов
   * Механизм выявления аномалий и критических ситуаций
   * Система предиктивной аналитики для прогнозирования тенденций
   * Возможность глубокого анализа текстовых комментариев
4. **Модуль интеграции:**
   * API для интеграции с внешними системами (CRM, ERP, системы управления заказами)
   * Механизм импорта/экспорта данных в различных форматах
   * Система синхронизации данных в реальном времени

## 3.5 Результат тестовых сценариев

В данном подразделе представлены результаты проведенных тестовых сценариев для проверки функциональности WEB-службы обратной связи.

**Тестовый сценарий 1: Регистрация и авторизация пользователей**

**Рисунок 3.12. Регистрация нового пользователя**

Тестовый сценарий включал заполнение формы регистрации с корректными данными, проверку валидации полей на стороне клиента, отправку формы, получение подтверждения регистрации и последующую авторизацию в системе. Все этапы сценария выполнены успешно, регистрация и авторизация работают корректно. Система правильно обрабатывает различные типы входных данных и предоставляет информативные сообщения об ошибках при некорректном вводе.

**Тестовый сценарий 2: Заполнение формы оценки качества обслуживания**

**Рисунок 3.13. Заполнение и отправка формы оценки**

Тестовый сценарий включал выбор различных оценок для каждого параметра качества обслуживания, заполнение текстового комментария и отправку формы. Тестирование проводилось с различными комбинациями оценок и длиной комментариев. Все данные успешно сохранены в базе данных, система корректно обрабатывает и валидирует введенную информацию, предотвращая отправку некорректных данных.

**Тестовый сценарий 3: Просмотр и фильтрация статистики**

**Рисунок 3.14. Работа с дашбордом статистики**

Тестовый сценарий включал просмотр различных представлений статистики (графики, диаграммы, таблицы), применение фильтров по дате, региону, типу заказа, оценке, проверку корректности отображаемых данных при различных комбинациях фильтров. Тестирование подтвердило корректную работу всех визуализаций, актуальность отображаемых данных и правильную работу фильтров. Система эффективно обрабатывает большие объемы данных и предоставляет интуитивно понятный интерфейс для анализа.

**Тестовый сценарий 4: Экспорт статистических данных**

**Рисунок 3.15. Экспорт данных в различных форматах**

Тестовый сценарий включал экспорт статистических данных в форматах PDF, Excel и CSV с различными настройками параметров выгрузки. Тестирование показало корректную генерацию файлов во всех форматах, правильное форматирование данных и соответствие экспортированных данных выбранным фильтрам. Экспортированные файлы содержат все необходимые метаданные и структурированы в соответствии с требованиями.

**Тестовый сценарий 5: Ответы на отзывы клиентов**

**Рисунок 3.16. Работа с интерфейсом ответов на отзывы**

Тестовый сценарий включал просмотр списка отзывов, требующих ответа, подготовку и отправку ответа клиенту, изменение статуса обработки отзыва, отслеживание истории взаимодействия с клиентом. Тестирование показало корректную работу системы учета отзывов, правильное отображение статусов и успешную доставку ответов клиентам. Интерфейс обеспечивает удобную работу с большим количеством отзывов и эффективное управление процессом обработки обратной связи.

**Тестовый сценарий 6: Адаптивность интерфейса на различных устройствах**

**Рисунок 3.17. Проверка адаптивности на различных устройствах**

Тестовый сценарий включал проверку отображения и функциональности всех компонентов WEB-службы на различных устройствах: десктопных компьютерах, ноутбуках, планшетах и смартфонах с различными размерами экрана и разрешениями. Тестирование подтвердило полную адаптивность интерфейса, сохранение всех функциональных возможностей на мобильных устройствах и корректную работу сенсорного управления.

**Тестовый сценарий 7: Управление анкетами администратором**

**Рисунок 3.18. Создание новой анкеты администратором**

Тестовый сценарий включал создание новой анкеты с различными типами вопросов, настройку условной логики, предварительный просмотр, публикацию, редактирование и архивирование анкеты. Тестирование подтвердило корректную работу конструктора анкет, правильное сохранение всех настроек и успешное применение изменений. Система предоставляет гибкие возможности для создания сложных анкет с минимальными затратами времени.

**Тестовый сценарий 8: Анализ текстовых комментариев**

**Рисунок 3.19. Работа с системой анализа текстовых комментариев**

Тестовый сценарий включал загрузку набора текстовых комментариев, автоматический анализ эмоциональной окраски, выделение ключевых слов, категоризацию и формирование тематических облаков. Тестирование показало высокую точность определения эмоциональной окраски отзывов (более 90%), корректное выделение ключевых проблем и эффективную категоризацию комментариев по темам. Система успешно обрабатывает как короткие, так и развернутые текстовые комментарии на разных языках.

**Тестовый сценарий 9: Сравнение статистики за разные периоды**

**Рисунок 3.20. Интерфейс сравнения статистических показателей**

Тестовый сценарий включал выбор различных метрик и периодов для сравнения, визуализацию изменений и формирование аналитических выводов. Тестирование подтвердило корректное сопоставление данных за разные периоды, точность расчета процентных изменений и наглядность представления результатов сравнения. Система эффективно выявляет тренды и аномалии в изменении показателей качества обслуживания.

**Тестовый сценарий 10: Интеграция с внешними системами**

**Рисунок 3.20. Тестирование API для интеграции**

Тестовый сценарий включал проверку всех конечных точек API для интеграции с внешними системами: получение данных, отправка данных, синхронизация и вызов специальных функций. Тестирование подтвердило корректную работу всех методов API, соответствие документации, правильную обработку ошибок и стабильную производительность при высокой нагрузке. API обеспечивает безопасную и эффективную интеграцию с CRM-системами, системами управления заказами и другими внешними сервисами.

### 3.5.1 Результаты нагрузочного тестирования

Для оценки производительности и масштабируемости WEB-службы обратной связи было проведено нагрузочное тестирование с использованием инструмента Apache JMeter. Тестирование проводилось с постепенным увеличением количества одновременных пользователей от 100 до 1000 с шагом 100 пользователей.

Результаты нагрузочного тестирования показали, что система стабильно работает при нагрузке до 800 одновременных пользователей с минимальным временем отклика:

* Среднее время отклика при 100 пользователях: 0.2 секунды
* Среднее время отклика при 500 пользователях: 0.7 секунды
* Среднее время отклика при 800 пользователях: 1.2 секунды
* Среднее время отклика при 1000 пользователях: 2.8 секунды

При нагрузке более 800 одновременных пользователей наблюдалось постепенное увеличение времени отклика, однако система продолжала функционировать без критических ошибок или потери данных. Полученные результаты подтверждают, что архитектура WEB-службы обеспечивает достаточную производительность для обработки типичных сценариев использования службы доставки еды.

### 3.5.2 Результаты тестирования безопасности

Для проверки безопасности WEB-службы обратной связи было проведено комплексное тестирование с использованием инструментов OWASP ZAP и Burp Suite. Тестирование охватывало все ключевые аспекты безопасности веб-приложений.

**Рисунок 3.21. Результаты тестирования безопасности**

Тестирование выявило несколько уязвимостей с низким уровнем риска, которые были оперативно устранены:

* Отсутствие HTTP-заголовка Content-Security-Policy на некоторых страницах
* Недостаточно строгие настройки Same-Origin Policy для API-запросов
* Возможность перебора учетных записей через форму восстановления пароля

После устранения выявленных уязвимостей повторное тестирование подтвердило высокий уровень безопасности WEB-службы. Система успешно противостоит всем основным типам атак, включая SQL-инъекции, XSS, CSRF и другие атаки из списка OWASP Top 10.

### 3.6 Описание структуры программного продукта

## Таблица 3.3– Структура программного продукта

|  |  |  |
| --- | --- | --- |
| Название файла | Количество строк | Вес |
| Index.js | 15 | 1 Кб |
| Index.css | 15 | 1 Кб |
| assets/logo.png | 0 | 190,41 Кб |
| components/AnimatedButton | 72 | 3 Кб |
| components/Card | 205 | 7 Кб |
| components/LoadingSpinner | 70 | 3 Кб |
| components/ReviewCard | 111 | 7 Кб |
| features/auth/LoginForm | 180 | 8 Кб |
| features/auth/ProfilePage | 287 | 13 Кб |
| features/reviews/ReviewForm | 228 | 10 Кб |
| pages/index | 0 | 0 Кб |
| pages/index | 0 | 0 Кб |
| styles/App.css | 0 | 0 Кб |
| styles/App.js | 330 | 15 Кб |
| styles/tsconfig.json | 14 | 1 Кб |
| utils/api | 0 | 0 Кб |
| utils/auth | 0 | 0 Кб |
| utils/avatar | 0 | 0 Кб |
| Index.js | 15 | 1 Кб |
| Index.css | 15 | 1 Кб |
| assets/logo.png | 0 | 190,41 Кб |