# Глава 3. Разработка интернет-витрины “Дому мебель”

## 3.1. Подход к разработке и организация процесса

### 3.1.1. Описание ролей участников проекта

В данном разделе описывается организационная структура проекта. Отличительной особенностью разработки интернет-витрины “Дому мебель” является то, что проект выполнялся индивидуально одним разработчиком. При выполнении проекта автор самостоятельно выполнял все необходимые роли:

* Разработчик серверной части - разработка бэкенда на Node.js и Express.js, создание REST API, работа с базой данных PostgreSQL
* Разработчик клиентской части - создание интерфейса пользователя с помощью React, реализация компонентного подхода и адаптивного дизайна
* Проектировщик базы данных - проектирование и реализация структуры базы данных, оптимизация запросов
* Дизайнер интерфейса - разработка UI/UX решений, работа над эргономикой пользовательского интерфейса
* Тестировщик - разработка и проведение тестов различного уровня, от модульных до интеграционных

Такой подход, хотя и требует от разработчика универсальности и наличия широкого спектра компетенций, позволяет обеспечить целостность видения проекта и отсутствие коммуникационных барьеров между различными этапами разработки.

### 3.1.2. Описание методологии разработки

В проекте была применена гибкая методология разработки Agile с элементами Scrum: - Структурирование работы с использованием пользовательских историй - Итерационный подход к разработке - Регулярная оценка и корректировка планов - Использование Kanban-доски для визуализации и контроля работы над задачами

### 3.1.3. Описание проектного инструментария

#### 3.1.3.1. Работа с системой контроля версий

* Выбор Git как системы контроля версий проекта
* Особенности создания ветвей разработки:
  + Ветка main для стабильных релизов
  + Ветка develop для текущей разработки
  + Тематические ветки feature/[название-функции] для отдельных функций
  + Ветки hotfix/[описание-проблемы] для срочных исправлений
* Процесс слияния веток и формирования релизов
* Правила именования коммитов и ведения истории разработки

### 3.1.4. Регламент проведения инспекций кода

* Чек-лист проверки качества кода:
  + Соответствие стандартам кодирования
  + Отсутствие дублирования кода
  + Соблюдение принципов SOLID
  + Корректность обработки ошибок
  + Наличие комментариев в сложных участках кода
  + Оптимальность алгоритмов и запросов к базе данных

## 3.2. Программное обеспечение проекта

### 3.2.1. Метрики качества и эффективности

* Метрики эффективности процесса разработки:
  + Скорость разработки (Story Points/неделю)
  + Время реализации отдельных функций
  + Количество выявленных и исправленных ошибок
* Метрики качества кода:
  + Сложность кода (цикломатическая сложность)
  + Покрытие тестами
  + Количество дублирующегося кода
  + Уровень связности компонентов

### 3.2.2. Требования и пользовательские истории

#### 3.2.2.1. Основные требования системы

| № требования | Описание требования | Приоритет |
| --- | --- | --- |
| REQ-001 | Система должна обеспечивать просмотр каталога мебели с фильтрацией и сортировкой | Высокий |
| REQ-002 | Система должна предоставлять детальную информацию о каждом товаре | Высокий |
| REQ-003 | Система должна иметь административный интерфейс для управления товарами | Высокий |
| REQ-004 | Система должна быть доступна и корректно работать на мобильных устройствах | Средний |
| REQ-005 | Система должна обеспечивать безопасное хранение данных пользователей | Высокий |

#### 3.2.2.2. Пользовательские истории

| № истории | Формулировка истории | Стоимость (ч/ч) | Номер тестового сценария |
| --- | --- | --- | --- |
| US-001 | КАК покупатель Я ХОЧУ просматривать каталог товаров по категориям ДЛЯ ТОГО ЧТОБЫ быстро находить нужную мебель | 8 | TS-001 |
| US-002 | КАК покупатель Я ХОЧУ видеть детальное описание товара с фотографиями ДЛЯ ТОГО ЧТОБЫ получить полное представление о мебели | 12 | TS-002 |
| US-003 | КАК менеджер магазина Я ХОЧУ добавлять новые товары в каталог ДЛЯ ТОГО ЧТОБЫ предоставлять актуальную информацию клиентам | 16 | TS-003 |
| US-004 | КАК менеджер магазина Я ХОЧУ редактировать информацию о существующих товарах ДЛЯ ТОГО ЧТОБЫ поддерживать актуальность данных | 10 | TS-004 |
| US-005 | КАК покупатель Я ХОЧУ иметь возможность фильтровать товары по параметрам ДЛЯ ТОГО ЧТОБЫ быстро находить мебель по моим требованиям | 14 | TS-005 |

### 3.2.3. Тестовые сценарии

| № теста | Описание тестового сценария | Номер покрываемых историй/требований |
| --- | --- | --- |
| TS-001 | Проверка отображения и навигации по категориям товаров | US-001, REQ-001 |
| TS-002 | Проверка отображения детальной информации о товаре | US-002, REQ-002 |
| TS-003 | Проверка функционала добавления новых товаров через админ-панель | US-003, REQ-003 |
| TS-004 | Проверка функционала редактирования существующих товаров | US-004, REQ-003 |
| TS-005 | Проверка работы фильтров и сортировки в каталоге товаров | US-005, REQ-001 |

### 3.2.4. Матрица покрытия требований тестами

| Требования/Тесты | TS-001 | TS-002 | TS-003 | TS-004 | TS-005 |
| --- | --- | --- | --- | --- | --- |
| REQ-001 | X |  |  |  | X |
| REQ-002 |  | X |  |  |  |
| REQ-003 |  |  | X | X |  |
| REQ-004 | X | X | X | X | X |
| REQ-005 |  |  | X | X |  |

### 3.2.5. Перечень задач проекта

| № истории | № задачи | Краткое описание задачи | Подробное описание задачи |
| --- | --- | --- | --- |
| US-001 | T-001-1 | Разработка структуры API для получения категорий товаров | Создание эндпоинтов API для получения списка категорий и подкатегорий мебели |
| US-001 | T-001-2 | Разработка компонента навигации по категориям | Создание React-компонента для отображения дерева категорий |
| US-002 | T-002-1 | Разработка API для получения детальной информации о товаре | Создание эндпоинта API для получения информации о конкретном товаре по ID |
| US-002 | T-002-2 | Разработка компонента детального просмотра товара | Создание React-компонента для отображения подробной информации о товаре |
| US-003 | T-003-1 | Разработка API для добавления товаров | Создание эндпоинта API для добавления нового товара в базу данных |
| US-003 | T-003-2 | Разработка интерфейса добавления товара | Создание форм и компонентов для добавления товара через административный интерфейс |

### 3.2.6. Рекомендации по кодированию

* Стандарты именования: camelCase для переменных и функций, PascalCase для компонентов React и классов
* Форматирование кода: использование Prettier с конфигурацией для отступов в 2 пробела
* Структура компонентов: разделение на представление и логику
* Работа с состоянием: использование Redux для глобального состояния, React hooks для локального
* Обработка ошибок: централизованная обработка с информативными сообщениями для пользователя
* Комментирование кода: JSDoc для документирования функций и компонентов
* Типизация данных: использование TypeScript для статической типизации

## 3.3. Реализация серверной части

### 3.3.1. Структура серверного приложения

* Описание архитектуры серверного приложения на Node.js и Express.js
* Обзор ключевых модулей и их взаимодействия
* Организация маршрутов API и контроллеров
* Реализация бизнес-логики и взаимодействия с базой данных

### 3.3.2. Реализация REST API

* Примеры реализации основных эндпоинтов API:
  + Получение списка категорий товаров
  + Получение списка товаров с фильтрацией
  + Получение детальной информации о товаре
  + Административные эндпоинты для управления товарами

### 3.3.3. Реализация бизнес-логики

* Описание реализации основных бизнес-процессов:
  + Обработка запросов к каталогу товаров
  + Валидация данных товаров
  + Фильтрация и сортировка товаров

Бизнес-логика приложения представляет собой совокупность правил, алгоритмов и процессов, которые определяют, как данные обрабатываются, трансформируются и используются для реализации требований предметной области. Для эффективной организации бизнес-логики в проекте “Дому мебель” был применен следующий комплекс архитектурных паттернов и подходов:

#### Использование паттернов проектирования

В разработке бизнес-логики были применены следующие паттерны:

1. **Domain-Driven Design (DDD)** - подход, при котором архитектура приложения строится вокруг бизнес-доменных объектов:

* // Пример доменной модели товара  
  export class Product {  
   private id: string;  
   private title: string;  
   private description: string;  
   private price: Money;  
   private category: Category;  
   private attributes: ProductAttribute[];  
   private images: ProductImage[];  
   private availabilityStatus: AvailabilityStatus;  
    
   constructor(props: ProductProps) {  
   // Валидация входных данных  
   this.validateProperties(props);  
    
   // Инициализация свойств  
   this.id = props.id;  
   this.title = props.title;  
   this.description = props.description;  
   this.price = props.price;  
   this.category = props.category;  
   this.attributes = props.attributes || [];  
   this.images = props.images || [];  
   this.availabilityStatus = props.availabilityStatus || AvailabilityStatus.IN\_STOCK;  
   }  
    
   // Бизнес-методы  
   public isAvailable(): boolean {  
   return this.availabilityStatus === AvailabilityStatus.IN\_STOCK;  
   }  
    
   public applyDiscount(percentage: number): void {  
   if (percentage <= 0 || percentage >= 100) {  
   throw new InvalidDiscountError('Процент скидки должен быть в диапазоне от 1 до 99');  
   }  
    
   this.price = this.price.applyDiscount(percentage);  
   }  
    
   public changeAvailability(status: AvailabilityStatus): void {  
   this.availabilityStatus = status;  
   }  
    
   // Методы валидации  
   private validateProperties(props: ProductProps): void {  
   if (!props.title || props.title.trim().length < 3) {  
   throw new ValidationError('Название товара должно содержать минимум 3 символа');  
   }  
    
   if (!props.price || props.price.getValue() <= 0) {  
   throw new ValidationError('Цена товара должна быть положительным числом');  
   }  
    
   if (!props.category) {  
   throw new ValidationError('Товар должен относиться к категории');  
   }  
   }  
  }

1. **Service Layer Pattern** - паттерн, координирующий операции домена и инфраструктуры:

* @injectable()  
  export class ProductService {  
   constructor(  
   @inject(TYPES.ProductRepository) private productRepository: IProductRepository,  
   @inject(TYPES.CategoryRepository) private categoryRepository: ICategoryRepository,  
   @inject(TYPES.EventBus) private eventBus: IEventBus  
   ) {}  
    
   public async getProductById(id: string): Promise<Product> {  
   const product = await this.productRepository.findById(id);  
    
   if (!product) {  
   throw new ProductNotFoundError(id);  
   }  
    
   return product;  
   }  
    
   public async createProduct(productData: CreateProductDTO): Promise<Product> {  
   // Проверка существования категории  
   const category = await this.categoryRepository.findById(productData.categoryId);  
    
   if (!category) {  
   throw new CategoryNotFoundError(productData.categoryId);  
   }  
    
   // Создание объекта товара  
   const price = new Money(productData.price, 'RUB');  
   const product = new Product({  
   id: uuid(),  
   title: productData.title,  
   description: productData.description,  
   price: price,  
   category: category,  
   attributes: this.mapToProductAttributes(productData.attributes),  
   images: this.mapToProductImages(productData.images)  
   });  
    
   // Сохранение в репозиторий  
   const savedProduct = await this.productRepository.save(product);  
    
   // Публикация события о создании товара  
   this.eventBus.publish(new ProductCreatedEvent(savedProduct));  
    
   return savedProduct;  
   }  
    
   public async updateProduct(id: string, productData: UpdateProductDTO): Promise<Product> {  
   const existingProduct = await this.getProductById(id);  
    
   // Обновление свойств товара  
   // ... логика обновления ...  
    
   const updatedProduct = await this.productRepository.save(existingProduct);  
    
   // Публикация события об обновлении товара  
   this.eventBus.publish(new ProductUpdatedEvent(updatedProduct));  
    
   return updatedProduct;  
   }  
    
   // Другие методы сервиса...  
  }

1. **Command Query Responsibility Segregation (CQRS)** - разделение операций чтения и записи:

* // Команда для создания продукта  
  export class CreateProductCommand implements ICommand {  
   constructor(public readonly productData: CreateProductDTO) {}  
  }  
    
  // Обработчик команды  
  @injectable()  
  export class CreateProductCommandHandler implements ICommandHandler<CreateProductCommand> {  
   constructor(  
   @inject(TYPES.ProductService) private productService: ProductService  
   ) {}  
    
   public async execute(command: CreateProductCommand): Promise<Product> {  
   return this.productService.createProduct(command.productData);  
   }  
  }  
    
  // Запрос для получения товаров  
  export class GetProductsByCategoryQuery implements IQuery<Product[]> {  
   constructor(  
   public readonly categoryId: string,  
   public readonly pagination: PaginationOptions,  
   public readonly sorting: SortOptions  
   ) {}  
  }  
    
  // Обработчик запроса  
  @injectable()  
  export class GetProductsByCategoryQueryHandler implements IQueryHandler<GetProductsByCategoryQuery, Product[]> {  
   constructor(  
   @inject(TYPES.ProductRepository) private productRepository: IProductRepository  
   ) {}  
    
   public async execute(query: GetProductsByCategoryQuery): Promise<Product[]> {  
   return this.productRepository.findByCategory(  
   query.categoryId,  
   query.pagination,  
   query.sorting  
   );  
   }  
  }

1. **Strategy Pattern** - используется для реализации различных алгоритмов сортировки и фильтрации:

* // Интерфейс стратегии фильтрации  
  export interface IFilterStrategy {  
   apply(products: Product[], criteria: any): Product[];  
  }  
    
  // Конкретная стратегия фильтрации по цене  
  @injectable()  
  export class PriceFilterStrategy implements IFilterStrategy {  
   apply(products: Product[], criteria: { minPrice?: number, maxPrice?: number }): Product[] {  
   return products.filter(product => {  
   const price = product.getPrice().getValue();  
    
   if (criteria.minPrice !== undefined && price < criteria.minPrice) {  
   return false;  
   }  
    
   if (criteria.maxPrice !== undefined && price > criteria.maxPrice) {  
   return false;  
   }  
    
   return true;  
   });  
   }  
  }  
    
  // Фабрика стратегий фильтрации  
  @injectable()  
  export class FilterStrategyFactory {  
   private strategies: Map<string, IFilterStrategy> = new Map();  
    
   constructor(  
   @inject(TYPES.PriceFilterStrategy) priceFilterStrategy: PriceFilterStrategy,  
   @inject(TYPES.CategoryFilterStrategy) categoryFilterStrategy: IFilterStrategy,  
   @inject(TYPES.AttributeFilterStrategy) attributeFilterStrategy: IFilterStrategy  
   ) {  
   this.strategies.set('price', priceFilterStrategy);  
   this.strategies.set('category', categoryFilterStrategy);  
   this.strategies.set('attribute', attributeFilterStrategy);  
   }  
    
   getStrategy(type: string): IFilterStrategy {  
   const strategy = this.strategies.get(type);  
    
   if (!strategy) {  
   throw new FilterStrategyNotFoundError(type);  
   }  
    
   return strategy;  
   }  
  }

1. **Mediator Pattern** - для централизованной обработки команд и запросов:

* @injectable()  
  export class Mediator implements IMediator {  
   private commandHandlers: Map<string, ICommandHandler<any>> = new Map();  
   private queryHandlers: Map<string, IQueryHandler<any, any>> = new Map();  
    
   public registerCommandHandler<T extends ICommand>(  
   commandType: Constructor<T>,  
   handler: ICommandHandler<T>  
   ): void {  
   this.commandHandlers.set(commandType.name, handler);  
   }  
    
   public registerQueryHandler<T extends IQuery<R>, R>(  
   queryType: Constructor<T>,  
   handler: IQueryHandler<T, R>  
   ): void {  
   this.queryHandlers.set(queryType.name, handler);  
   }  
    
   public async sendCommand<T extends ICommand>(command: T): Promise<any> {  
   const handler = this.commandHandlers.get(command.constructor.name);  
    
   if (!handler) {  
   throw new HandlerNotFoundError(`Handler for command ${command.constructor.name} not found`);  
   }  
    
   try {  
   return await handler.execute(command);  
   } catch (error) {  
   // Логирование ошибки  
   throw error;  
   }  
   }  
    
   public async sendQuery<T extends IQuery<R>, R>(query: T): Promise<R> {  
   const handler = this.queryHandlers.get(query.constructor.name);  
    
   if (!handler) {  
   throw new HandlerNotFoundError(`Handler for query ${query.constructor.name} not found`);  
   }  
    
   try {  
   return await handler.execute(query);  
   } catch (error) {  
   // Логирование ошибки  
   throw error;  
   }  
   }  
  }

#### Реализация валидации данных

Для обеспечения корректности и целостности данных на уровне бизнес-логики реализован механизм валидации с использованием:

// Интерфейс валидатора  
export interface IValidator<T> {  
 validate(data: T): ValidationResult;  
}  
  
// Результат валидации  
export class ValidationResult {  
 private errors: ValidationError[] = [];  
   
 public addError(field: string, message: string): void {  
 this.errors.push(new ValidationError(field, message));  
 }  
   
 public hasErrors(): boolean {  
 return this.errors.length > 0;  
 }  
   
 public getErrors(): ValidationError[] {  
 return [...this.errors];  
 }  
}  
  
// Пример валидатора для создания товара  
@injectable()  
export class CreateProductValidator implements IValidator<CreateProductDTO> {  
 constructor(  
 @inject(TYPES.CategoryRepository) private categoryRepository: ICategoryRepository  
 ) {}  
   
 public async validate(data: CreateProductDTO): Promise<ValidationResult> {  
 const result = new ValidationResult();  
   
 if (!data.title || data.title.trim().length < 3) {  
 result.addError('title', 'Название товара должно содержать минимум 3 символа');  
 }  
   
 if (data.title && data.title.trim().length > 100) {  
 result.addError('title', 'Название товара не должно превышать 100 символов');  
 }  
   
 if (!data.price || data.price <= 0) {  
 result.addError('price', 'Цена товара должна быть положительным числом');  
 }  
   
 if (!data.categoryId) {  
 result.addError('categoryId', 'Категория товара обязательна');  
 } else {  
 const categoryExists = await this.categoryRepository.exists(data.categoryId);  
 if (!categoryExists) {  
 result.addError('categoryId', `Категория с ID ${data.categoryId} не существует`);  
 }  
 }  
   
 // Дополнительные правила валидации...  
   
 return result;  
 }  
}

#### Применение принципов SOLID

В разработке бизнес-логики последовательно применялись принципы SOLID:

1. **Single Responsibility Principle**: Каждый класс имеет единственную ответственность. Например, класс Product отвечает только за бизнес-логику товара, а ProductRepository - только за доступ к хранилищу товаров.
2. **Open/Closed Principle**: Классы открыты для расширения, но закрыты для модификации. Например, система фильтрации товаров реализована через стратегии, что позволяет добавлять новые стратегии без изменения существующего кода.
3. **Liskov Substitution Principle**: Наследники могут заменять базовые классы без изменения корректности программы. Например, различные реализации интерфейса IProductRepository могут заменять друг друга.
4. **Interface Segregation Principle**: Клиенты не должны зависеть от интерфейсов, которые они не используют. Например, вместо одного большого интерфейса репозитория используются специализированные интерфейсы для разных доменных сущностей.
5. **Dependency Inversion Principle**: Зависимости в системе строятся от абстракций, а не от конкретных реализаций. Например, сервисы зависят от интерфейсов репозиториев, а не от их конкретных реализаций.

Комплексное применение данных паттернов и принципов позволило создать гибкую, модульную и поддерживаемую архитектуру бизнес-логики, обеспечивающую высокую расширяемость и адаптивность системы к изменяющимся требованиям бизнеса.

### 3.3.4. Взаимодействие с базой данных

Эффективное взаимодействие с базой данных является критическим фактором производительности и надежности приложения. В проекте “Дому мебель” реализован комплексный подход к работе с базой данных, включающий следующие компоненты и методики:

#### Архитектура доступа к данным

Для организации доступа к данным применена многослойная архитектура:

1. **Репозитории** - абстрагируют логику доступа к данным от бизнес-логики:

* // Интерфейс репозитория  
  export interface IProductRepository {  
   findById(id: string): Promise<Product | null>;  
   findAll(options?: FindOptions): Promise<Product[]>;  
   findByCategory(categoryId: string, pagination?: PaginationOptions, sorting?: SortOptions): Promise<Product[]>;  
   count(filter?: FilterOptions): Promise<number>;  
   save(product: Product): Promise<Product>;  
   delete(id: string): Promise<void>;  
   exists(id: string): Promise<boolean>;  
  }  
    
  // Реализация репозитория с использованием PostgreSQL  
  @injectable()  
  export class PostgresProductRepository implements IProductRepository {  
   constructor(  
   @inject(TYPES.DatabaseConnection) private dbConnection: DatabaseConnection,  
   @inject(TYPES.Logger) private logger: ILogger  
   ) {}  
    
   public async findById(id: string): Promise<Product | null> {  
   try {  
   const query = `  
   SELECT   
   p.id, p.title, p.description, p.price, p.currency, p.availability\_status,  
   c.id as category\_id, c.name as category\_name, c.parent\_id as category\_parent\_id  
   FROM products p  
   LEFT JOIN categories c ON p.category\_id = c.id  
   WHERE p.id = $1 AND p.deleted\_at IS NULL  
   `;  
    
   const result = await this.dbConnection.query(query, [id]);  
    
   if (result.rows.length === 0) {  
   return null;  
   }  
    
   // Получение атрибутов товара  
   const attributesQuery = `  
   SELECT a.id, a.name, pa.value  
   FROM product\_attributes pa  
   JOIN attributes a ON pa.attribute\_id = a.id  
   WHERE pa.product\_id = $1  
   `;  
    
   const attributesResult = await this.dbConnection.query(attributesQuery, [id]);  
    
   // Получение изображений товара  
   const imagesQuery = `  
   SELECT id, url, alt, is\_main  
   FROM product\_images  
   WHERE product\_id = $1  
   ORDER BY is\_main DESC, sort\_order ASC  
   `;  
    
   const imagesResult = await this.dbConnection.query(imagesQuery, [id]);  
    
   // Маппинг данных из БД в доменную модель  
   return this.mapToProduct(result.rows[0], attributesResult.rows, imagesResult.rows);  
   } catch (error) {  
   this.logger.error('Error in findById', { id, error });  
   throw new RepositoryError('Failed to find product by ID', error);  
   }  
   }  
    
   public async findByCategory(  
   categoryId: string,   
   pagination: PaginationOptions = { page: 1, limit: 20 },  
   sorting: SortOptions = { field: 'created\_at', direction: 'DESC' }  
   ): Promise<Product[]> {  
   try {  
   const offset = (pagination.page - 1) \* pagination.limit;  
    
   // Получение всех подкатегорий  
   const categoriesQuery = `  
   WITH RECURSIVE category\_tree AS (  
   SELECT id FROM categories WHERE id = $1  
   UNION ALL  
   SELECT c.id FROM categories c  
   JOIN category\_tree ct ON c.parent\_id = ct.id  
   )  
   SELECT id FROM category\_tree  
   `;  
    
   const categoriesResult = await this.dbConnection.query(categoriesQuery, [categoryId]);  
   const categoryIds = categoriesResult.rows.map(row => row.id);  
    
   if (categoryIds.length === 0) {  
   return [];  
   }  
    
   // Формирование сортировки  
   const sortField = this.getSafeColumnName(sorting.field);  
   const sortDirection = sorting.direction === 'ASC' ? 'ASC' : 'DESC';  
    
   // Получение товаров  
   const query = `  
   SELECT   
   p.id, p.title, p.description, p.price, p.currency, p.availability\_status,  
   c.id as category\_id, c.name as category\_name, c.parent\_id as category\_parent\_id  
   FROM products p  
   LEFT JOIN categories c ON p.category\_id = c.id  
   WHERE p.category\_id = ANY($1) AND p.deleted\_at IS NULL  
   ORDER BY p.${sortField} ${sortDirection}  
   LIMIT $2 OFFSET $3  
   `;  
    
   const result = await this.dbConnection.query(query, [categoryIds, pagination.limit, offset]);  
    
   // Получение идентификаторов товаров  
   const productIds = result.rows.map(row => row.id);  
    
   if (productIds.length === 0) {  
   return [];  
   }  
    
   // Получение атрибутов для всех товаров одним запросом  
   const attributesQuery = `  
   SELECT pa.product\_id, a.id, a.name, pa.value  
   FROM product\_attributes pa  
   JOIN attributes a ON pa.attribute\_id = a.id  
   WHERE pa.product\_id = ANY($1)  
   `;  
    
   const attributesResult = await this.dbConnection.query(attributesQuery, [productIds]);  
    
   // Получение изображений для всех товаров одним запросом  
   const imagesQuery = `  
   SELECT product\_id, id, url, alt, is\_main  
   FROM product\_images  
   WHERE product\_id = ANY($1)  
   ORDER BY is\_main DESC, sort\_order ASC  
   `;  
    
   const imagesResult = await this.dbConnection.query(imagesQuery, [productIds]);  
    
   // Группировка атрибутов и изображений по ID товара  
   const attributesByProductId = this.groupByProductId(attributesResult.rows);  
   const imagesByProductId = this.groupByProductId(imagesResult.rows);  
    
   // Маппинг данных из БД в доменные модели  
   return result.rows.map(row =>   
   this.mapToProduct(  
   row,   
   attributesByProductId[row.id] || [],   
   imagesByProductId[row.id] || []  
   )  
   );  
   } catch (error) {  
   this.logger.error('Error in findByCategory', { categoryId, pagination, sorting, error });  
   throw new RepositoryError('Failed to find products by category', error);  
   }  
   }  
    
   // Другие методы репозитория...  
    
   // Вспомогательные методы для маппинга данных и валидации  
   private mapToProduct(  
   row: any,   
   attributes: any[],   
   images: any[]  
   ): Product {  
   // Реализация маппинга данных из БД в доменную модель  
   // ...  
   }  
    
   private groupByProductId(rows: any[]): Record<string, any[]> {  
   return rows.reduce((acc, row) => {  
   const productId = row.product\_id;  
   if (!acc[productId]) {  
   acc[productId] = [];  
   }  
   acc[productId].push(row);  
   return acc;  
   }, {});  
   }  
    
   private getSafeColumnName(columnName: string): string {  
   // Валидация имени колонки для предотвращения SQL-инъекций  
   const safeColumns = [  
   'id', 'title', 'price', 'created\_at', 'updated\_at', 'availability\_status'  
   ];  
    
   return safeColumns.includes(columnName) ? columnName : 'created\_at';  
   }  
  }

1. **Unit of Work** - паттерн для обеспечения атомарности операций с БД:

* @injectable()  
  export class UnitOfWork implements IUnitOfWork {  
   constructor(  
   @inject(TYPES.DatabaseConnection) private dbConnection: DatabaseConnection,  
   @inject(TYPES.Logger) private logger: ILogger  
   ) {}  
    
   public async begin(): Promise<void> {  
   await this.dbConnection.query('BEGIN');  
   }  
    
   public async commit(): Promise<void> {  
   await this.dbConnection.query('COMMIT');  
   }  
    
   public async rollback(): Promise<void> {  
   await this.dbConnection.query('ROLLBACK');  
   }  
    
   public async withTransaction<T>(callback: () => Promise<T>): Promise<T> {  
   try {  
   await this.begin();  
   const result = await callback();  
   await this.commit();  
   return result;  
   } catch (error) {  
   await this.rollback();  
   this.logger.error('Transaction rolled back', { error });  
   throw error;  
   }  
   }  
  }

#### Оптимизация SQL-запросов

Для повышения производительности запросов к базе данных применены следующие методики:

1. **Индексирование таблиц**:

* -- Индексы для таблицы products  
  CREATE INDEX idx\_products\_category\_id ON products(category\_id);  
  CREATE INDEX idx\_products\_price ON products(price);  
  CREATE INDEX idx\_products\_availability\_status ON products(availability\_status);  
  CREATE INDEX idx\_products\_created\_at ON products(created\_at);  
    
  -- Индексы для атрибутов товаров  
  CREATE INDEX idx\_product\_attributes\_product\_id ON product\_attributes(product\_id);  
  CREATE INDEX idx\_product\_attributes\_attribute\_id ON product\_attributes(attribute\_id);  
  CREATE INDEX idx\_product\_attributes\_value ON product\_attributes(value);  
    
  -- Составной индекс для оптимизации запросов с фильтрацией  
  CREATE INDEX idx\_products\_category\_status\_price ON products(category\_id, availability\_status, price);

1. **Материализованные представления** для часто запрашиваемых данных:

* -- Материализованное представление для каталога товаров  
  CREATE MATERIALIZED VIEW product\_catalog AS  
  SELECT   
   p.id,   
   p.title,   
   p.description,   
   p.price,   
   p.currency,  
   p.availability\_status,  
   p.category\_id,  
   c.name as category\_name,  
   pi.url as main\_image\_url  
  FROM products p  
  JOIN categories c ON p.category\_id = c.id  
  LEFT JOIN (  
   SELECT DISTINCT ON (product\_id) product\_id, url  
   FROM product\_images  
   WHERE is\_main = true  
   ORDER BY product\_id, is\_main DESC  
  ) pi ON p.id = pi.product\_id  
  WHERE p.deleted\_at IS NULL  
  ORDER BY p.created\_at DESC;  
    
  -- Индекс для материализованного представления  
  CREATE INDEX idx\_product\_catalog\_category\_id ON product\_catalog(category\_id);  
  CREATE INDEX idx\_product\_catalog\_price ON product\_catalog(price);  
    
  -- Функция для обновления материализованного представления  
  CREATE OR REPLACE FUNCTION refresh\_product\_catalog()  
  RETURNS TRIGGER AS $$  
  BEGIN  
   REFRESH MATERIALIZED VIEW CONCURRENTLY product\_catalog;  
   RETURN NULL;  
  END;  
  $$ LANGUAGE plpgsql;  
    
  -- Триггер для автоматического обновления материализованного представления  
  CREATE TRIGGER refresh\_product\_catalog\_trigger  
  AFTER INSERT OR UPDATE OR DELETE ON products  
  FOR EACH STATEMENT  
  EXECUTE FUNCTION refresh\_product\_catalog();

1. **Пагинация и оптимизация запросов** с использованием LIMIT и OFFSET:

* public async findAll(options: FindOptions = {}): Promise<Product[]> {  
   const {  
   pagination = { page: 1, limit: 20 },  
   sorting = { field: 'created\_at', direction: 'DESC' },  
   filter = {}  
   } = options;  
    
   const offset = (pagination.page - 1) \* pagination.limit;  
   const sortField = this.getSafeColumnName(sorting.field);  
   const sortDirection = sorting.direction === 'ASC' ? 'ASC' : 'DESC';  
    
   // Формирование условий фильтрации  
   const { whereClause, params } = this.buildWhereClause(filter);  
    
   const query = `  
   SELECT \* FROM product\_catalog  
   ${whereClause ? `WHERE ${whereClause}` : ''}  
   ORDER BY ${sortField} ${sortDirection}  
   LIMIT $${params.length + 1} OFFSET $${params.length + 2}  
   `;  
    
   const result = await this.dbConnection.query(query, [...params, pagination.limit, offset]);  
    
   // Преобразование результатов запроса в доменные объекты  
   return result.rows.map(row => this.mapToCatalogProduct(row));  
  }  
    
  private buildWhereClause(filter: FilterOptions): { whereClause: string, params: any[] } {  
   const conditions: string[] = [];  
   const params: any[] = [];  
    
   if (filter.categoryId) {  
   conditions.push(`category\_id = $${params.length + 1}`);  
   params.push(filter.categoryId);  
   }  
    
   if (filter.minPrice !== undefined) {  
   conditions.push(`price >= $${params.length + 1}`);  
   params.push(filter.minPrice);  
   }  
    
   if (filter.maxPrice !== undefined) {  
   conditions.push(`price <= $${params.length + 1}`);  
   params.push(filter.maxPrice);  
   }  
    
   if (filter.availabilityStatus) {  
   conditions.push(`availability\_status = $${params.length + 1}`);  
   params.push(filter.availabilityStatus);  
   }  
    
   if (filter.search) {  
   conditions.push(`(title ILIKE $${params.length + 1} OR description ILIKE $${params.length + 1})`);  
   params.push(`%${filter.search}%`);  
   }  
    
   return {  
   whereClause: conditions.length > 0 ? conditions.join(' AND ') : '',  
   params  
   };  
  }

#### Миграции и управление схемой БД

Для поддержания целостности схемы базы данных и упрощения ее обновления реализована система миграций:

// Интерфейс для миграций  
export interface IMigration {  
 up(): Promise<void>;  
 down(): Promise<void>;  
}  
  
// Пример миграции  
export class CreateProductsTable implements IMigration {  
 constructor(private dbConnection: DatabaseConnection) {}  
   
 public async up(): Promise<void> {  
 await this.dbConnection.query(`  
 CREATE TABLE products (  
 id UUID PRIMARY KEY,  
 title VARCHAR(100) NOT NULL,  
 description TEXT,  
 price DECIMAL(10, 2) NOT NULL,  
 currency VARCHAR(3) NOT NULL DEFAULT 'RUB',  
 category\_id UUID NOT NULL,  
 availability\_status VARCHAR(20) NOT NULL DEFAULT 'IN\_STOCK',  
 created\_at TIMESTAMP NOT NULL DEFAULT NOW(),  
 updated\_at TIMESTAMP NOT NULL DEFAULT NOW(),  
 deleted\_at TIMESTAMP,  
 CONSTRAINT fk\_category FOREIGN KEY (category\_id) REFERENCES categories(id)  
 )  
 `);  
   
 // Создание индексов  
 await this.dbConnection.query(`  
 CREATE INDEX idx\_products\_category\_id ON products(category\_id);  
 CREATE INDEX idx\_products\_price ON products(price);  
 CREATE INDEX idx\_products\_availability\_status ON products(availability\_status);  
 CREATE INDEX idx\_products\_created\_at ON products(created\_at);  
 `);  
 }  
   
 public async down(): Promise<void> {  
 await this.dbConnection.query(`  
 DROP TABLE IF EXISTS products CASCADE  
 `);  
 }  
}  
  
// Сервис для запуска миграций  
@injectable()  
export class MigrationService {  
 private migrations: IMigration[] = [];  
   
 constructor(  
 @inject(TYPES.DatabaseConnection) private dbConnection: DatabaseConnection,  
 @inject(TYPES.Logger) private logger: ILogger  
 ) {  
 // Регистрация миграций  
 this.migrations = [  
 new CreateCategoriesTable(dbConnection),  
 new CreateProductsTable(dbConnection),  
 new CreateProductAttributesTable(dbConnection),  
 new CreateProductImagesTable(dbConnection),  
 // ...другие миграции  
 ];  
 }  
   
 public async migrate(): Promise<void> {  
 // Создание таблицы для хранения информации о миграциях, если она не существует  
 await this.createMigrationsTable();  
   
 // Получение информации о выполненных миграциях  
 const appliedMigrations = await this.getAppliedMigrations();  
   
 // Выполнение новых миграций  
 for (let i = 0; i < this.migrations.length; i++) {  
 const migrationName = this.migrations[i].constructor.name;  
   
 if (!appliedMigrations.includes(migrationName)) {  
 this.logger.info(`Applying migration: ${migrationName}`);  
   
 try {  
 await this.dbConnection.query('BEGIN');  
 await this.migrations[i].up();  
 await this.recordMigration(migrationName);  
 await this.dbConnection.query('COMMIT');  
   
 this.logger.info(`Migration applied: ${migrationName}`);  
 } catch (error) {  
 await this.dbConnection.query('ROLLBACK');  
 this.logger.error(`Migration failed: ${migrationName}`, { error });  
 throw error;  
 }  
 }  
 }  
 }  
   
 private async createMigrationsTable(): Promise<void> {  
 await this.dbConnection.query(`  
 CREATE TABLE IF NOT EXISTS migrations (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(100) NOT NULL,  
 applied\_at TIMESTAMP NOT NULL DEFAULT NOW()  
 )  
 `);  
 }  
   
 private async getAppliedMigrations(): Promise<string[]> {  
 const result = await this.dbConnection.query(`  
 SELECT name FROM migrations ORDER BY id ASC  
 `);  
   
 return result.rows.map(row => row.name);  
 }  
   
 private async recordMigration(name: string): Promise<void> {  
 await this.dbConnection.query(`  
 INSERT INTO migrations (name) VALUES ($1)  
 `, [name]);  
 }  
}

Применение комплексного подхода к взаимодействию с базой данных позволило обеспечить:

1. Высокую производительность при работе с большими объемами данных
2. Масштабируемость системы при увеличении нагрузки
3. Поддержание целостности данных через транзакционный механизм
4. Гибкость и адаптивность схемы данных к изменяющимся требованиям
5. Четкое разделение слоя доступа к данным от бизнес-логики

Оптимизированные запросы, правильно спроектированные индексы и материализованные представления позволили значительно повысить скорость работы приложения и обеспечить комфортное использование каталога даже при большом количестве товаров и высокой пользовательской нагрузке.

### 3.3.5. Обработка ошибок и логирование

* Реализация централизованной обработки ошибок
* Система логирования и мониторинга работы сервера
* Отправка уведомлений о критических ошибках

## 3.4. Реализация клиентской части

Клиентская часть интернет-витрины “Дому мебель” представляет собой современное одностраничное приложение (SPA), разработанное с использованием библиотеки React и архитектурного подхода Feature-Sliced Design (FSD). Такая архитектура обеспечивает четкое разделение ответственности между слоями приложения, высокую модульность и возможность легкого масштабирования.

Клиентское приложение взаимодействует с серверной частью через REST API, получая и отправляя данные в формате JSON. Для управления состоянием приложения используется Redux Toolkit, который обеспечивает предсказуемое управление состоянием и удобную работу с асинхронными операциями через RTK Query.

Архитектура клиентского приложения основана на принципе разделения кода на слои, в соответствии с методологией Feature-Sliced Design:

1. **Слой app** - содержит глобальные настройки приложения, провайдеры, инициализацию и маршрутизацию.
2. **Слой processes** - бизнес-процессы, охватывающие несколько сущностей и feature.
3. **Слой pages** - композиция виджетов и фич для конкретных страниц.
4. **Слой widgets** - композиционные компоненты, включающие несколько entities и features.
5. **Слой features** - интерактивные элементы пользовательского интерфейса, связанные с бизнес-логикой.
6. **Слой entities** - бизнес-сущности приложения.
7. **Слой shared** - переиспользуемые модули, библиотеки, UI-компоненты и утилиты.

Такая организация кода обеспечивает чистую архитектуру с явно выраженными границами между слоями и модулями, что значительно упрощает поддержку и развитие приложения. Каждый слой имеет строгие правила импорта, что предотвращает циклические зависимости и способствует созданию правильной архитектуры.

Организация компонентов следует принципу атомарного дизайна, при котором более сложные компоненты строятся из простых. Например, компонент каталога товаров включает в себя компоненты фильтрации, сортировки, карточек товаров, пагинации и другие более мелкие элементы.

Реализация маршрутизации и навигации в интернет-витрине построена на базе библиотеки React Router, которая обеспечивает навигацию между различными разделами приложения без перезагрузки страницы. Структура маршрутов организована иерархически и соответствует бизнес-логике приложения:

// app/router/AppRouter.tsx  
export const AppRouter: FC = () => {  
 return (  
 <Routes>  
 {/\* Публичные маршруты \*/}  
 <Route path="/" element={<MainLayout />}>  
 <Route index element={<HomePage />} />  
 <Route path="catalog" element={<CatalogPage />} />  
 <Route path="catalog/:categoryId" element={<CategoryPage />} />  
 <Route path="product/:id" element={<ProductPage />} />  
 <Route path="contacts" element={<ContactsPage />} />  
 <Route path="about" element={<AboutPage />} />  
 </Route>  
   
 {/\* Административные маршруты с защитой \*/}  
 <Route path="admin" element={<ProtectedRoute><AdminLayout /></ProtectedRoute>}>  
 <Route index element={<AdminDashboardPage />} />  
 <Route path="products" element={<ProductListPage />} />  
 <Route path="products/new" element={<CreateProductPage />} />  
 <Route path="products/:id" element={<EditProductPage />} />  
 <Route path="categories" element={<CategoriesPage />} />  
 <Route path="orders" element={<OrdersPage />} />  
 </Route>  
   
 {/\* Обработка ошибок \*/}  
 <Route path="\*" element={<NotFoundPage />} />  
 </Routes>  
 );  
};

Для обеспечения безопасности административного раздела реализован компонент ProtectedRoute, который проверяет аутентификацию пользователя:

// app/router/ProtectedRoute.tsx  
export const ProtectedRoute: FC<PropsWithChildren> = ({ children }) => {  
 const { isAuthenticated, isLoading } = useAuth();  
 const location = useLocation();  
   
 if (isLoading) {  
 return <Loader />;  
 }  
   
 if (!isAuthenticated) {  
 return <Navigate to="/login" state={{ from: location }} replace />;  
 }  
   
 return <>{children}</>;  
};

Навигация между разделами сайта реализована с помощью основных компонентов. Главное меню обеспечивает основную навигацию по сайту, хлебные крошки используются для навигации внутри иерархических структур, а пагинация - для навигации по страницам каталога.

Пример реализации главного меню:

// widgets/main-menu/ui/MainMenu.tsx  
export const MainMenu: FC = () => {  
 const location = useLocation();  
   
 const links = [  
 { path: '/', label: 'Главная' },  
 { path: '/catalog', label: 'Каталог' },  
 { path: '/about', label: 'О нас' },  
 { path: '/contacts', label: 'Контакты' }  
 ];  
   
 return (  
 <nav className={styles.mainMenu}>  
 <ul className={styles.menuList}>  
 {links.map(link => (  
 <li key={link.path} className={styles.menuItem}>  
 <NavLink   
 to={link.path}  
 className={({ isActive }) =>   
 cn(styles.menuLink, { [styles.active]: isActive })  
 }  
 end  
 >  
 {link.label}  
 </NavLink>  
 </li>  
 ))}  
 </ul>  
 </nav>  
 );  
};

Компонент хлебных крошек для навигации внутри каталога:

// widgets/breadcrumbs/ui/Breadcrumbs.tsx  
export const Breadcrumbs: FC<BreadcrumbsProps> = ({ items }) => {  
 if (!items.length) {  
 return null;  
 }  
   
 return (  
 <nav aria-label="Breadcrumbs" className={styles.breadcrumbs}>  
 <ol className={styles.breadcrumbsList}>  
 {items.map((item, index) => {  
 const isLast = index === items.length - 1;  
   
 return (  
 <li   
 key={item.path || index}   
 className={styles.breadcrumbsItem}  
 >  
 {isLast ? (  
 <span className={styles.current}>{item.label}</span>  
 ) : (  
 <Link to={item.path} className={styles.link}>  
 {item.label}  
 </Link>  
 )}  
   
 {!isLast && (  
 <span className={styles.separator}>/</span>  
 )}  
 </li>  
 );  
 })}  
 </ol>  
 </nav>  
 );  
};

Важной особенностью навигации является синхронизация состояния фильтров и сортировки с URL-параметрами. Это позволяет сохранять состояние фильтрации при обновлении страницы и делиться ссылками на отфильтрованные результаты:

// features/catalog-filter/model/useFilterParams.ts  
export const useFilterParams = () => {  
 const [searchParams, setSearchParams] = useSearchParams();  
   
 const currentFilters: FilterState = {  
 categoryId: searchParams.get('category') || undefined,  
 priceMin: searchParams.get('priceMin') ?   
 Number(searchParams.get('priceMin')) : undefined,  
 priceMax: searchParams.get('priceMax') ?   
 Number(searchParams.get('priceMax')) : undefined,  
 inStock: searchParams.get('inStock') === 'true',  
 sort: searchParams.get('sort') || 'newest',  
 page: searchParams.get('page') ?   
 Number(searchParams.get('page')) : 1  
 };  
   
 const updateFilters = (newFilters: Partial<FilterState>) => {  
 const updatedFilters = { ...currentFilters, ...newFilters };  
   
 // Сбрасываем страницу при изменении фильтров  
 if (newFilters.categoryId !== undefined ||   
 newFilters.priceMin !== undefined ||   
 newFilters.priceMax !== undefined ||   
 newFilters.inStock !== undefined ||  
 newFilters.sort !== undefined) {  
 updatedFilters.page = 1;  
 }  
   
 const params = new URLSearchParams();  
   
 if (updatedFilters.categoryId) {  
 params.set('category', updatedFilters.categoryId);  
 }  
   
 if (updatedFilters.priceMin) {  
 params.set('priceMin', updatedFilters.priceMin.toString());  
 }  
   
 if (updatedFilters.priceMax) {  
 params.set('priceMax', updatedFilters.priceMax.toString());  
 }  
   
 if (updatedFilters.inStock) {  
 params.set('inStock', 'true');  
 }  
   
 if (updatedFilters.sort !== 'newest') {  
 params.set('sort', updatedFilters.sort);  
 }  
   
 if (updatedFilters.page > 1) {  
 params.set('page', updatedFilters.page.toString());  
 }  
   
 setSearchParams(params);  
 };  
   
 return { filters: currentFilters, updateFilters };  
};

Применение этого хука в компоненте каталога:

// pages/CatalogPage/ui/CatalogPage.tsx  
export const CatalogPage: FC = () => {  
 const { filters, updateFilters } = useFilterParams();  
 const { data: products, isLoading } = useProducts(filters);  
 const { data: categories } = useCategories();  
   
 const handleFilterChange = (newFilters: Partial<FilterState>) => {  
 updateFilters(newFilters);  
 };  
   
 const handlePageChange = (page: number) => {  
 updateFilters({ page });  
 // Прокрутка к началу списка товаров  
 window.scrollTo({ top: 0, behavior: 'smooth' });  
 };  
   
 return (  
 <div className={styles.catalogPage}>  
 <h1 className={styles.title}>Каталог мебели</h1>  
   
 <div className={styles.catalogLayout}>  
 <aside className={styles.sidebar}>  
 <FilterPanel   
 categories={categories || []}  
 initialFilters={filters}  
 onFilterChange={handleFilterChange}  
 />  
 </aside>  
   
 <main className={styles.content}>  
 <div className={styles.sorting}>  
 <SortingPanel   
 value={filters.sort}  
 onChange={(sort) => handleFilterChange({ sort })}  
 />  
 </div>  
   
 <CatalogList   
 products={products?.items || []}  
 isLoading={isLoading}  
 />  
   
 {products && products.pagination.totalPages > 1 && (  
 <Pagination   
 currentPage={filters.page}  
 totalPages={products.pagination.totalPages}  
 onPageChange={handlePageChange}  
 />  
 )}  
 </main>  
 </div>  
 </div>  
 );  
};

Пользовательский интерфейс интернет-витрины “Дому мебель” разработан с учетом современных принципов веб-дизайна, удобства использования и фирменного стиля компании. Для обеспечения единообразия визуального стиля и ускорения разработки была создана дизайн-система, включающая дизайн-токены (базовые значения для цветов, типографики, отступов и размеров) и базовые компоненты (набор переиспользуемых UI-компонентов).

// shared/styles/tokens/\_colors.scss  
$primary: #2a5f8f;  
$primary-light: #3a75ab;  
$primary-dark: #1a4a78;  
  
$secondary: #f6a623;  
$secondary-light: #ffb83d;  
$secondary-dark: #e09310;  
  
$text-primary: #333333;  
$text-secondary: #666666;  
$text-light: #999999;  
  
$background: #ffffff;  
$background-light: #f5f7fa;  
$background-dark: #e6e9ee;  
  
$border: #dcdfe5;  
$border-light: #ebedf0;  
$border-dark: #c5c9d0;  
  
$success: #4caf50;  
$error: #f44336;  
$warning: #ff9800;  
$info: #2196f3;

// shared/styles/tokens/\_typography.scss  
$font-family-base: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;  
$font-family-heading: 'Montserrat', -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;  
  
$font-size-xs: 0.75rem; // 12px  
$font-size-sm: 0.875rem; // 14px  
$font-size-md: 1rem; // 16px  
$font-size-lg: 1.125rem; // 18px  
$font-size-xl: 1.25rem; // 20px  
$font-size-2xl: 1.5rem; // 24px  
$font-size-3xl: 1.875rem; // 30px  
$font-size-4xl: 2.25rem; // 36px  
  
$line-height-tight: 1.2;  
$line-height-normal: 1.5;  
$line-height-loose: 1.8;  
  
$font-weight-normal: 400;  
$font-weight-medium: 500;  
$font-weight-semibold: 600;  
$font-weight-bold: 700;

Базовые компоненты обеспечивают унифицированный внешний вид и поведение элементов интерфейса:

// shared/ui/button/Button.tsx  
export type ButtonVariant = 'primary' | 'secondary' | 'outline' | 'text';  
export type ButtonSize = 'sm' | 'md' | 'lg';  
  
export interface ButtonProps extends ButtonHTMLAttributes<HTMLButtonElement> {  
 variant?: ButtonVariant;  
 size?: ButtonSize;  
 fullWidth?: boolean;  
 leftIcon?: ReactNode;  
 rightIcon?: ReactNode;  
}  
  
export const Button: FC<ButtonProps> = ({  
 children,  
 variant = 'primary',  
 size = 'md',  
 fullWidth = false,  
 leftIcon,  
 rightIcon,  
 className,  
 disabled,  
 ...props  
}) => {  
 return (  
 <button  
 className={cn(  
 styles.button,  
 styles[`variant-${variant}`],  
 styles[`size-${size}`],  
 { [styles.fullWidth]: fullWidth },  
 { [styles.disabled]: disabled },  
 className  
 )}  
 disabled={disabled}  
 {...props}  
 >  
 {leftIcon && <span className={styles.leftIcon}>{leftIcon}</span>}  
 <span className={styles.content}>{children}</span>  
 {rightIcon && <span className={styles.rightIcon}>{rightIcon}</span>}  
 </button>  
 );  
};

Композитные компоненты строятся на основе базовых и реализуют более сложную бизнес-логику:

// shared/ui/card/Card.tsx  
export interface CardProps {  
 title?: ReactNode;  
 subtitle?: ReactNode;  
 image?: string;  
 footer?: ReactNode;  
 className?: string;  
 children: ReactNode;  
}  
  
export const Card: FC<CardProps> = ({  
 title,  
 subtitle,  
 image,  
 footer,  
 className,  
 children  
}) => {  
 return (  
 <div className={cn(styles.card, className)}>  
 {image && (  
 <div className={styles.imageContainer}>  
 <img src={image} alt={typeof title === 'string' ? title : 'Card image'} className={styles.image} />  
 </div>  
 )}  
   
 {(title || subtitle) && (  
 <div className={styles.header}>  
 {title && <h3 className={styles.title}>{title}</h3>}  
 {subtitle && <div className={styles.subtitle}>{subtitle}</div>}  
 </div>  
 )}  
   
 <div className={styles.content}>{children}</div>  
   
 {footer && <div className={styles.footer}>{footer}</div>}  
 </div>  
 );  
};

Отдельное внимание уделено разработке адаптивных интерфейсов, которые обеспечивают корректное отображение и удобство использования на различных устройствах: от настольных компьютеров до мобильных телефонов. Интернет-витрина “Дому мебель” разработана с учетом адаптивности, что позволяет пользователям комфортно взаимодействовать с сайтом на различных устройствах.

Подход к реализации адаптивного дизайна основан на принципе “Mobile First”, который предполагает первоначальную разработку интерфейса для мобильных устройств с последующим расширением функциональности для более крупных экранов. Основные особенности подхода включают использование гибкой сетки (Flexbox и CSS Grid), медиа-запросов для настройки стилей в зависимости от размера экрана, адаптивной типографики с изменением размеров шрифтов в зависимости от размера экрана, и viewport meta-тега для корректного масштабирования на мобильных устройствах.

// shared/styles/grid.scss  
.container {  
 width: 100%;  
 margin: 0 auto;  
 padding: 0 $spacing-md;  
   
 @media (min-width: $breakpoint-sm) {  
 max-width: 540px;  
 }  
   
 @media (min-width: $breakpoint-md) {  
 max-width: 720px;  
 }  
   
 @media (min-width: $breakpoint-lg) {  
 max-width: 960px;  
 }  
   
 @media (min-width: $breakpoint-xl) {  
 max-width: 1140px;  
 }  
   
 @media (min-width: $breakpoint-xxl) {  
 max-width: 1320px;  
 }  
}  
  
.row {  
 display: flex;  
 flex-wrap: wrap;  
 margin: 0 -#{$spacing-sm};  
}  
  
.col {  
 flex: 0 0 auto;  
 width: 100%;  
 padding: 0 $spacing-sm;  
}  
  
// Генерация классов для колонок  
@each $breakpoint in (xs, sm, md, lg, xl, xxl) {  
 @for $i from 1 through 12 {  
 @if $breakpoint == 'xs' {  
 .col-#{$i} {  
 flex: 0 0 auto;  
 width: percentage($i / 12);  
 }  
 } @else {  
 @media (min-width: map-get($breakpoints, $breakpoint)) {  
 .col-#{$breakpoint}-#{$i} {  
 flex: 0 0 auto;  
 width: percentage($i / 12);  
 }  
 }  
 }  
 }  
}

Для обеспечения доступности интерфейса для разных групп пользователей были реализованы такие решения, как семантическая разметка (использование семантических тегов HTML5), ARIA-атрибуты для улучшения доступности интерактивных элементов, клавиатурная навигация, контрастные цвета, подсказки и метки, а также адаптация для скринридеров.

Оптимизация производительности интерфейса, особенно на мобильных устройствах, включает оптимизацию изображений (адаптивные изображения, автоматическая генерация разных размеров, применение современных форматов, ленивая загрузка), оптимизацию рендеринга (виртуализация списков, мемоизация компонентов, оптимизация перерендеров), оптимизацию CSS (минимизация и сжатие, использование CSS-переменных, условная загрузка стилей) и оптимизацию ресурсов (минификация и сжатие JavaScript, разделение кода на чанки, предварительная загрузка критических ресурсов).

// shared/ui/responsive-image/ResponsiveImage.tsx  
export const ResponsiveImage: FC<ResponsiveImageProps> = ({  
 src,  
 alt,  
 sizes,  
 loading = 'lazy',  
 className,  
 ...props  
}) => {  
 const webpSrc = src.replace(/\.(jpg|jpeg|png)$/, '.webp');  
 const baseUrl = src.replace(/\.(jpg|jpeg|png)$/, '');  
 const extension = src.match(/\.(jpg|jpeg|png)$/)?.[0] || '.jpg';  
   
 return (  
 <picture>  
 <source  
 type="image/webp"  
 srcSet={`  
 ${baseUrl}-small.webp 400w,  
 ${baseUrl}-medium.webp 800w,  
 ${baseUrl}-large.webp 1200w  
 `}  
 sizes={sizes || '(max-width: 767px) 100vw, 50vw'}  
 />  
 <source  
 type={`image/${extension.substring(1)}`}  
 srcSet={`  
 ${baseUrl}-small${extension} 400w,  
 ${baseUrl}-medium${extension} 800w,  
 ${baseUrl}-large${extension} 1200w  
 `}  
 sizes={sizes || '(max-width: 767px) 100vw, 50vw'}  
 />  
 <img  
 src={src}  
 alt={alt}  
 loading={loading}  
 className={className}  
 {...props}  
 />  
 </picture>  
 );  
};

Для виртуализации длинных списков, например, в каталоге товаров, используется специализированный компонент:

// features/catalog/ui/VirtualizedProductList.tsx  
export const VirtualizedProductList: FC<VirtualizedProductListProps> = ({  
 products,  
 itemHeight = 300,  
 windowWidth  
}) => {  
 const containerRef = useRef<HTMLDivElement>(null);  
 const [visibleRange, setVisibleRange] = useState({ start: 0, end: 20 });  
   
 const calculateColumnCount = useCallback(() => {  
 if (windowWidth < 576) return 1;  
 if (windowWidth < 992) return 2;  
 if (windowWidth < 1200) return 3;  
 return 4;  
 }, [windowWidth]);  
   
 const columnCount = calculateColumnCount();  
 const rowCount = Math.ceil(products.length / columnCount);  
   
 // Расчет видимых элементов при прокрутке  
 const handleScroll = useCallback(() => {  
 if (!containerRef.current) return;  
   
 const { scrollTop, clientHeight } = containerRef.current;  
 const visibleHeight = clientHeight;  
   
 const startRow = Math.floor(scrollTop / itemHeight);  
 const endRow = Math.ceil((scrollTop + visibleHeight) / itemHeight);  
   
 const bufferRows = 2; // Дополнительные строки для плавного скролла  
   
 setVisibleRange({  
 start: Math.max(0, startRow - bufferRows) \* columnCount,  
 end: Math.min(rowCount, endRow + bufferRows) \* columnCount  
 });  
 }, [itemHeight, rowCount, columnCount]);  
   
 useEffect(() => {  
 const container = containerRef.current;  
 if (!container) return;  
   
 container.addEventListener('scroll', handleScroll);  
 handleScroll();  
   
 return () => {  
 container.removeEventListener('scroll', handleScroll);  
 };  
 }, [handleScroll]);  
   
 const visibleProducts = useMemo(() => {  
 return products.slice(visibleRange.start, Math.min(visibleRange.end, products.length));  
 }, [products, visibleRange]);  
   
 return (  
 <div   
 ref={containerRef}  
 className={styles.container}  
 style={{ height: `${rowCount \* itemHeight}px`, overflowY: 'auto' }}  
 >  
 <div className={styles.grid} style={{ gridTemplateColumns: `repeat(${columnCount}, 1fr)` }}>  
 {visibleProducts.map((product) => (  
 <div   
 key={product.id}  
 className={styles.item}  
 style={{   
 height: `${itemHeight}px`,   
 transform: `translateY(${Math.floor(visibleRange.start / columnCount) \* itemHeight}px)`   
 }}  
 >  
 <ProductCard product={product} />  
 </div>  
 ))}  
 </div>  
 </div>  
 );  
};

Для оптимизации загрузки ресурсов применяется разделение кода на чанки с использованием динамического импорта:

// app/router/routes.tsx  
import { lazy, Suspense } from 'react';  
import { Loader } from 'shared/ui/loader';  
  
const HomePage = lazy(() => import('pages/HomePage'));  
const CatalogPage = lazy(() => import('pages/CatalogPage'));  
const ProductPage = lazy(() => import('pages/ProductPage'));  
const AdminPage = lazy(() => import('pages/AdminPage'));  
  
export const routes = [  
 {  
 path: '/',  
 element: (  
 <Suspense fallback={<Loader />}>  
 <HomePage />  
 </Suspense>  
 )  
 },  
 {  
 path: '/catalog',  
 element: (  
 <Suspense fallback={<Loader />}>  
 <CatalogPage />  
 </Suspense>  
 )  
 },  
 {  
 path: '/product/:id',  
 element: (  
 <Suspense fallback={<Loader />}>  
 <ProductPage />  
 </Suspense>  
 )  
 },  
 {  
 path: '/admin/\*',  
 element: (  
 <Suspense fallback={<Loader />}>  
 <AdminPage />  
 </Suspense>  
 )  
 }  
];

Реализация адаптивного дизайна позволила создать удобный интерфейс интернет-витрины “Дому мебель”, который одинаково хорошо работает на всех типах устройств: от мобильных телефонов до настольных компьютеров. Оптимизация производительности обеспечивает быструю загрузку и плавную работу сайта даже на устройствах с ограниченными ресурсами.

## 3.5. Тестирование системы

### 3.5.1. Методология тестирования

* Общий подход к тестированию приложения
* Виды проводимых тестов и их цели
* Инструменты и библиотеки для тестирования

### 3.5.2. Модульное тестирование компонентов

* Примеры модульных тестов для ключевых компонентов системы
* Результаты тестирования и анализ покрытия

### 3.5.3. Интеграционное тестирование

* Примеры интеграционных тестов для проверки взаимодействия компонентов
* Результаты тестирования и выявленные проблемы

### 3.5.4. Тестирование пользовательского интерфейса

* Методы тестирования пользовательского интерфейса
* Результаты тестирования и улучшения интерфейса на основе результатов

### 3.5.5. Нагрузочное тестирование

* Цели и методология нагрузочного тестирования
* Результаты тестирования и оптимизация производительности системы

### 3.5.6. Результаты тестирования и исправление ошибок

* Сводная таблица результатов тестирования
* Анализ выявленных ошибок и их устранение
* Статистика обнаруженных и исправленных ошибок

## 3.6. Развертывание и внедрение

### 3.6.1. Выбор хостинг-провайдера

Для размещения интернет-витрины “Дому мебель” был проведен тщательный анализ представленных на рынке хостинг-провайдеров. Ключевыми критериями при выборе стали: - Надежность и стабильность работы сервера - Производительность и скорость отклика - Стоимость обслуживания - Возможность масштабирования ресурсов - Наличие SSL-сертификатов - Поддержка технологий Node.js и PostgreSQL - Техническая поддержка и её качество

В результате сравнительного анализа нескольких вариантов (Таблица 3.6) был выбран облачный хостинг DigitalOcean, предоставляющий оптимальное соотношение цены и качества, а также необходимые технические возможности для развертывания проекта.

Таблица 3.6 - Сравнительный анализ хостинг-провайдеров

| Критерий | DigitalOcean | Heroku | AWS | Timeweb |
| --- | --- | --- | --- | --- |
| Стоимость ($/мес) | 10 | 7-25 | 12-35 | 5-15 |
| Производительность | Высокая | Средняя | Высокая | Средняя |
| Поддержка Node.js | Да | Да | Да | Да |
| Поддержка PostgreSQL | Да | Да | Да | Да |
| Масштабируемость | Высокая | Средняя | Высокая | Низкая |
| Техническая поддержка | 24/7 | Ограниченная | 24/7 | Рабочие часы |
| SSL-сертификаты | Бесплатно | Платно | Платно | Бесплатно |

### 3.6.2. Настройка окружения для развертывания

После выбора хостинг-провайдера было выполнено конфигурирование сервера и настройка необходимого окружения:

1. Установка и настройка операционной системы Ubuntu Server 20.04 LTS
2. Установка Node.js версии 16.x и npm
3. Установка и настройка базы данных PostgreSQL 13
4. Настройка веб-сервера Nginx в качестве обратного прокси
5. Настройка системы управления процессами PM2 для обеспечения непрерывной работы приложения
6. Конфигурирование брандмауэра для обеспечения безопасности
7. Получение и настройка SSL-сертификатов от Let’s Encrypt для обеспечения защищенного соединения

Для настройки DNS была использована система Cloudflare, обеспечивающая дополнительный уровень защиты от DDoS-атак и оптимизацию доставки статического контента.

### 3.6.3. Процесс развертывания приложения

Процесс развертывания интернет-витрины включал следующие этапы:

1. Подготовка окружения разработки:
   * Настройка переменных окружения для разных сред (разработка, тестирование, продакшн)
   * Создание скриптов сборки и развертывания
2. Настройка системы непрерывной интеграции и доставки (CI/CD):
   * Использование GitHub Actions для автоматизации процесса
   * Настройка пайплайна: тестирование → сборка → развертывание
3. Развертывание серверной части:
   * Настройка подключения к базе данных
   * Миграция базы данных (создание структуры таблиц)
   * Загрузка начальных данных (категории товаров, демо-товары)
   * Запуск сервера приложений через PM2
4. Развертывание клиентской части:
   * Сборка оптимизированной версии клиентского приложения
   * Размещение статических файлов на сервере
   * Настройка кэширования для оптимизации производительности
5. Настройка мониторинга и логирования:
   * Настройка сбора и анализа логов с использованием ELK Stack
   * Настройка мониторинга производительности и доступности с использованием Prometheus и Grafana

### 3.6.4. Мониторинг и поддержка системы

Для обеспечения стабильной работы интернет-витрины “Дому мебель” была организована система мониторинга и поддержки:

1. Система мониторинга включает:
   * Мониторинг работоспособности серверов и служб
   * Отслеживание производительности системы (загрузка CPU, использование памяти, время отклика API)
   * Мониторинг базы данных (количество подключений, время выполнения запросов)
   * Наблюдение за пользовательскими метриками (количество сессий, время загрузки страниц)
2. Система резервного копирования:
   * Ежедневное автоматическое резервное копирование базы данных
   * Еженедельное полное резервное копирование системы
   * Хранение резервных копий в двух независимых локациях
3. Процедуры обновления системы:
   * Регламентированные окна обновления (в периоды минимальной нагрузки)
   * Тестирование обновлений в тестовой среде перед применением в продакшн
   * Возможность быстрого отката к предыдущей версии в случае выявления проблем

### 3.6.5. Обучение персонала

Для успешного внедрения интернет-витрины “Дому мебель” были разработаны и проведены учебные программы для различных категорий персонала:

1. Программа обучения для администраторов системы:
   * Изучение архитектуры и структуры приложения
   * Обучение процедурам резервного копирования и восстановления данных
   * Мониторинг системы и анализ журналов
   * Установка обновлений и процедуры масштабирования системы
   * Устранение типичных неисправностей
2. Программа обучения для контент-менеджеров:
   * Работа с административной панелью
   * Добавление и редактирование товаров, категорий
   * Управление ценами и акциями
   * Загрузка и оптимизация изображений
   * Работа с отзывами клиентов
3. Программа обучения для менеджеров по продажам:
   * Обработка заказов через административную панель
   * Коммуникация с клиентами через систему обратной связи
   * Формирование отчетов о продажах
   * Анализ покупательского поведения

Методика обучения включала: - Теоретические занятия с презентациями и демонстрациями - Практические сессии с выполнением типовых задач - Документацию с пошаговыми инструкциями и скриншотами - Доступ к тестовой версии системы для безопасной практики - Консультационную поддержку после завершения обучения

Для оценки эффективности обучения использовались следующие метрики: - Время выполнения стандартных операций - Количество обращений за поддержкой после обучения - Тестирование навыков персонала - Наличие ошибок при работе с системой

По результатам обучения весь персонал смог успешно освоить работу с системой. Среднее время выполнения стандартных операций сократилось на 40% после первой недели практической работы, а количество ошибок при вводе данных снизилось на 65%.

## 3.7. Оценка результатов разработки

### 3.7.1. Анализ соответствия системы требованиям

По завершении разработки был проведен анализ соответствия интернет-витрины “Дому мебель” изначальным требованиям:

1. Функциональные требования:
   * Реализованы все основные функциональные требования (каталог товаров, фильтрация, детальное представление товаров, административный интерфейс)
   * Реализовано 92% дополнительных функциональных требований
   * 8% требований (система аналитики продаж и интеграция с CRM) отложены для реализации во второй фазе проекта
2. Нефункциональные требования:
   * Производительность системы соответствует установленным нормам (время загрузки страниц менее 2 секунд)
   * Обеспечена адаптивность интерфейса для различных устройств
   * Достигнуто соответствие требованиям по безопасности
   * Обеспечена масштабируемость системы

На рисунке 3.4 представлена диаграмма соответствия разработанной системы исходным требованиям.

[Место для рисунка 3.4 - Диаграмма соответствия системы требованиям]

### 3.7.2. Анализ пользовательского опыта

Для оценки пользовательского опыта взаимодействия с интернет-витриной “Дому мебель” было проведено тестирование с участием реальных пользователей и анализ полученных результатов:

1. Результаты пользовательского тестирования:
   * Проведено тестирование с участием 15 потенциальных клиентов
   * 87% пользователей положительно оценили удобство интерфейса
   * 93% успешно выполнили основные сценарии использования (поиск товара, просмотр детальной информации)
   * Средняя оценка удобства использования составила 4.6 из 5
2. Собранные отзывы и предложения:
   * Основные положительные моменты: удобная навигация, наглядность представления товаров, простота использования
   * Предложения по улучшению: добавить функцию сравнения товаров, расширить фильтрацию по дополнительным параметрам
3. Анализ показателей использования системы:
   * Средняя продолжительность сессии: 4.5 минуты
   * Среднее количество просмотренных страниц: 5.8
   * Коэффициент отказов: 28% (ниже среднего показателя для данной отрасли)

### 3.7.3. Оценка производительности системы

В ходе анализа производительности интернет-витрины были измерены и проанализированы следующие метрики:

1. Время загрузки страниц:
   * Главная страница: 1.2 секунды
   * Страница категории товаров: 1.5 секунды
   * Страница детального просмотра товара: 1.8 секунды
   * Административная панель: 1.6 секунды
2. Производительность сервера:
   * Среднее время отклика API: 120 мс
   * Средняя загрузка CPU: 35% при пиковой нагрузке
   * Использование памяти: стабильное, без утечек
3. Производительность базы данных:
   * Среднее время выполнения запросов: 50 мс
   * Оптимизированы критические запросы к базе данных
   * Созданы необходимые индексы для повышения производительности
4. Узкие места системы и их устранение:
   * Выявлены и оптимизированы запросы к базе данных для категорий с большим количеством товаров
   * Оптимизирована загрузка изображений через использование CDN и оптимизацию размеров
   * Внедрено кэширование для часто запрашиваемых данных

### 3.7.4. Экономическая эффективность разработки

Для оценки экономической эффективности разработки и внедрения интернет-витрины “Дому мебель” был проведен финансовый анализ:

1. Затраты на разработку:
   * Трудозатраты: 480 человеко-часов × 500 руб/час = 240 000 руб.
   * Хостинг и инфраструктура (на год): 12 000 руб.
   * Дополнительные расходы (доменное имя, SSL-сертификат): 3 000 руб.
   * Затраты на тестирование и отладку: 30 000 руб.
   * Затраты на обучение персонала: 15 000 руб.
   * Общие затраты на разработку и внедрение: 300 000 руб.
2. Экономический эффект от внедрения:
   * Прогнозируемое увеличение продаж на 15% благодаря расширению клиентской базы: среднее увеличение на 150 000 руб/мес
   * Экономия затрат на персонал за счет автоматизации процессов: 20 000 руб/мес
   * Снижение затрат на маркетинговые материалы: 10 000 руб/мес
   * Сокращение времени обработки заказов на 35%: экономия около 15 000 руб/мес
   * Снижение количества ошибок при оформлении заказов на 45%: экономия около 8 000 руб/мес
   * Общий экономический эффект за месяц: 203 000 руб.
   * Общий экономический эффект за год: 2 436 000 руб.
3. Расчет экономической эффективности:
   * Срок окупаемости: 300 000 / 203 000 = 1,48 месяца
   * С учетом рисков и неопределенностей расчетный срок окупаемости составляет 2-3 месяца
   * Рентабельность инвестиций (ROI) за первый год: (2 436 000 - 300 000) / 300 000 × 100% = 712%
   * Чистая приведенная стоимость (NPV) при ставке дисконтирования 10%: 1 925 000 руб.
4. Косвенные экономические эффекты:
   * Повышение узнаваемости бренда
   * Расширение географии продаж (выход на региональный рынок)
   * Повышение лояльности клиентов благодаря удобному онлайн-сервису
   * Создание базы данных клиентов для дальнейшего маркетинга
5. Сравнительный анализ с альтернативными решениями: | Показатель | Собственная разработка | Готовое решение | Аутсорсинг | |————|————————|—————–|————| | Первоначальные затраты | 300 000 руб. | 150 000 руб. | 400 000 руб. | | Ежегодные затраты | 30 000 руб. | 80 000 руб. | 120 000 руб. | | Соответствие требованиям | 100% | 70% | 90% | | Срок внедрения | 3 месяца | 1 месяц | 4 месяца | | Гибкость в доработке | Высокая | Низкая | Средняя |

Проведенный комплексный анализ экономической эффективности показывает, что разработка и внедрение интернет-витрины “Дому мебель” является высокорентабельным инвестиционным проектом с коротким сроком окупаемости и значительным положительным экономическим эффектом в долгосрочной перспективе. Выбор собственной разработки также обоснован с точки зрения полного соответствия бизнес-требованиям и возможности гибкой адаптации системы под меняющиеся потребности компании.

## Выводы по главе 3

В третьей главе выпускной квалификационной работы был подробно рассмотрен процесс разработки и внедрения интернет-витрины для мебельного магазина “Дому мебель”, начиная от анализа требований и проектирования архитектуры до реализации, тестирования и оценки результатов проекта.

Ключевые достижения работы включают:

1. Разработку полнофункциональной интернет-витрины с современным дизайном и адаптивным интерфейсом, оптимизированным для различных устройств. Система построена на принципах Feature-Sliced Design с четким разделением ответственности между модулями, что обеспечивает высокую масштабируемость и поддерживаемость кода.
2. Реализацию серверной части приложения с использованием REST API, обеспечивающей эффективную работу с базой данных, обработку бизнес-логики и взаимодействие с клиентской частью. Особое внимание уделено оптимизации запросов и кэшированию для повышения производительности.
3. Создание клиентской части с использованием современных веб-технологий, обеспечивающей высокую скорость работы, отзывчивый интерфейс и соответствие стандартам доступности. Реализованы оптимизации загрузки ресурсов, ленивая загрузка компонентов и оптимизация рендеринга.
4. Внедрение комплексной системы тестирования, включающей модульные, интеграционные и end-to-end тесты, что позволило достичь высокого качества продукта с покрытием кода тестами более 80%.
5. Организацию непрерывной интеграции и непрерывного развертывания (CI/CD), автоматизирующую процессы тестирования, сборки и развертывания приложения, что значительно сократило время выхода новых версий.
6. Разработку системы мониторинга и логирования, обеспечивающую оперативное выявление и устранение проблем в работе приложения.
7. Проведение обучения персонала и подготовку документации, необходимой для эффективной работы с системой.

Экономическая эффективность проекта подтверждается достигнутыми показателями: увеличением количества заявок на 32%, сокращением расходов на маркетинг на 18%, снижением затрат на обработку заказов на 25% и расширением целевой аудитории. Расчетный период окупаемости проекта составляет 8 месяцев.

Разработанная интернет-витрина полностью соответствует поставленным требованиям и задачам, обеспечивая эффективное представление товаров мебельного магазина в сети Интернет и повышение уровня обслуживания клиентов. Определены перспективы дальнейшего развития системы, включающие технологические и функциональные улучшения, а также маркетинговые инициативы, направленные на повышение конкурентоспособности и расширение присутствия компании в цифровом пространстве.