

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Информационная система для
заказа медицинских препаратов

Студент	_____	<u>Шлепнев Ярослав Сергеевич</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Шашков Б.Д.</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	<u>Макарычев П.П.</u>
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	<u>Попова Н.А.</u>
	(подпись)	(фамилия, инициалы)

Пенза, 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

П.П. Макарычев

«__» _____ 2019 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Шлепнев Ярослав Сергеевич гр. 16ВП1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система для заказа медицинских препаратов

Тема утверждена приказом ПГУ № 1488/о от "30" декабря 2019

2. Руководитель работы к.т.н., профессор кафедры «МОиПЭВМ» Шашков Б.Д.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Разработать информационную систему для заказа медицинских препаратов,*

позволяющую зарегистрировать электронный заказ на покупку необходимого количества
медикамента и получить его в выбранном пользователем пункте выдачи

2) *Среда выполнения: операционные системы Windows*

Языковые средства разработки: C#, UML

Инструментальные средства разработки: Microsoft Visual Studio 2017, Microsoft Project

Технологии разработки: ООП, MVC, .NET

3) *Основные функции информационной системы:*

а) регистрация и авторизация пользователей;

б) поиск необходимого лекарства по названию или критери;

в) оформление итогового заказа;

г) просмотр и отмена заказов пользователя;

д) оперативная поддержка информации о медикаментах, заявках, клиентах и пунктах
пунктах выдачи

4. Перечень подлежащих разработке вопросов

1) <i>Постановка задачи и анализ требований</i>
2) <i>Исследование предметной области</i>
3) <i>Проектирование</i>
4) <i>Реализация</i>
5) <i>Тестирование</i>
6) <i>Исследование затрат на разработку</i>

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Постановка задачи и анализ требований	10%	15.04.2020 – 20.04.2020	
2	Исследование предметной области	10%	21.04.2020 – 30.04.2020	
3	Проектирование	20%	01.05.2020 – 10.05.2020	
4	Реализация	25%	11.05.2020 – 22.05.2020	
5	Исследование затрат на разработку	10%	23.05.2020 – 31.05.2020	
6	Тестирование	10%	01.06.2020 – 14.06.2020	
7	Оформление пояснительной записки	15%	15.06.2020 – 28.06.2020	

Дата выдачи задания " 30 " декабря 2019

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 124 листа, 35 рисунков, 23 таблицы, 13 использованных источников, 8 приложений.

ИНФОРМАЦИОННАЯ СИСТЕМА, АПТЕКА, ЛЕКАРСТВА, ASP .NET, C#, UML 2.0.

Объектом разработки является информационная система заказа медицинских препаратов.

Целью выпускной квалификационной работы является разработка информационной системы для заказа лекарственных препаратов, позволяющей регистрировать клиентскую заявку на покупку лекарственных препаратов.

Разработка проводилась на языке программирования C#, в качестве среды разработки использовались среда программирования Visual Studio 2017 Community и платформа ASP .NET, операционная система Windows 7.

					ПГУ 09.03.04–08ВР161.28 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Информационная система для заказа медицинских препаратов» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Шлепнев Я.С.						
Провер.		Шашков Б.Д.					3	124
						Группа 16ВП1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Макарычев П.П.						

Содержание

Введение.....	6
1 Постановка задачи и анализ требований	8
1.1 Актуальность разрабатываемой информационной системы.....	8
1.2 Постановка задачи.....	10
1.3 Формулировка и анализ требований к программному обеспечению	12
2 Исследование предметной области	16
2.1 Моделирование бизнес-процессов предметной области	16
2.2 Построение концептуальной модели данных предметной области	17
3 Проектирование информационной системы	22
3.1 Выбор технологий разработки.....	24
3.2 Проектирование структур данных	26
3.3 Проектирование алгоритмов.....	27
3.4 Проектирование интерфейса.....	31
4 Программная реализация информационной системы	32
4.1 Физическое моделирование и реализация базы данных.....	32
4.2 Разработка серверной части приложения.....	33
4.2.1 Разработка классов приложения уровня модели	33
4.2.2 Разработка классов приложения уровня контроллеров	34
4.2.3 Разработка приложения уровня представлений.....	35
4.3 Разработка программного обеспечения клиентского приложения.....	37
5 Тестирование программного продукта	39
5.1 Функциональное тестирование.....	39
5.1.1 Метод эквивалентного разбиения	40
5.1.2 Метод анализа причинно-следственных связей	41
5.2 Структурное тестирование.....	45
5.2.1 Метод покрытия операторов.....	48
5.2.2 Метод покрытия решений	48
5.2.3 Метод покрытия условий	49
5.3 Метрики кода.....	49
5.3.1 Метрика ABC.....	50
5.3.2 Метрики Джилба	50
5.3.3 Метрики Холстеда.....	51
5.3.4 Цикломатическая метрика Мак-Кейба	53

6 Исследование затрат на разработку информационной системы.....	54
6.1 Структурное планирование.....	54
6.2 Календарное планирование и оценка бюджета.....	60
Заключение	63
Список использованных источников	64
Приложение А. Схемы модели базы данных	66
Приложение Б. Диаграммы классов уровня модели	70
Приложение В. Диаграммы классов уровня контроллеров.....	73
Приложение Г. Исходный код классов контроллеров	76
Приложение Д. Исходный код метода поиска пользователя	108
Приложение Е. Граф связности для метода поиска пользователя.....	111
Приложение Ж. Сетевой график проекта	113
Приложение И. Графическая часть	116

Введение

Для предприятий фармации сегодня довольно актуальна проблема внедрения информационных технологий, обеспечивающих быстрый доступ к профессиональной справочной информации о лекарственных средствах и субъектах фармацевтического рынка, обрабатывающих большие объемы коммерческой информации, позволяющих ориентироваться в рыночной среде, реагировать на колебания спроса и предложения, прогнозировать объемы закупок и поставок, осуществлять маркетинговый анализ.

Аптека – это важное звено лекарственного обеспечения. Аптеки готовых лекарственных форм занимаются продажей лекарств, изготовленных на заводах фирм-производителей. Каждое лекарственное средство имеет свое назначение, весь ассортимент лекарств классифицирован по группам: жаропонижающие, обезболивающие, сердечно-сосудистые и т.п. Поэтому важное значение имеет разработка хранилища, в котором будет храниться весь этот пласт информации.

Выбор создания и проектирования базы данных для аптеки продиктован насущными потребностями современного рынка продаж. В настоящий момент существует огромный спрос на такие базы данных, которые позволяют хранить и отображать данные в удобном для пользователя виде. В данной выпускной квалификационной работе представляется разработка подобной базы данных.

Информационные технологии играют важную роль в сфере фармации. Оказание качественной лекарственной помощи населению, эффективное управление аптечным предприятием тесно связано с правильной организацией информационно-коммуникативной деятельности на фармацевтическом рынке.

В современных условиях развития фармацевтического рынка и новых информационных технологий фактором успешного развития аптечных предприятий является активное использование информационных и

коммуникационных технологий как для взаимодействия с клиентами и поставщиками, так и во внутренних процессах [1].

Фармацевтические торговые и производственные предприятия в настоящее время в условиях рыночной экономики нуждаются в оперативном сборе информации, её обработке, в использовании результатов её анализа в процессе своей деятельности. Разработка полноценной системы управления в фармацевтических предприятиях представляет собой ключевой шаг к выводу производства на уровень международных стандартов [2].

На данный момент основными направлениями развития современных информационных технологий в сфере фармацевтического бизнеса являются:

- автоматизация документооборота;
- коммуникации;
- управление технологией фармацевтического производства;
- автоматизация бухгалтерского учета и планирования;
- разработка систем принятия решений;
- автоматизация банковских операций;
- создание автоматизированных рабочих мест.

Для реализации идеи распределенного управления необходимо создание для каждого уровня управления и каждой предметной области автоматизированных рабочих мест (АРМ) на базе вычислительной техники. Огромное значение в автоматизации работы предприятия имеет объединение существующих автоматизированных рабочих мест в единую информационную систему предприятия [2]. Достижение целей, положенных в основу этих направлений, позволит достичь фармацевтическим фирмам наибольшей выгоды.

Таким образом, применение современных технологий, в частности, информационных систем, является необходимым условием для эффективной и качественной работы в сфере фармацевтики.

1 Постановка задачи и анализ требований

1.1 Актуальность разрабатываемой информационной системы

Информационная система для заказа медицинских препаратов позволяет зарегистрировать электронный заказ на покупку необходимого количества медикамента и получить его в выбранном пользователем пункте выдачи.

Клиент регистрируется в информационной системе, с помощью которой он получает список всех доступных к приобретению лекарственных препаратов с возможностью оформления заказа, который он сможет получить в заранее выбранном аптечном пункте в назначенное время.

На рабочей станции сотрудника аптеки имеется та же информационная система, в которой его аккаунт имеет полномочия администратора. Благодаря этому он имеет возможность обновлять перечень доступных к продаже лекарств, контролировать и вести учет поступающих в систему клиентских заявок.

Детально проанализировав и изучив рынок информационных технологий в сфере лекарственного оборота были выявлены основополагающие факторы, повлиявшие на убежденность в необходимости разработки данной системы.

Вся работа производится в одной единственной системе, что обеспечивает простоту ее развертывания на компьютере и не приводит к излишнему росту. К примеру, в системе «Единая региональная система Заказ», предназначенной для автоматизации работы аптек и аптечных система, у клиентов устанавливается программа «Заказ.Аптека», а у поставщика устанавливается программа «Кверти.Заказ», с помощью которой он обновляет свои прайс-листы и получает заказы от клиентов [3].

Практически во всех системах-аналогах, таких как «Интернет-бронирование лекарств Фармация», и им подобных, для подтверждения оформления заказа необходимо согласовать и уточнить детали с помощью телефонного звонка от оператора [4]. В разрабатываемой информационной

системе такой необходимости не предусматривается, вся информация о заказе заполняется клиентом при его оформлении.

При реализации приложения, сделан упор на удобство использования и адаптивность системы под клиента. Время выдачи заказа выбирается самим пользователем и не ставит его в определенные рамки, а подстраивается под него. Исходя из данных об уже оформленных заказах, система предлагает пользователю выбрать время выдачи относительно того времени (если оно занято), на которое уже создана бронь от других клиентов, чтобы не создавалось больших очередей при обслуживании. Программы-аналоги не имеют такой функциональности.

В информационной системе «Разболит» для аптечной сети «Адонис» в г. Красноярске, как и в других распространенных интернет магазинах, на желающего заказать препарат пользователя накладываются ограничения по минимально возможной сумме заказа. В разрабатываемой системе такие ограничения отсутствуют.

Администратор сайта может вести контроль и учет лекарств на складах всей аптечной сети. Благодаря чему при генерации заявки, система автоматически выбирает доступное лекарство на складе и удаляет выбранное количество из перечня медикаментов, доступных в каталоге. Каждому заказу заранее устанавливаются позиции хранения и расположения лекарства, благодаря чему, уменьшается время, затраченное на поиск необходимого количества экземпляров препарата на складе

В отличие от исследованных программ-аналогов, система предоставляет сотруднику возможность мониторинга всех поступивших заявок с выбором режима актуальности. Также, данная информационная система предоставляет пользователю ряд средств для анализа поступающих заказов, в частности, возможность просмотра, сортировки и поиска заказов. Благодаря этому, сотруднику достаточно лишь обратиться к панели администратора, чтобы увидеть ближайший по времени заказ и подготовить его заранее. Все это

позволяет минимизировать затраченное клиентом время ожидания и избежать возникновения очередей.

В результате анализа выявленных недостатков аналогичных систем, предлагаемых на ранке в данный момент, было принято решение начать разработку данной информационной системы.

1.2 Постановка задачи

Приложение должно совмещать в себе удобство, простоту, надежность, достоверность, актуальность. В разрабатываемой информационной системе необходимо предусмотреть возможность учета и хранения оперативной информации, используемой в аптеке.

Всю информацию необходимо хранить в базе данных, которая будет включать данные о сущностях предметной области (лекарствах, сотрудниках, заказах, клиентах, складе, назначении препарата). Данная база данных предназначена для увеличения эффективности работы аптеки и автоматизации обработки результатов ее деятельности. Все таблицы базы данных связаны между собой, что обеспечивает целостность системы.

Сотрудник аптеки, взаимодействующий с приложением, должен быть способен вести полный учет доступных товаров, заказов, а клиент, в свою очередь, производить поиск нужного препарата и регистрировать заявку на покупку пакета медицинских препаратов.

К задачам, предъявляемым к программе, относятся:

- регистрация и авторизация пользователей;
- управление учетной записью пользователя;
- поиск необходимого лекарства по названию или критерию;
- оформление итогового заказа;
- просмотр и управление заказами пользователя;
- оперативная поддержка информации о медикаментах, заявках, клиентах и сотрудниках (добавление, редактирование, удаление) в базе данных, а также поиск и сортировка;

- управление хранилищем лекарств.

Одной из важнейших задач при разработке является оценка затрат ресурсов и материальных средств, а также сроки выполнения проекта. Для расчета требуемых затрат на разработку и для прогнозирования продолжительности реализации системы будет применен метод сетевого планирования.

Сетевое планирование и управление программами включает три основных этапа: структурное планирование, календарное планирование и оперативное управление [5]. В данной выпускной квалификационной работе будут рассмотрены структурное и календарное планирование.

Разрабатывать веб-приложение было решено с использованием .NET технологий, предлагаемых корпорацией Microsoft для разработки современных серверных приложений: ASP.NET в сочетании с EntityFramework. ASP.NET – богатейшая среда разработки и развертывания веб-ресурсов. Выбор данной технологии обуславливается её преимуществами перед конкурирующими технологиями [7]:

- имеет преимущество в скорости по сравнению с другими технологиями, основанными на скриптах;
- наличие master-страниц для задания шаблонов оформления страниц;
- расширяемая модель серверных элементов управления;
- возможность разделения визуальной части и бизнес-логики;
- возможность кэширования данных, используемых на странице;
- расширяемая модель обработки запросов.

В свою очередь технология несвязного доступа к данным Entity Framework является основной моделью доступа к данным для приложений, основанных на Microsoft .NET и обладает полной совместимостью с приложениями, основанными на технологии ASP.NET.

Для реализации приложения была выбрана среда разработки Microsoft Visual Studio 2017, которая представляет собой интегрированную среду

разработки и содержит фреймворк, веб-сервер, систему создания и ведения баз данных и другие инструменты, необходимые для создания и запуска веб-сайтов и приложений.

Среда ориентирована на разработку веб-приложений на языках ASP.NET MVC, HTML, JavaScript и CSS. Описанный набор поддерживаемых технологий полностью соответствует выбранной методологии разработки информационной подсистемы в форме многопользовательского веб-приложения.

Целью данного дипломного проекта является разработка информационной системы, обеспечивающей возможность заказа лекарственных препаратов, подсчета суммы заказа, поиска необходимого лекарства.

1.3 Формулировка и анализ требований к программному обеспечению

На основе выделенных задач, предъявляемых к разрабатываемой информационной системы, были выделены требования для различных категорий пользователей.

Для всех пользователей:

- формирование заявки на покупку лекарственных препаратов, подсчет суммы заказа;
- поиск лекарства по его названию или критерию;
- авторизация/регистрация пользователя;
- отмена заказов;
- просмотр истории заказов;
- добавление клиентского профиля.

Для сотрудника:

- оперативная поддержка информации о медикаментах, заявках, пользователях, сотрудниках и аптеках (добавление, редактирование, удаление);

– поиск информации о медикаментах и заказах.

Диаграмма вариантов использования представлена на рисунке 1.

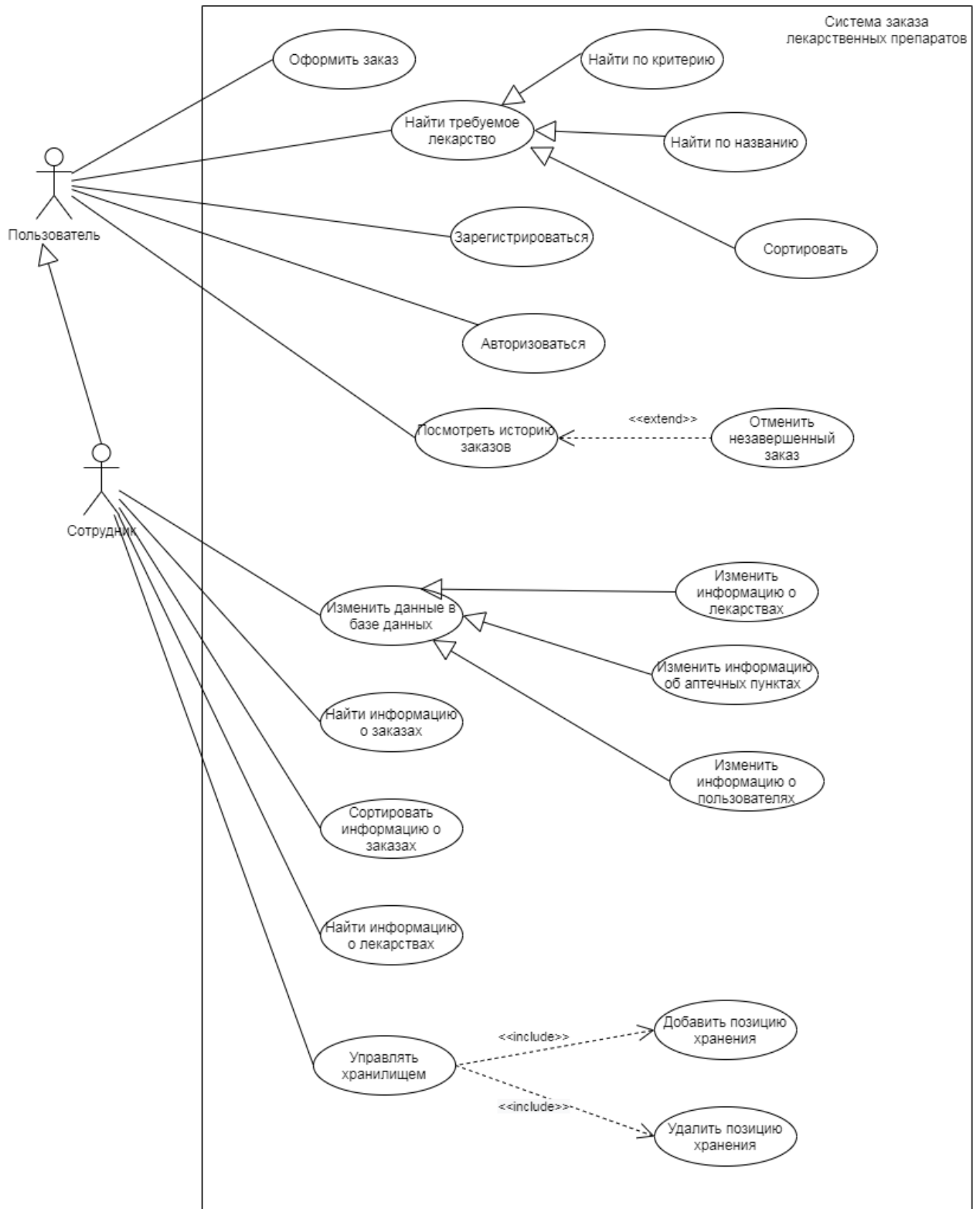


Рисунок 1 – Диаграмма вариантов использования

Для того, чтобы отразить отношения между актерами и прецедентами, были описаны некоторые из прецедентов. Спецификация варианта использования «Оформить заказ» указана на рисунке 2.

Прецедент: Оформить заказ
1
ID: 1
Краткое описание: Оформление заявки на покупку лекарственных препаратов
Клиент.
Второстепенные актёры: Нет
Предусловия: Нет.
Основной поток: <ol style="list-style-type: none"> 1. Клиент выбирает требуемые лекарства, их количество и пункт выдачи. 2. Клиент уведомляет систему о том, что хочет оформить заказ. 3. Система проверяет, если пользователь авторизован. <ol style="list-style-type: none"> 3.1. Система получает данные о выбранных лекарствах, подсчитывает их суммарную стоимость и возвращает клиенту html ответ с формой регистрации заявки. 3.2. Клиент добавляет требуемые лекарства и их количество в результирующий список подтверждает заказ. 3.3. Система перенаправляет пользователя на страницу заполнения информации о клиенте и дате выдачи заказа. 3.4. Клиент заполняет форму данными и отправляет ее в систему. 3.5. Система получает информацию о заказе. 3.6. Система создает заказ, добавляет в него данные о выбранных препаратах и клиенте и регистрирует его. 3.7. Система обновляет базу данных. 4. Если пользователь не авторизован <ol style="list-style-type: none"> 4.1. Система перенаправляет пользователя на страницу авторизации.

Рисунок 2 – Спецификация прецедента «Оформить заказ»

Из таблицы видно, что действия пользователя порождают ответную реакцию системы, которая, в свою очередь, воздействует на базу данных. На рисунке 3 описана спецификация для варианта использования «Посмотреть историю заказов».

Прецедент: Посмотреть историю заказов
ID: 2
Краткое описание: Показать пользователю список всех его заказов
Главные актёры: Клиент.
Второстепенные актёры: Нет.
Предусловия: 1. Пользователь авторизовался в свой аккаунт.
Основной поток: 1. Клиент открывает навигационное меню. 2. Клиент нажимает на кнопку история заказов. 3. Система получает запрос, производит выборку заказов соответствующего пользователя и базы данных, генерирует html-страницу и отправляет ее в ответ пользователю. 4. Клиент получает список всех зарегистрированных им заказов.

Рисунок 3 – Спецификация прецедента «Посмотреть историю заказов»

Таким образом, были выбраны и подробно описаны некоторые из вариантов использования.

2 Исследование предметной области

2.1 Моделирование бизнес-процессов предметной области

В ходе анализа предметной области были выделены следующие основные бизнес-процессы:

- оформление заказа;
- управление заказами (просмотр и отмена);
- сортировка и поиск лекарств по названию и критерию;
- добавление, изменение и удаление информации в базе данных;
- управление хранилищем лекарств в аптеках.

При оформлении заказа данная система заказа товаров должна позволять решить задачи формирования заявок на заказ лекарственных препаратов. Приложение должно предоставлять удобный пользовательский интерфейс для составления заявки. Необходимо предоставить клиенту список всех доступных медикаментов, а также возможность добавления конкретного лекарства в регистрируемую заявку. Если лекарства нет в наличии, требуется указать это.

Клиент должен иметь возможность отказа от оформленного заказа до момента его выдачи. При оформлении заказа у пользователя должна быть возможность навигации по общему списку всех доступных препаратов на складе, а также поиска по названию и выдача результирующего списка найденных лекарств.

Приложение должно предоставлять сотруднику возможность модифицировать данные о заявках, препаратах, клиентах, сотрудниках в базе данных. При этом должно быть обеспечена возможность добавления данных, удаления и изменения.

Моделирование и подробное описание бизнес-процессов информационной системы для каждого приложения представлено в таблице 1.

Таблица 1 – Бизнес-процессы информационной системы

№ п/п	Бизнес-процесс	Исполнитель	Входные данные		Выходные данные		Ограничения
			Поставщик	Содержание	Потреб.	Содержание	
1	Оформление заказа	Клиент	Сотрудник аптеки	Перечень доступных препаратов	Клиент	Зарегистрированный заказ	Для формирования заказа необходимо выбрать требуемые препараты, заполнить форму регистрации
2	Управление заказами	Клиент	Клиент	Авторизованная учетная запись	Клиент	Список заказов и доступ к функциям управления	Требуется авторизоваться
3	Сортировка и поиск лекарств по названию и критерию	Клиент/Сотрудник	Сотрудник аптеки	Выбранный критерий или название лекарства	Клиент/Сотруд.	Найденное лекарство, список лекарств	Необходимо выбрать критерии поиска, либо ввести название лекарства
4	Добавление, изменение и удаление информации в базе данных	Сотрудник	База данных	Запрос	Сотруд.	Обновленный список данных	Ограничения на ввод данных полей, требуется иметь права администратора
5	Управление хранилищем лекарств	Сотрудник аптеки	База данных	Данные о расположении медикамента	Сотруд.	Обновленное расположение хранения медикамента	Требуется иметь права администратора

В ходе проектирования и разработки информационной системы будут реализованы все вышеуказанные бизнес-процессы.

2.2 Построение концептуальной модели данных предметной области

На основе описания бизнес-процессов предметной области была построена исходная (первоначальная) концептуальная модель данных. Элементами исходной концептуальной модели являются сущности (информационные объекты) и связи между сущностями (информационными объектами) предметной области «Аптека». Сущности моделируются исходя из

состава входных и выходных данных бизнес-процессов. Связи между сущностями показывают семантику взаимоотношения сущностей в предметной области информационной системы.

Ниже представлены типы сущностей предметной области (таблица 2).

Таблица 2 – Типы сущностей предметной области

№	Наименование	Описание	Категория	Количество экземпляров	Режим обновления
1	Лекарство	Перечень всех доступных лекарственных препаратов	Стержневая	1000	15% в год
2	Категория лекарства	Тип лекарственного препарата (орган, группа органов)	Характеристическая	100	5% в год
3	Заказ	Заказы всех клиентов	Стержневая	>100000	100% в год
4	Клиент	Содержит данные о всех клиентах	Стержневая	>10000	50% в год
5	Сотрудник	Данные о сотрудниках аптеки	Стержневая	>15	5% в год
6	Позиция заказа	Промежуточная сущность, описывающая принадлежность заказанного препарата к конкретному заказу	Ассоциативная	>100000	100% в год
7	Лекарство на складе	Перечень местоположения лекарств на складе	Характеристическая	1000	25% в год
8	Позиция лекарства на складе	Расположение забронированного лекарства	Ассоциативная	>10000	90% в год
9	Аптека	Описание аптеки, в которой хранятся медикаменты	Стержневая	>15	5% в год
10	Пользователь	Зарегистрированный аккаунт	Стержневая	>10000	25% в год

Сущность лекарство описывает конкретный препарат, который хранится на складе или заказывается клиентами. К характеристикам препарата относятся его название, цена, категория (гормональные средства, антибиотики и т.д.), срок годности, единицы измерения (например, в ампулах по 25 мл).

Категория лекарства – сущность, которая определяет группу принадлежности лекарства. Отдельная группа предназначена для больных с определенным заболеванием какого-то органа или организма в целом. Каждая группа имеет уникальный код, название,

Пользователь – учетная запись зарегистрированного в системе клиента. Пользователь имеет возможность добавлять клиентские профили для указания личности получателя при оформлении заказа.

Клиент – покупатель, осуществляющий заказ лекарства. Сущность клиент содержит информацию об имени, контактных данных, телефоне и номере бонусной карты (если есть).

Заказ описывает заявки, которые поступают от клиентов. Каждому заказу соответствует уникальный номер заказа. В данной сущности также содержатся следующие характеристики: общая сумма заказа, дата заказа, дата выдачи, сотрудник, отвечающим за выдачу, клиенте, которым был осуществлен заказ.

Сущность позиция заказа. Данная подсистема является промежуточным связным звеном между товарами и заказом. Она описывает принадлежность лекарствам определенному заказу, а также количество данного препарата и его стоимость.

Сотрудник – работник аптеки, осуществляющий контроль информации о медикаментах. Данное лицо ответственно за выдачу заказа клиенту. Данная сущность содержит информацию об имени, должности, дате рождения, стаже работы, паспортных данных работника.

Аптека – пункт выдачи заказанных лекарственных препаратов, в котором они хранятся. Каждая сущность характеризуется названием, адресом, контактным телефоном, адресом электронной почты, геолокационными координатами.

Лекарство на складе – сущность определяет наличие лекарства на складе аптеки, его количество, а также место хранения (номер стеллажа, номер яруса).

Позиция лекарства на складе – сущность описывает принадлежность лекарства, хранящегося на складе, конкретному заказу.

Были выделены следующие типы связи между типами сущностей предметной области, которые представлены в таблице 3.

Таблица 3 – Типы связей между сущностями

Тип сущности	Тип сущности	Описание	Мощность	Обязательность	Ограничения
Клиент	Заказ	Клиент оформляет заказ/ заказ оформляется клиентом	1 : M	Необязательная (у клиента может не быть заказов)	Клиент может оформить не более 5 заказов.
Сотрудник	Заказ	Сотрудник ответственен за заказ/ у заказа есть ответственный за него сотрудник	1 : M	Необязательный (в конкретный момент времени у сотрудника может не быть заказов)	У каждого заказа обязательно должен быть сотрудник
Категория лекарства	Лекарство	У лекарства есть категория, под которую оно попадает/Под категорию могут попадать разные лекарства	1 : M	Обязательный (у лекарства должна быть категория)	
Позиция заказа	Заказ	В позиции заказа имеется заказ, к которому относится/ Заказ состоит из позиций	M : 1	Обязательный (заказ состоит как минимум из одной позиции)	Позиций в заказе может быть максимум 30.
Лекарство	Склад	Лекарство хранится на складе/ На складе хранится лекарство	1 : M	Обязательный (лекарство обязательно должно храниться на складе)	Если конкретного лекарства нет на складе, то об этом можно узнать из атрибута количества.
Позиция заказа	Лекарство	Позиция заказа описывает одно лекарство/ лекарство в заказе определяется позицией	1 : M	Обязательный (позиция заказа всегда описывает конкретное лекарство)	
Пользователь	Клиент	У пользователя может быть множество клиентских профилей/ клиентский профиль может принадлежать одному пользователю	1 : M	Необязательный (пользователь может не создавать профиль, а ввести данные при оформлении заказа вручную)	У пользователя может иметься максимум 30 клиентских профилей
Позиция заказа	Аптека	Выдача заказа происходит в одной аптеке/Заказывать лекарство можно в различных пунктах выдачи	1 : M	Обязательный (клиент должен получить заказ в определенной аптеке)	Клиент должен выбрать пункт выдачи

Продолжение таблицы 3

Тип сущности	Тип сущности	Описание	Мощность	Обязательность	Ограничения
Расположение	Позиция расположения	Расположение можно разделить на конечное количество позиций в зависимости от общего кол-ва/Позиция расположения это часть от общего количество расположения	1 : M	Обязательный (позиция расположения это составная часть от расположения)	Сумма количеств позиций должна быть меньше или равна количеству расположения
Заказ	Позиция расположения	Заказ содержит экземпляры лекарств их разных мест хранения/позиция заказа относится к одному заказу	1 : M	Обязательный (позиция расположения это составная часть лекарств в заказе)	Одному заказу соответствует как минимум одна позиция
Заказ	Аптека	В одной аптеки выдается множество заказов/Один конкретный заказ выдается в одной аптеке	M : 1	Обязательный (заказ должен быть выдан в аптеке)	

Так как, при разработке данного приложения использовался подход CodeFirst и использовался объектно-реляционный модуль сопоставления, что позволяет работать с базой данных, используя объекты, то данные определения сущностей были использованы при проектировании классов модели. Диаграмма классов объектов на концептуальном уровне представлена на рисунке А.1.

3 Проектирование информационной системы

При публикации БД на Web-страницы в архитектуру Web-приложений вводятся дополнительные уровни, включающие сервер БД и сервер приложений.

При такой архитектуре Web-сервер передает запрос на генерацию Web-страниц программе-расширению Web-сервера, которая формирует требуемый документ по информации из БД и затем Web-сервер отправляет готовые Web-страницы обратно обозревателю [8].

При разработке данного приложения была использована платформа ASP.NET MVC 5 и трехуровневая архитектура веб-приложения. Это обусловлено тем, что по сравнению с двухуровневой клиент-серверной архитектурой или файл-серверной архитектурой трехуровневая архитектура обеспечивает, как правило, большую масштабируемость, большую конфигурируемость (за счёт изолированности уровней друг от друга) [9].

На рисунке 4 представлена схема трехуровневой архитектуры веб-приложения

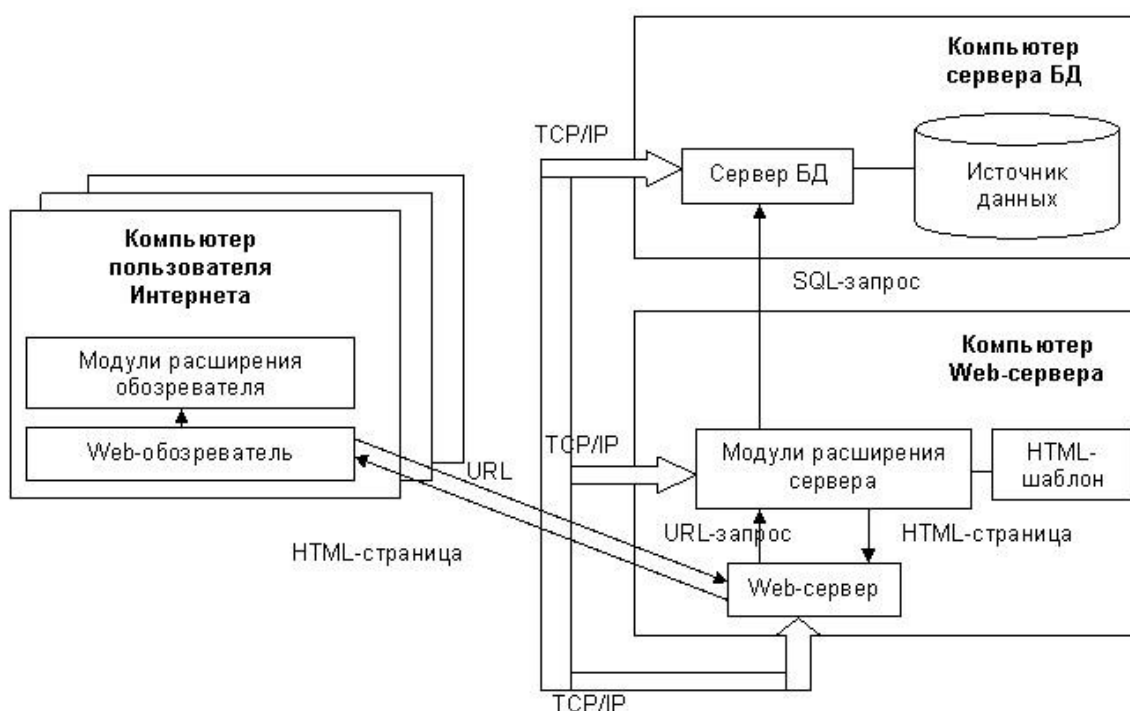


Рисунок 4 – Трехуровневая архитектура веб-приложения

Для подробного отображения общего взаимодействия компонентов системы воспользуемся диаграммой коммуникации (рисунок 5).

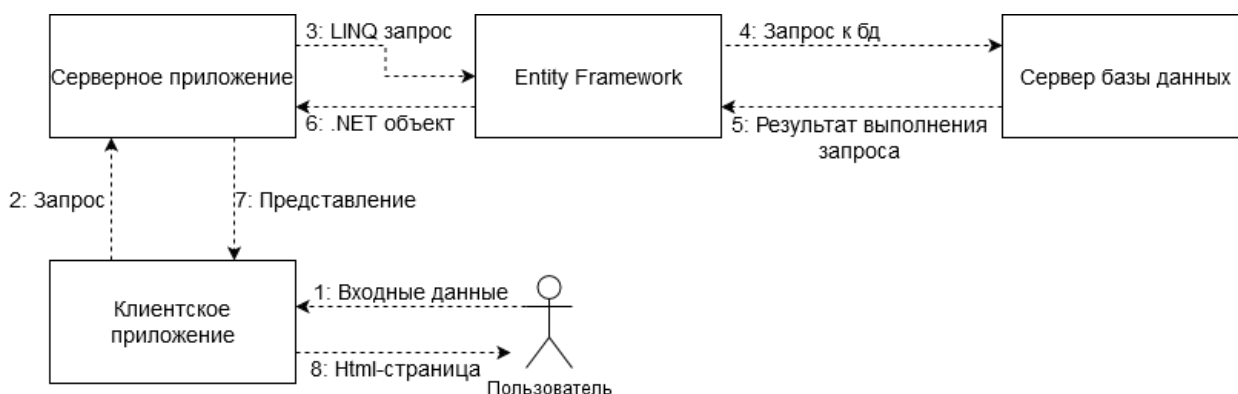


Рисунок 5 – Диаграмма коммуникации

Данная диаграмма отображает взаимодействие системы с пользователем через клиентское приложение (веб-интерфейс). Клиент передаёт серверу входные данные через форму ввода посредством POST/GET запросов. Серверное приложение обрабатывает поступивший запрос, выполняет операции над коллекциями данных посредством синтаксиса языка запросов LINQ. Благодаря Entity Framework запросы к данным транслируются к SQL запросам и выполняются на сервере базы данных. Полученный результат преобразуется в .NET объект, и, исходя из его значения, сервер отправляет соответствующее представление узлу, с которого поступил запрос. Веб-браузер пользователя показывает html-документ.

Взаимодействие пользователя осуществляется в соответствии с естественным циклом: пользователь предпринимает действие, браузер отправляет http-запрос на компьютер сервера, в ответ на которое приложение изменяет свою модель данных, выполняет соответствующий запрос к серверу БД, и доставляет ответ, содержащий обновленное представление пользователю. Передача данных осуществляется с использованием REST подхода. Затем цикл повторяется.

3.1 Выбор технологий разработки

При реализации приложения была выбрана программная платформа ASP .NET MVC 5. Данная платформа имеет ряд встроенных функций, которые необходимы при разработке информационной системы. К ним относятся: аутентификация, авторизация, автоматическая привязка данных, AJAX. Приложения пишутся на строго типизированных языках .NET.

Также к достоинствам данной платформы можно отнести следующие особенности [7]:

- компилируемый код выполняется быстрее, большинство ошибок отлавливается ещё на стадии разработки;
- значительно улучшенная обработка ошибок во время выполнения запущенной готовой программы, с использованием блоков try..catch;
- пользовательские элементы управления (controls) позволяют выделять часто используемые шаблоны, такие как меню сайта;
- опирается на многоязыковые возможности .NET, что позволяет писать код страниц на VB.NET, Delphi.NET, Visual C#, J# и т. д.;
- возможность кэширования всей страницы или её части для увеличения производительности;
- возможность кэширования данных, используемых на странице;
- возможность разделения визуальной части и бизнес-логики по разным файлам («code behind»);
- расширяемая модель обработки запросов;
- расширяемая модель серверных элементов управления;
- наличие master-страниц для задания шаблонов оформления страниц;
- встроенная поддержка AJAX.

В ASP.NET поддерживается несколько программных моделей для создания веб-приложений. Архитектура MVC позволяет нам разделить код

приложения на 3 части: модель (Model), вид или представление (View) и контроллер (Controller).

Модель содержит в себе всю логику приложения, она хранит и обрабатывает данные, при этом, не взаимодействуя с пользователем напрямую. В разрабатываемой информационной системе уровень модели используется для работы с источником данных.

Представление отображает данные, которые ему передаются из контроллера. В разрабатываемом приложении оно состоит из HTML-шаблонов страниц, каскадных таблиц стилей и сценариев на языке JavaScript.

Контроллер отвечает за выполнение запросов, пришедших от пользователя. Он разбирает параметры HTTP-запроса из POST/GET, обращается к модели, чтобы получить или изменить какие-то данные, и в конце вызывает представление, чтобы отобразить результат выполнения запроса.

В серверных приложениях используется схема с пассивной моделью. Когда пользователь заходит на страницу, его браузер отправляет HTTP запрос на получение страницы со списком сообщений. Схема модели представлена на рисунке 6 [7].

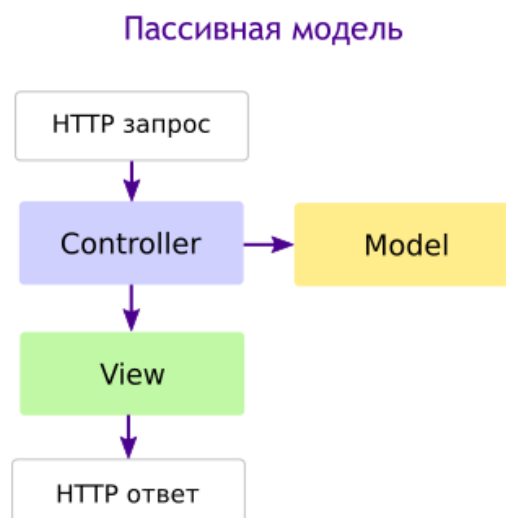


Рисунок 6 – Пассивная MVC модель

Пассивная модель не имеет никаких способов воздействовать на представление или контроллер, и используется ими в качестве источника данных для отображения. Все изменения модели отслеживаются контроллером, и он же отвечает за выбор результирующего представления, если это необходимо.

3.2 Проектирование структур данных

Одним из наиболее популярных программных продуктов в области автоматизированного проектирования баз данных является ERwin фирмы Platinum. Процесс моделирования в ERwin базируется на методологии проектирования реляционных баз данных – IDEF1X.

В ERwin используются два уровня представления модели данных: логический и физический. На логическом уровне не рассматривается использование конкретной СУБД, не определяются типы данных и не определяются индексы для таблиц. Целевая СУБД, имена объектов и типы данных, индексы составляют второй (физический) уровень модели ERwin.

Процесс построения информационной модели включает в себя следующие этапы.

- создание логической модели данных;
- переход к физическому описанию модели;
- генерация физического представления базы данных.

Для того, чтобы у пользователя имелась возможность быстрого и эффективного доступа к данным, требуется спроектировать таблицы связанных между собой сущностей. Все сущности информационной системы были выделены в отдельные модели на основании диаграммы классов. Построенная логическая модель базы данных является развитием сущностей, представленных на диаграмме классов и описанных в разделе «Исследование предметной области». На рисунке А.2 представлена логическая модель базы данных, построенная в ERwin Data Modeler.

При реализации физической модели предметной области будет использована данная логическая модель.

3.3 Проектирование алгоритмов

Одним из основных требований к приложению является наличие сервера. Клиент не может напрямую взаимодействовать с базой данных, а сама база данных может быть развернута как на самом сервере-приложения, так и на отдельно выделенном сервере (в зависимости от архитектуры). Обобщенный алгоритм взаимодействия клиента и сервера можно представить в следующем виде:

- клиент инициирует отправку данных, совершая запрос на определенное действие;
- на стороне сервера инициализируется контроллер, ответственный за взаимодействие с запрошенной страницей;
- сервер приложения посредством механизма LINQ отправляет запрос к серверу базы данных;
- сервер базы данных выполняет запрос и возвращает результат веб-серверу;
- веб-сервер формирует результат в виде HTML-страницы;
- веб-сервер возвращает сформированный гипертекст клиенту.

Исходя из описанных действий, можно сделать вывод, что все операции, связанные с работой с базой данных и генерированием результирующей веб-страницы, выполняются на стороне сервера приложения и сервера базы данных.

Для детализации особенностей алгоритмической и логической реализации выполняемых операций, опишем поведение приложения с помощью диаграммы деятельности на примере выполнения прецедента «Оформить заказ» (рисунок 7).

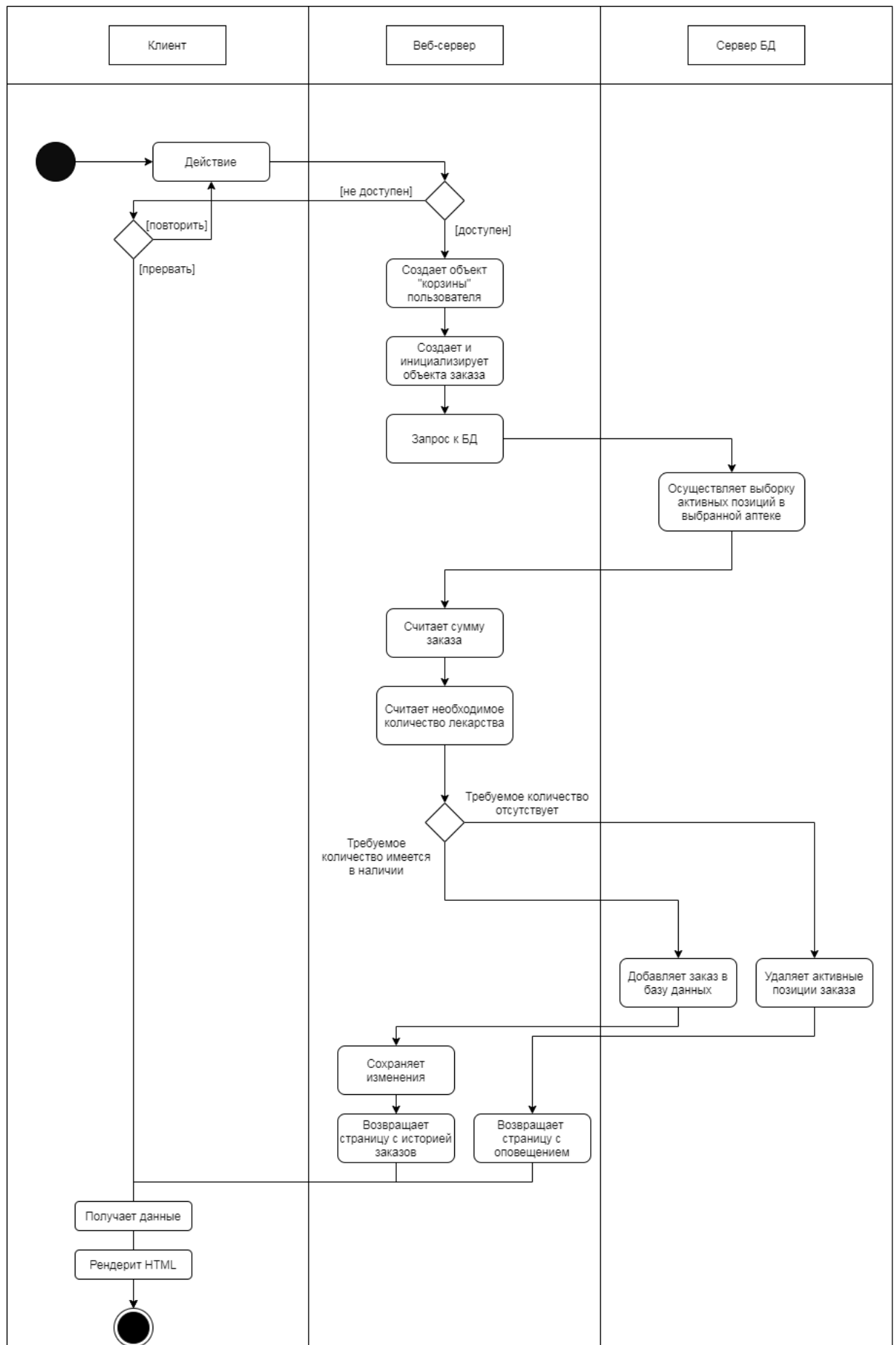


Рисунок 7 – Диаграмма деятельности прецедента «Оформить заказ»

Представленная диаграмма деятельности описывает алгоритм работы сервера веб-приложения. Вариант использования «Оформить заказ» позволяет все «активные» заказы (находящиеся в корзине и в итоговом списке заказов) перевести в состояние «завершенных» (отображаемых в истории пользовательских заказов). Для доступа к электронной корзине пользователя сервер создает и инициализирует соответствующий объект. Веб-сервер отправляет запрос серверу баз данных на выборку активных позиций, тот, в свою очередь, возвращает результат. У текущих активных позиций заказа изменяется статус принадлежности заказу. Сервер выполняет подсчет суммы заказа, проверяет наличие необходимого лекарства на складе, и, если требуемое количество медикамента имеется, то добавляет информацию о заказе в базу данных и возвращает пользователю страницу с историей заказов. Иначе сервер удаляет выбранные активные позиции и перенаправляет пользователя на страницу с уведомлением об отсутствии медикамента на складе. Клиент получает и отображает страницу в браузере пользователя.

Для моделирования взаимодействия объектов системы во времени, а также обмена сообщениями между ними составим диаграмму последовательностей для варианта использования «Добавить позицию хранения». Авторизованный сотрудник выбирает необходимое лекарство и запрашивает форму управления хранилищем. Контроллер обращается к EntityFramework для трансляции LINQ-запроса к SQL-запросу. SQL-запрос выполняется на сервере базы данных, который возвращает результат в виде списка позиций хранения лекарства. Список выбранных лекарств группируется по существующим аптекам, и полученная коллекция данных передается в представление, которое преобразуется к html-странице и отображается в браузере пользователя. Для добавления новой позиции хранения, сотрудник заполняет данные о расположении лекарства и нажимает кнопку «Добавить позицию». Контроллер получает данные о позиции, уведомляет EntityFramework о добавлении новой сущности и сохраняет результат в базе данных. В результате в базу данных добавляется новая

позиция хранения лекарства. Построенная диаграмма последовательностей представлена на рисунке 8.

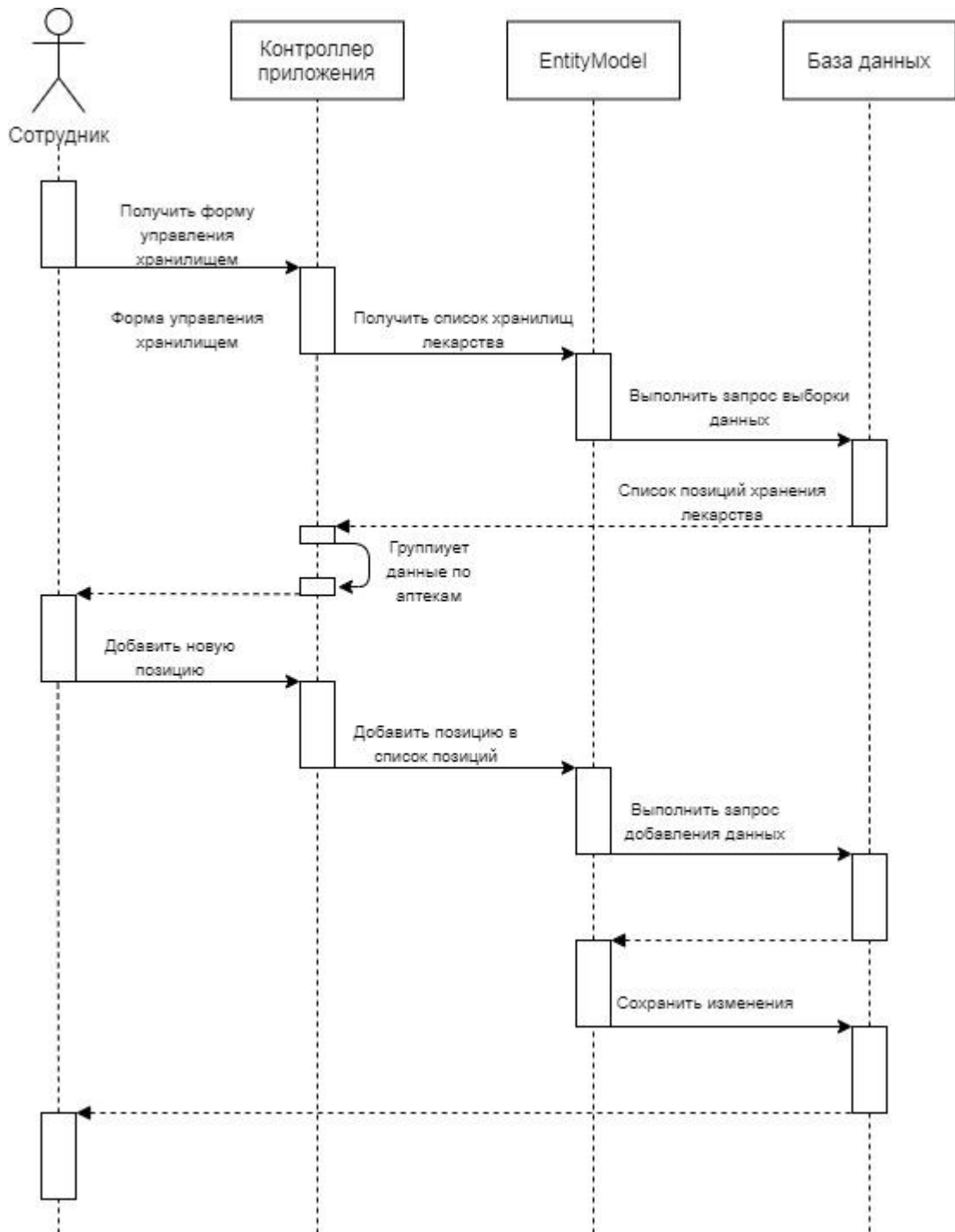


Рисунок 8 – Диаграмма последовательностей для прецедента «Добавить позицию хранения»

Данная диаграмма последовательности позволяет показать время жизни объектов, а также порядок обмена сообщениями между модулями системы.

3.4 Проектирование интерфейса

При выбранной архитектуре Web-сервер передает запрос на генерацию Web-страниц программе-расширению Web-сервера, которая формирует требуемый документ исходя из информации, полученной из БД, и затем Web-сервер отсылает готовые Web-страницы обратно обозревателю, которая выступает в качестве интерфейса [9].

Необходимо спроектировать удобный, адаптивный интерфейс, позволяющий решать все поставленные задачи. Для администрирования и управления веб-приложением необходимо разработать навигационное меню и панель администратора.

Таким образом, интерфейс системы будет состоять из 8 Web-страниц:

- главная html-страница – каталог всех медицинских препаратов, навигация по страницам и фильтрация;
- страница управления заказами пользователя;
- страница описания медикамента, указание мест хранения и количества на складе в аптеках;
- страница управления корзиной пользователя;
- страницы формирования и регистрации заказа;
- страница пользовательского профиля;
- страница панели управления заказами, медикаментами, аптеками;
- страница авторизации пользователя;
- страница регистрации пользователя.

4 Программная реализация информационной системы

4.1 Физическое моделирование и реализация базы данных

Для взаимодействия приложения с базой данных используется специализированная ORM – Entity Framework. Entity Framework является объектно-реляционным модулем сопоставления, который позволяет разработчикам .NET работать с базой данных, используя объекты .NET.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. Entity Data Model – модель, которая сопоставляет классы сущностей с реальными таблицами в БД.

При построении физической модели была использована компьютерная программа для проектирования и документирования баз данных ERwin Data Modeler. При отображении сущностей логической модели в их физические эквиваленты была получена физическая модель, представленная на рисунке А.3.

Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга). Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

Использование подхода CodeFirst сначала было реализовано приложении, содержащее модель данных. Модель была описана с помощью нескольких классов и кода взаимодействия между этими классами.

В качестве представлений в приложении используются шаблоны html-страниц, которые отправляются пользователю по запросу. К странице прилагаются таблицы стилей css и скрипты, написанные на языке JavaScript.

4.2 Разработка серверной части приложения

4.2.1 Разработка классов приложения уровня модели

Все сущности в приложении были выделены в отдельные модели на основании сущностей концептуальной модели.

Таким образом, через классы, определенные в приложении, взаимодействовать с таблицами из базы данных. Диаграммы классов приложения уровня модели представлены в приложении Б.

Класс «MedecineContext» является контекстом данных. Контекст данных представляет собой класс, производный от класса DbContext. Контекст данных содержит одно или несколько свойств типа DbSet<T>, где T представляет тип объекта, хранящегося в базе данных. Данный класс выполняет управление данными о категориях, лекарствах, и сотрудниках, а также пользователях и их ролях.

Классы «Medecine», «Category», «Client», «Employee», «Stock», «OrderPosition», «Order», «StockPosition», «Pharmacy» описывают объекты предметной области, и содержат свойства, которые были выделены на этапе анализа предметной области.

Для авторизации и регистрации пользователей был использован ASP.NET Identity – это API-интерфейс от Microsoft для управления пользователями в приложениях ASP.NET.

Для того, чтобы воспользоваться предоставленным типами данных, необходимо реализовать класс наследник, в котором, по необходимости, можно расширить функционал. В разрабатываемом приложении, классами, ответственными за работу с пользователями и ролями, будут являться, соответственно, «AppUser» и «AppRole». Диаграмма классов, предназначенных для авторизации, представлена на рисунке 9.

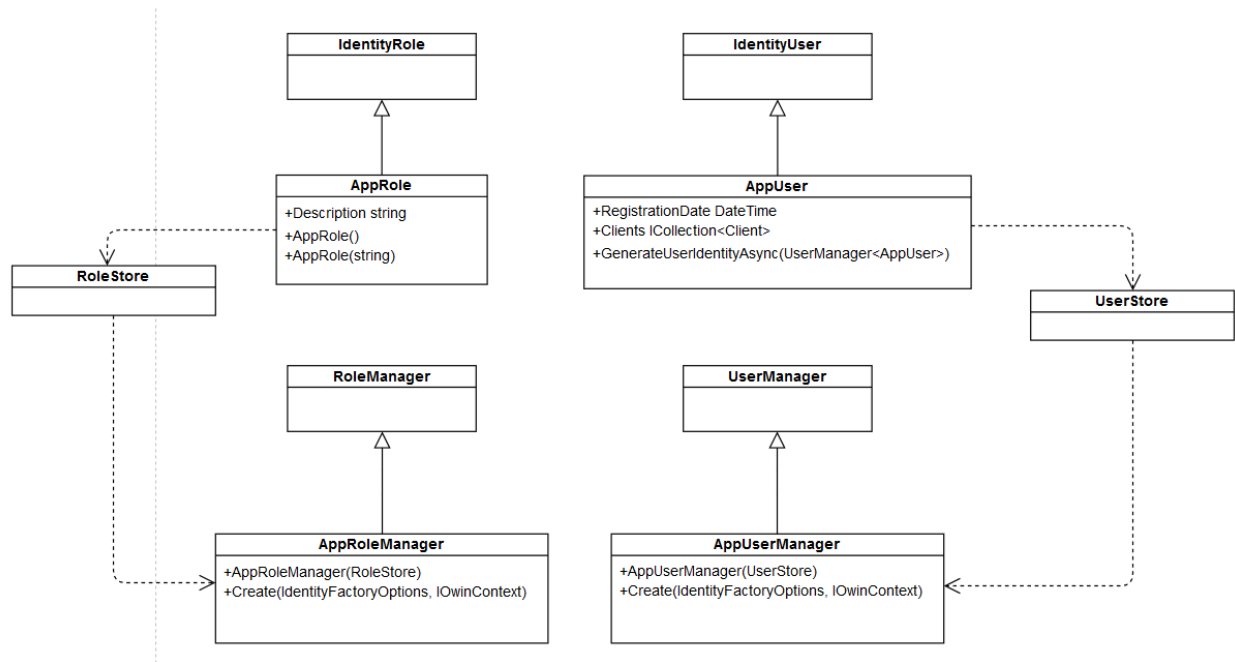


Рисунок 9 – Диаграмма классов, предназначенных для авторизации

Таким образом, благодаря классам модели можно управлять состоянием базы данных на физическом уровне.

4.2.2 Разработка классов приложения уровня контроллеров

Реализация серверной части состояла из разработки контроллеров, в которые приходят запросы от пользователя и реализации классов модели.

Контроллер является центральным компонентом в архитектуре MVC. Контроллер получает ввод пользователя, обрабатывает его и посылает обратно результат обработки в виде представления.

При использовании контроллеров существуют некоторые условности. Так, по соглашениям об именовании названия контроллеров должны оканчиваться на суффикс «Controller», остальная же часть до этого суффикса считается именем контроллера.

В приложении были реализованы следующие классы контроллеров:

- **HomeController** – основной контроллер, который отвечает на навигацию по сайту, а также выдачу информации о каталоге лекарств, истории заказов;

- `OrderConstructorController` – отвечает за оформление и регистрацию заказа;
- `ProfileController` – отвечает за управление профилем пользователя;
- `AdminController` – отвечает за панель управления администратора (авторизованного сотрудника), поиск и модификацию данных, управление хранилищем;
- `AuthorizationController` – отвечает за авторизацию пользователя;
- `MedecineController` – отвечает за представление информации о хранимых лекарствах.

Все вышеописанные контроллеры реализуют интерфейс «`IController`», который определяет один единственный метод «`Execute`», отвечающий за обработку контекста запроса.

Диаграммы классов контроллеров представлены на рисунках В.1, В.2.

Контроллер представляет обычный класс, который наследуется от базового класса «`BaseController`», который, в свою очередь, наследуется от «`System.Web.Mvc.Controller`». Исходный код большинства классов контроллеров представлен в приложении Г.

4.2.3 Разработка приложения уровня представлений

Для визуализации некоторой части модели в виде пользовательского интерфейса в архитектуре используются представления. В данном приложении представления являются как целыми веб-страницами, так и их частью.

В качестве представлений в приложении используются шаблоны `html`-страниц, к которым прилагаются таблицы стилей `css` и сценарии, написанные на языке `JavaScript`.

Страница каталога лекарств и страница с информацией об определенном препарате представлены на рисунках 10 и 11.

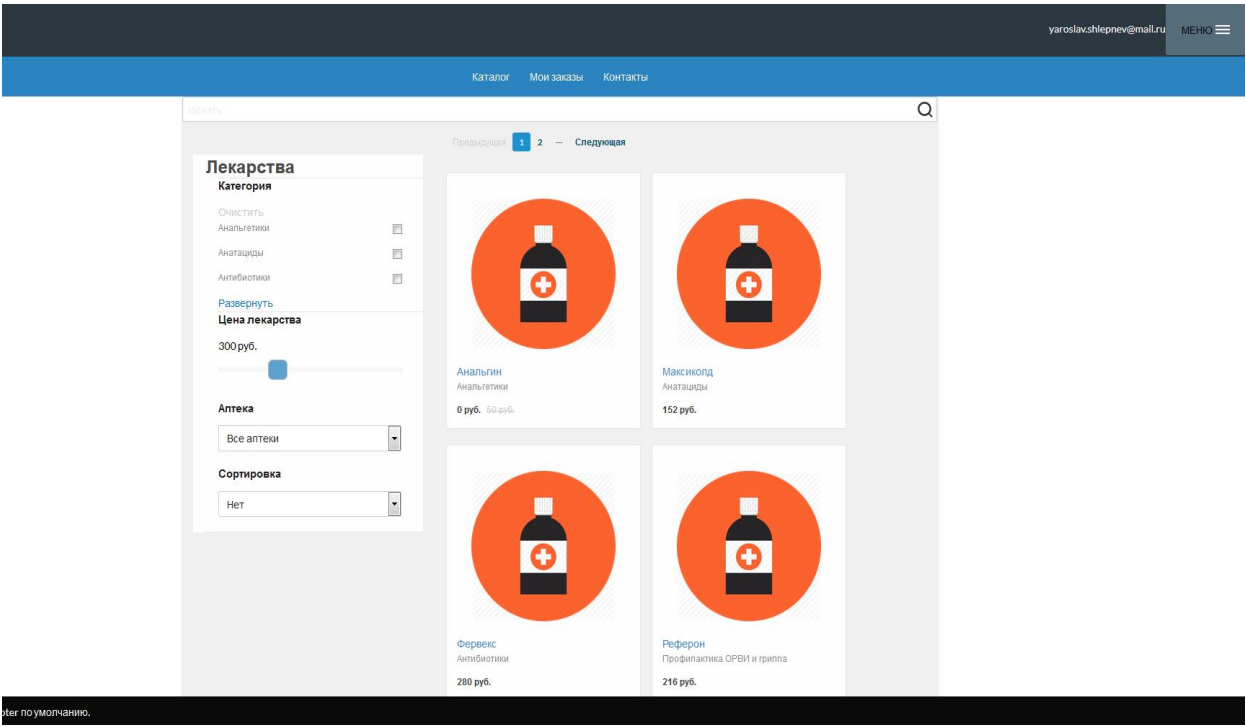


Рисунок 10 – Каталог лекарств

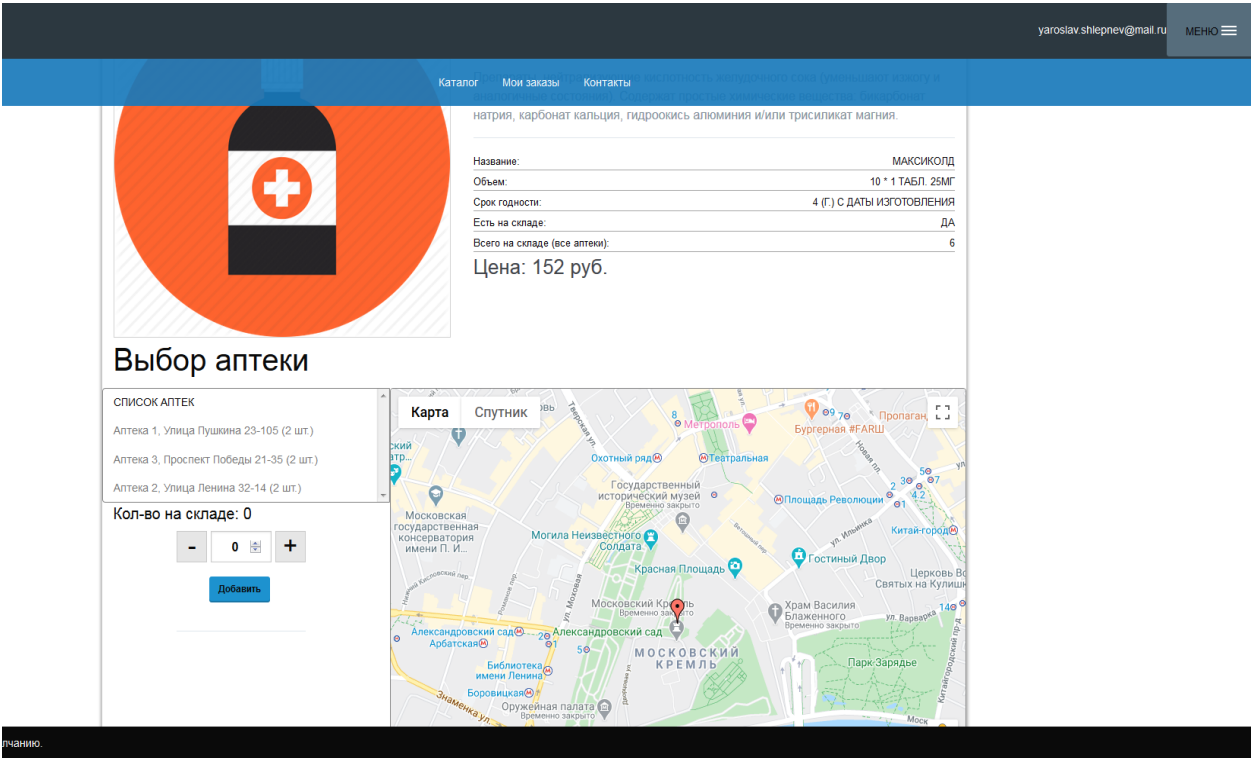


Рисунок 11 – Страница препарата

Каждая разработанная веб-страница содержит навигационную панель, позволяющую перемещаться по portalу. Страница оформления заказа представлена на рисунке 12.

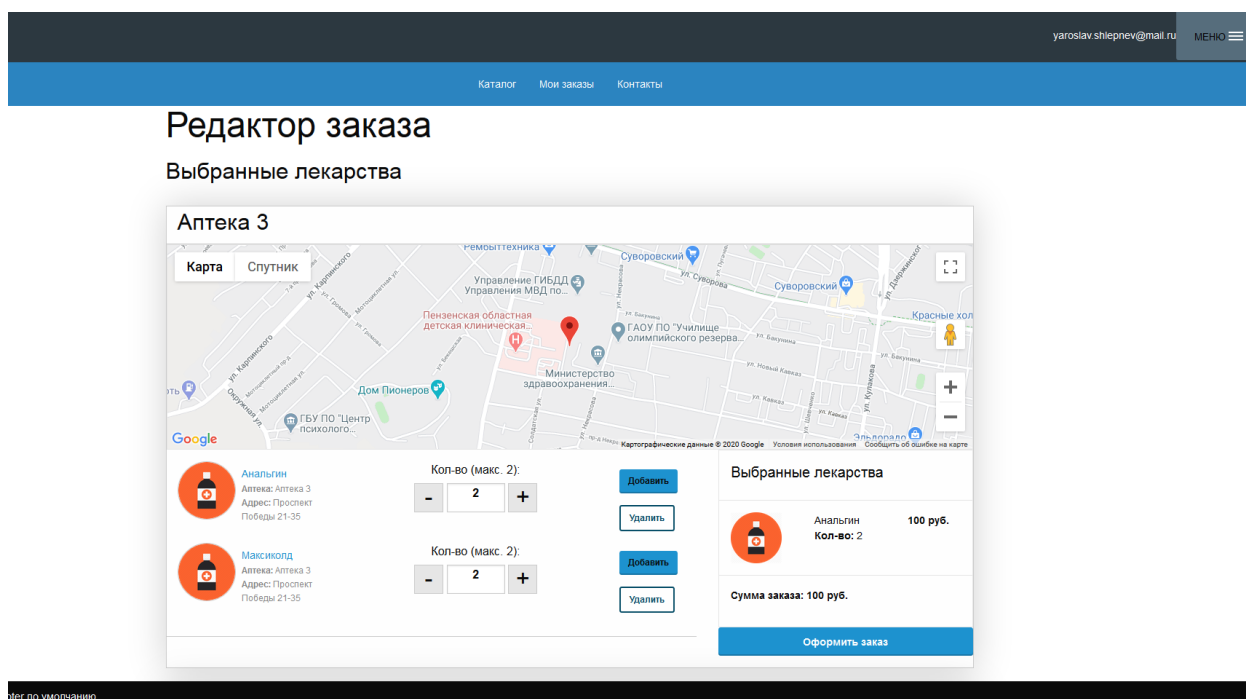


Рисунок 12 – Страница оформления заказа

Каждое из представлений активно взаимодействует с соответствующими контроллерами. Один контроллер может работать с несколькими моделями, и наоборот, одна модель может использоваться в нескольких контроллерах.

4.3 Разработка программного обеспечения клиентского приложения

Для отображения получаемых представлений используется веб-браузер. Представлением веб приложения является веб-страницей или совокупностью веб-страниц, которые предоставляют возможности для взаимодействия с сервисом или устройством посредством протокола HTTP и веб-браузера.

При разработке веб-страницы использовался специальный синтаксис Razor. Стандартное представление очень похоже на обычную веб-страницу с кучей кода html. Однако оно также имеет вставки кода на C#, которые предваряются знаком «собака». Этот знак используется движком представлений Razor для перехода к коду на языке C#.

Описание частичного представления для отображения текущих выбранных активных (входящих в результирующую заявку) позиций заказа представлено на листинге 1.

Листинг 1 – Представление для отображения активных позиций заказа

```
<div class="checkout-summary-title">
    <h5>Выбранные лекарства</h5>
</div>
<div class="scrollable">
@foreach (var ActivePosition in ViewBag.ActivePositions)
{
    <div class="checkout-summary-item">

            <div class="item-name">
                <a>@ActivePosition.Medecine.Name</a>
                <p><span class="title">Кол-во:</span>
@ActivePosition.PositionCount</p>
            </div>
            <div class="item-price">
                <p class="title">@ActivePosition.PositionPrice
руб.</p>
            </div>
        </div>
    }
</div>
<div class="checkout-summary-details">
<div class="left">
    <p><strong>Сумма заказа: @ViewBag.Sum руб.</strong></p>
</div>
</div>
```

Из кода видно, что в представление были переданы данные о текущих выбранных активных позициях и происходит итеративная обработки коллекции позиций. На каждой итерации в код страницы вставляются html-элементы, в результате чего образуется блок данных о каждой позиции заказа.

При вызове метода View контроллер не производит рендеринг представления и не генерирует разметку html. Контроллер только готовит данные и выбирает, какое представление надо вернуть в качестве объекта ViewResult. Затем уже объект ViewResult обращается к движку представления для рендеринга представления в выходной результат.

5 Тестирование программного продукта

5.1 Функциональное тестирование

Одним из способов проверки исправной и корректной работы программы является тестирование с управлением по данным или по принципу «черного ящика». В этом случае программа рассматривается как «черный ящик», и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Для обнаружения всех ошибок в программе, используя управление по данным, необходимо выполнить исчерпывающее тестирование, т.е. тестирование на всех возможных наборах данных. Очевидно, что проведение исчерпывающего тестирования для подавляющего большинства случаев невозможно. Поэтому обычно выполняют «разумное» или «приемлемое» тестирование, которое ограничивается прогонами программы на небольшом подмножестве всех возможных входных данных.

Правильно выбранный тест должен уменьшать, причем более чем на единицу, число других тестов, которые должны быть разработаны для обеспечения требуемого качества программного обеспечения.

При функциональном тестировании различают следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

Для тестирования корректной работы информационной системы были составлены тесты по методу эквивалентных разбиений и методу причинно-следственных связей для функции «Оформить заказ», а частности, для валидации корректного ввода даты выдачи заказа.

5.1.1 Метод эквивалентного разбиения

Идея тестирования по методу разбиения классов эквивалентности состоит в том, чтобы исключить набор входных данных, которые заставляют систему вести себя одинаково и давать одинаковый результат при тестировании программы. Данный метод был применен для проверки работы функции «Оформить заказ», в частности, для валидации корректного ввода даты выдачи заказа. Выделенные классы эквивалентности представлены в таблицах 4, 5.

Таблица 4 – Классы эквивалентов для поля «Дата выдачи»

Входные условия	Правильные классы эквивалентности	Неправильные классы эквивалентности
Дата между 12.12.2019 и 12.12.2099	12.12.2019 < значение даты < 12.12.2099(1)	значение даты <= 12.12.2019(2) значение даты >= 12.12.2099(3)
Должны использоваться только цифры(1..9)	1..9(4)	а..я, А..Я, а..z, А..Z(5)
Разделители только точка(.) и слеш(/)	.,/(6)	-, (7)
Поле не должно быть пустым	Не пусто	Пусто

Таблица 5 – Правильность классов эквивалентности для поля «Дата выдачи»

№	Входные данные	Правильность класса
1	12.12.2019..12.12.2099	+
2	<= 12.12.2019	-
3	>= 12.12.2099	-
4	1..9	+
5	а..я, А..Я, а..z, А..Z	-
6	.,/	+
7	-,	-

В таблице 6 представлены результаты тестирования методом «эквивалентных разбиений» для ввода данных в поле «Дата выдачи».

Таблица 6 – Результаты тестирования для поля «Дата выдачи»

Класс	Вход. данные	Предполагаемые выходные данные	Результат	Уровень ошибки
1	30.12.2026	Ввод данных	Данные успешно добавлены в базу	-
	05.03.2055	Ввод данных	Данные успешно добавлены в базу	-
	14.10.2098	Ввод данных	Данные успешно добавлены в базу	-
2	14.10.1998	Сообщение о недопустимости даты	Данные не добавлены	-
3	14.10.3998	Сообщение о недопустимости даты	Данные не добавлены	-
4	14.10.2027	Ввод данных	Данные успешно добавлены в базу	-
	20.01.2046	Ввод данных	Данные успешно добавлены в базу	-
	25.06.2077	Ввод данных	Данные успешно добавлены в базу	-
5	пп.zz.AbCd	Сообщение о недопустимости данных	Данные не добавлены	-
6	12.12.2071	Ввод данных	Данные успешно добавлены в базу	-
	12/12/2071	Ввод данных	Данные успешно добавлены в базу	-
7	12-12-2071	Недопустимость ввода	Данные не ввелись	-
	12 12 2071	Недопустимость ввода	Данные не ввелись	-

Таким образом, благодаря применению метода классов эквивалентности, было выявлено, что валидация поля ввода «Дата выдачи» работает корректно.

5.1.2 Метод анализа причинно-следственных связей

Анализ причинно-следственных связей позволяет системно выбирать высоко результативные тесты. Метод использует алгебру логики и оперирует понятиями «причина» и «следствие». Причиной в данном случае называют отдельное входное условие или класс эквивалентности. Следствие – выходное условие или преобразование системы. Идея метода заключается в отнесении всех следствий к причинам, т. е. в уточнении причинно-следственных связей. Данный метод дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций.

Рассмотрим спецификацию добавления медикамента в базу данных.

Причины:

- название корректно (1);
- срок годности корректный (2);
- категория существует (3);
- количество корректно (4);
- цена корректна (5).

Следствия:

- сообщение о некорректности названия (X1);
- сообщение о некорректности срока годности (X2);
- сообщение об отсутствии категории (X3);
- сообщение о некорректности количества (X4);
- сообщение о некорректности цены (X5);
- добавление лекарства в базу данных (X6).

Исходя из выделенных причин и следствий построим диаграмму функциональности (рисунок 13).

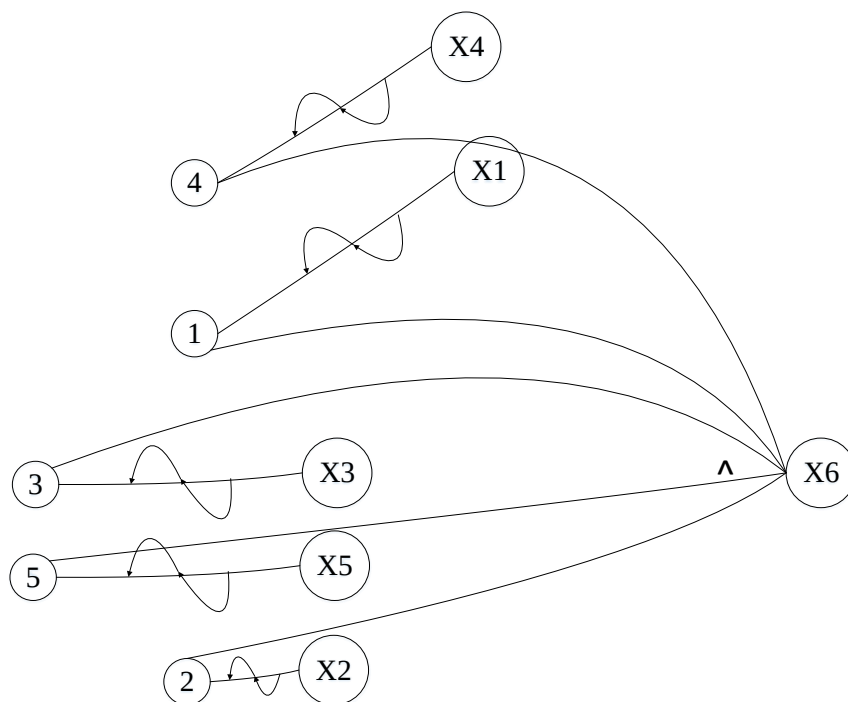


Рисунок 13 – Диаграмма функциональности

Далее на основе анализа семантического содержания спецификации была построена таблица истинности (таблица 7), в которой каждой возможной комбинации причин ставится в соответствие следствие. При этом целесообразно истину обозначать «1», ложь - «0», а для обозначения безразличных состояний условий применять обозначение «X», которое предполагает произвольное значение условия (0 или 1).

Таблица 7 – Таблица истинности диаграммы функциональности

1	2	3	4	5	X1	X2	X3	X4	X5	X7
0	0	0	0	0	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	0	0
0	0	0	1	0	1	1	1	0	1	0
0	0	0	1	1	1	1	1	0	0	0
0	0	1	0	0	1	1	0	1	1	0
0	0	1	0	1	1	1	0	1	0	0
0	0	1	1	0	1	1	0	0	1	0
0	0	1	1	1	1	1	0	0	0	0
0	1	0	0	0	1	0	1	1	1	0
0	1	0	0	1	1	0	1	1	0	0
0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1	0	0	0
0	1	1	0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0	0	1	0
0	1	1	1	1	1	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	0
1	0	0	0	1	0	1	1	1	0	0
1	0	0	1	0	0	1	1	0	1	0
1	0	0	1	1	0	1	1	0	0	0
1	0	1	0	0	0	1	0	1	1	0
1	0	1	0	1	0	1	0	1	0	0
1	0	1	1	0	0	1	0	0	1	0
1	0	1	1	1	0	1	0	0	0	0
1	1	0	0	0	0	0	1	1	1	0
1	1	0	0	1	0	0	1	1	0	0
1	1	0	1	0	0	0	1	0	1	0
1	1	0	1	1	0	0	1	0	0	0
1	1	1	0	0	0	0	0	1	1	0
1	1	1	0	1	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	0	1
1	1	1	1	1	1	0	0	0	0	1

На основе таблицы истинности были получены тест кейсы, каждая строка таблицы была преобразована в тест. Данный метод позволяет строить высоко результативные тесты и обнаруживать неполноту и неоднозначность исходных. Полученные тест кейсы представлены в таблице 8.

Таблица 8 – Полученные тест-кейсы методом причинно-следственных связей

Номер	Название корректно	Срок годности корректный	Категория существует	Количество корректно	Цена коррект.	Предполагаемый результат	Фактический результат
1	Нет	Нет	Нет	Нет	Нет	Медикамент не добавлен	Медикамент не добавлен
2	Нет	Нет	Нет	Нет	Да	Медикамент не добавлен	Медикамент не добавлен
3	Нет	Нет	Нет	Да	Нет	Медикамент не добавлен	Медикамент добавлен
4	Нет	Нет	Нет	Да	Да	Медикамент не добавлен	Медикамент не добавлен
5	Нет	Нет	Да	Нет	Нет	Медикамент не добавлен	Медикамент не добавлен
6	Нет	Нет	Да	Нет	Да	Медикамент не добавлен	Медикамент не добавлен
7	Нет	Нет	Да	Да	Нет	Медикамент не добавлен	Медикамент не добавлен
8	Нет	Нет	Да	Да	Да	Медикамент не добавлен	Медикамент не добавлен
9	Нет	Да	Нет	Нет	Нет	Медикамент не добавлен	Медикамент добавлен
10	Нет	Да	Нет	Нет	Да	Медикамент не добавлен	Медикамент не добавлен
11	Нет	Да	Нет	Да	Нет	Медикамент не добавлен	Медикамент не добавлен
12	Нет	Да	Нет	Да	Да	Медикамент не добавлен	Медикамент не добавлен
13	Нет	Да	Да	Нет	Нет	Медикамент не добавлен	Медикамент добавлен
14	Нет	Да	Да	Нет	Да	Медикамент не добавлен	Медикамент добавлен
15	Нет	Да	Да	Да	Нет	Медикамент не добавлен	Медикамент не добавлен
16	Нет	Да	Да	Да	Да	Медикамент не добавлен	Медикамент не добавлен
17	Да	Нет	Нет	Нет	Нет	Медикамент не добавлен	Медикамент не добавлен
18	Да	Нет	Нет	Нет	Да	Медикамент не добавлен	Медикамент добавлен
19	Да	Нет	Нет	Да	Нет	Медикамент не добавлен	Медикамент не добавлен
20	Да	Нет	Нет	Да	Да	Медикамент не добавлен	Медикамент не добавлен
21	Да	Нет	Да	Нет	Нет	Медикамент не добавлен	Медикамент не добавлен
22	Да	Нет	Да	Нет	Да	Медикамент не добавлен	Медикамент не добавлен
23	Да	Нет	Да	Да	Нет	Медикамент не добавлен	Медикамент не добавлен
24	Да	Нет	Да	Да	Да	Медикамент не добавлен	Медикамент не добавлен
25	Да	Да	Нет	Нет	Нет	Медикамент не добавлен	Медикамент не добавлен
26	Да	Да	Нет	Нет	Да	Медикамент не добавлен	Медикамент добавлен
27	Да	Да	Нет	Да	Нет	Медикамент не добавлен	Медикамент не добавлен
28	Да	Да	Нет	Да	Да	Медикамент не добавлен	Медикамент не добавлен
29	Да	Да	Да	Нет	Нет	Медикамент не добавлен	Медикамент не добавлен

Продолжение таблицы 8

Номер	Название корректно	Срок годности корректный	Категория существует	Количество корректно	Цена корректна	Предполагаемый результат	Фактический результат
30	Да	Да	Да	Нет	Да	Медикамент не добавлен	Медикамент не добавлен
31	Да	Да	Да	Да	Нет	Медикамент не добавлен	Медикамент добавлен
32	Да	Да	Да	Да	Да	Медикамент добавлен	Медикамент добавлен

Данный метод позволяет строить высоко результативные тесты и обнаруживать неполноту и неоднозначность исходных спецификаций. Его недостатком является неадекватное исследование граничных значений.

Таким образом, был проведен ряд тестов по методам эквивалентных разбиений и причинно-следственных связей. Тесты показали исправную работу механизма валидации поля «Дата выдачи» при оформлении заказа. При тестировании формы добавления нового медикамента методом причинно-следственных связей были выявлены ошибки, приводящие к неверному результату выполнения. Полученные результаты тестирования были использованы при исправлении ошибок в коде.

5.2 Структурное тестирование

Тестирование по принципу белого ящика характеризуется тем, как тесты покрывают логику (исходный текст программы). Структурное тестирование включает в себя метод покрытия операторов, метод покрытия условий и метод покрытия маршрутов.

Произведем оценку методов тестирования «Белого ящика» для корректной работы фильтрации каталога медикаментов. Сегмент кода метода фильтрации представлен на листинге 2.

Листинг 2 – Участок метода фильтрации каталога лекарств

```
var catalog = $('<div>.catalog-partial');
while (catalog.firstChild) {
    catalog.removeChild(catalog.firstChild);
}
var priceValue = $('<div>input[type=range]').val();
```

```

if (priceValue < 0)
    priceValue = 0;

$('#priceValue').text(priceValue + " руб.");

var check = false;
if (window.categories.length != 0)
    check = true;

catalog.load("/Home/GetCatalog",
    { filterOn: check, category_ids: window.categories, price:
      priceValue });

```

Представим данный участок кода в виде графа (рисунок 14). На графе изображены две версии алгоритма работы данного метода. Первый граф корректно отображает последовательность действий и условий, а во втором графе специальным образом допущены ошибки в этих условиях. Это необходимо для проверки реакции приложения на ошибочные ситуации в коде.

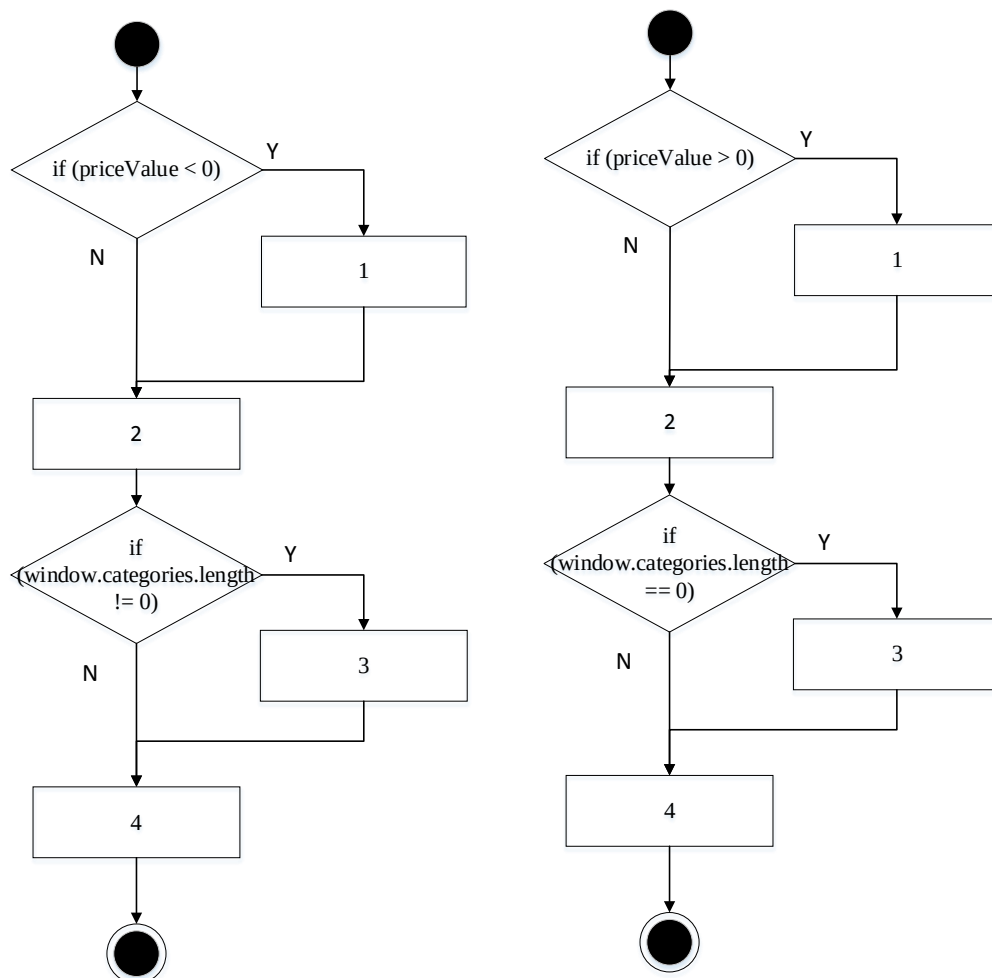


Рисунок 14 – Блок схема участка функции фильтрации каталога

Для данного участка кода был построен управляющий граф потока данных (рисунок 15).

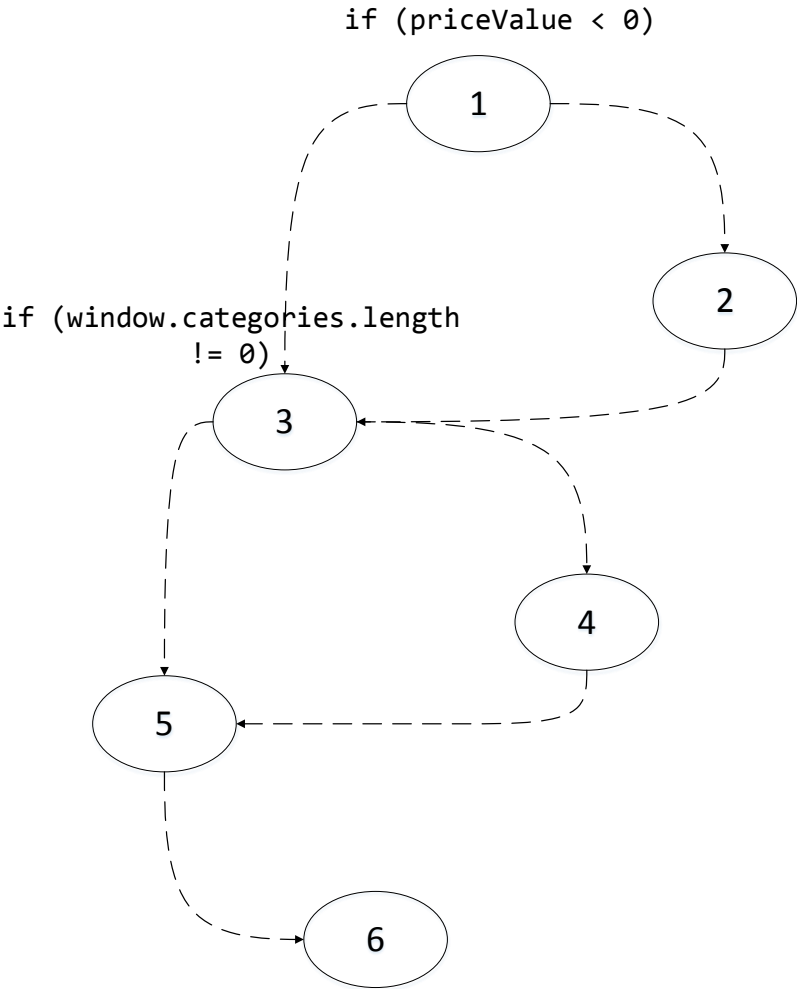


Рисунок 15 – Управляющий граф потока данных

Для сокращения названий локальных переменных, используемых в коде, введем некоторые обозначения, которые указаны в таблице 9.

Таблица 9 – Сокращение названий локальных переменных

Название переменной	Обозначение
priceValue	A
window.categories.length	B

Данные обозначения будут использоваться при применении методов тестирования.

5.2.1 Метод покрытия операторов

Было произведено тестирование методом покрытия операторов с использованием замены условий. Результаты сравнения работы программы приведены в таблице 10.

Таблица 10 – Результаты тестирования методом «Покрытия операторов»

Замена условий		Тест	Ожидаемый результат	Фактический результат	Результат тестирования
Исходное	Замена				
$A < 0$	$A > 0$	$A = 350$ $B = 2$	Применится фильтр лекарств по категориям	Фильтр не применится	Успешно
$B \neq 0$	$B == 0$				

Таким образом, по результатам, приведенным в таблице, можно сделать вывод, что ошибка в коде будет выявлена.

5.2.2 Метод покрытия решений

Было проведено тестирование методом покрытия решений. Для функции, приведенной на рисунке 20 покрытие решений может быть выполнено двумя тестами, покрывающими пути 13 и 24, либо 14 и 23. Результаты тестирования приведены в таблице 11.

Таблица 11 – Результаты тестирования методом «Покрытия решений»

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = -5$ $B = 0$	Фильтр не применится	Фильтр применится	Успешно
$A = 100$ $B = 5$	Фильтр применится	Фильтр не применится	Успешно

Таким образом, для любых входных данных работа программы не приведет в ошибку.

5.2.3 Метод покрытия условий

Критерий покрытия условий является еще более «сильным» по сравнению с предыдущими. В этом случае формируют некоторое количество тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней мере, один раз.

Однако, как и в случае покрытия решений, этот критерий не всегда приводит к выполнению каждого оператора, по крайней мере, один раз. К критерию требуется дополнение, заключающееся в том, что каждой точке входа управление должно быть передано, по крайней мере, один раз. В таблице 12 представлены результаты тестирования методом «покрытия условий».

Таблица 12 – Результаты тестирования методом «Покрытия условий»

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = -100 B = 0	Фильтр не применится	Фильтр применится	Успешно
A = 200 B = 6	Фильтр применится	Фильтр не применится	Успешно

Указанные два теста покрывают все результаты каждого из условий.

Применение структурного тестирования показало, что программа устойчива к ошибкам в программном коде, и, при их появлении, они будут выявлены.

5.3 Метрики кода

Метрики программного обеспечения позволяют получить численное значение некоторого свойства программного обеспечения или его спецификаций. Произведем подсчет метрик для функции поиска пользователя в базе данных, листинг которой приведен в приложении Д.

5.3.1 Метрика ABC

ABC-метрика основана на подсчете присваиваний значений переменным, явных передач управления за пределы области видимости, т.е. вызовов функций, и логических проверок. Мера записывается тройкой значений, например, «ABC = <16, 17, 9>» [13], но для оценки сложности программы вычисляется одно число, как квадратный корень из суммы квадратов А, В, С [13]. Для указанного участка кода данная тройка значений будет равна примерно 25, в связи с этим, можно сделать вывод, что данный программный код имеет приемлемую сложность и не слишком большие размеры.

5.3.2 Метрики Джилба

Метрики Джилба показывают сложность программного обеспечения на основе насыщенности программы условными операторами или операторами цикла [13]. Расчет относительной сложности программы по Джилбу производится по формуле:

$$cl = \frac{CL}{n},$$

где CL – абсолютная сложность (количество управляющих операторов);

n – общее число операторов программы.

Данная метрика хорошо отражает как трудоемкость разработки, так и сложность понимания программы [13]. Результаты вычисления данной метрики кода представлены в таблице 14.

Таблица 14 – Результаты вычисления метрики Джилба

Показатель	Значение
Всего операторов (n)	119
Управляющие операторы (CL)	13
Относительная сложность программы	0.11

Из полученных результатов анализа текста программы следует, что исходный код имеет невысокую сложность, так как на 119 операторов текста приходится всего лишь 13 операторов условия. Общее число программных модулей решения также невелико (1 модуль), что подтверждает низкий уровень сложности программы.

5.3.3 Метрики Холстеда

Метрики Холстеда позволяют частично учесть возможность записи одной и той же функциональности разным количеством строк и операторов [13]. Они основаны на большом числе количественных показателей, таких, как число уникальных операторов программы, число уникальных операндов программы, общее число операторов, общее число операндов, теоретическое число уникальных операторов и теоретическое число уникальных операндов. Через эти показатели различными формулами определяются уровень качества программирования, сложность понимания программы, трудоемкость кодирования программы, уровень языка выражения и информационное содержание программы [13]. Для подсчета требуемых показателей необходимо сформировать словарь операндов (таблица 15).

Таблица 15 – Операнды и операторы

Операторы	Число вхождений	Операнды	Число вхождений
=	16	parameters	11
.	35	userListResult	15
(	19	m_context	1
)	19	user	9
++	2	userListArray	5
if	9	i	4
for	2	j	4
foreach	2	clients	4

Продолжение таблицы 15

Операторы	Число вхождений	Операнды	Число вхождений
!=	5	client	4
!	1	result	6
[	2	ViewBag	1
]	2	UserName	3
<	2	Phone	2
break	2	Clients	3
return	1	Name	1
		CName	2
Операторы	Число вхождений	Операнды	Число вхождений
		LastName	3
		PhoneNumber	2

Результаты вычисления данной метрики кода представлены в таблице 16.

Таблица 16 – Результаты вычисления метрик Холстеда

Показатель	Значение
Общее число операторов программы (N1)	119
Общее число операндов (N2)	80
Число уникальных операторов (n1)	15
Число уникальных операндов (n2)	18
Словарь программы ($n = n1 + n2$)	33
Длина программы ($N = N1 + N2$)	199
Объем программы ($V = N \times \log_2(n) = 199 \times 5$)	995

Размер метрики объема программы должен быть от 20 до 1000 для одной функции, следовательно, вычисленный объем программы удовлетворяет данному критерию, и, можно сказать, что качество программирования находится на приемлемом уровне, а трудоемкость кода не высока.

5.3.4 Цикломатическая метрика Мак-Кейба

Цикломатическая сложность, или метрика Мак-Кейба является наиболее известной и широко используемой при создании инструментария для оценки сложности программ на различных языках [13]. Цикломатическая сложность вычисляется для графа потока управления процедуры или функции по формуле:

$$V(G) = e - n + 2p,$$

где e - количество дуг;

n - количество вершин;

p - число компонент связности графа потока управления.

Число компонент связности рассматривается как количество дуг, которые надо добавить для преобразования графа в сильно связный, т.е. любые две вершины которого взаимно достижимы [13]. Построенный граф связности представлен в приложении Е.

Результаты вычисления метрики Мак-Кейба кода представлены в таблице 17.

Таблица 17 – Результаты вычисления метрик Мак-Кейба

Показатель	Значение
Количество ребер в графе (e)	24
Количество узлов в графе (n)	17
Количество компонент связности (p)	1
Цикломатическая сложность ($V(G)$)	9

Из полученного значения метрики можно сделать вывод, что рассматриваемый код является достаточно понятным и доступным.

Таким образом, метрики кода позволяют изучить сложность разрабатываемой информационной системы, оценить объем работ, стилистику разрабатываемого приложения и усилия, потраченные каждым разработчиком для реализации данного решения.

6 Исследование затрат на разработку информационной системы

В связи с этим большое значение имеет величина затрат на создание качественного продукта и ее сравнение с аналогичными затратами у конкурентов. В первую очередь, это возросший интерес к затратам на качество, как одной из основных составляющих конкурентоспособности.

В этой связи производители ставят целью минимизации затрат, связанных с созданием качественного продукта. Поэтому все большую важность приобретает управление затратами на разработку [10].

6.1 Структурное планирование

Построение сетевой модели на этапе структурного планирования позволяет детально проанализировать все операции и внести улучшения в структуру программы еще до начала ее реализации. Однако еще более существенную роль играет использование сетевой модели для разработки календарного плана выполнения программы [10].

Структурное планирование включает в себя несколько этапов [11]:

- разбиение проекта на совокупность отдельных работ, выполнение которых необходимо для реализации проекта;
- построение сетевого графика, описывающего последовательность выполнения работ;
- оценка временных характеристик работ и анализ сетевого графика.

Рассмотрим перечень задач, представляющих из себя этапы создания информационной системы заказа медицинских препаратов (таблица 18).

Таблица 18 – Перечень задач

№	Название задачи	Длительность	Предшественники
1	Начало реализации проекта	-	-
2	Выбор программных средств и архитектуры	-	-
3	Программирование	-	-
4	Разработка базы данных	-	-
5	Постановка задачи	1	1
6	Проектирование базы данных	3	5

Продолжение таблицы 18

№	Название задачи	Длительность	Предшественники
7	Разработка логической модели	2	6
8	Разработка физической модели	2	6
9	Разработка БД завершена	-	7,8
10	Разработка клиентской части	-	-
11	Постановка задачи	1	1
12	Проектирование интерфейса	5	11
13	Разработка шаблонов html-страниц	6	12
14	Стилизация и добавление динамики в интерфейс	6	13
15	Разработка клиентской части завершена	-	14
16	Разработка серверной части	-	-
17	Постановка задачи	1	9, 15
18	Разработка уровня модели	5	17
19	Разработка уровня контроллеров	7	17
20	Разработка серверной части завершена		18, 19
21	Программирование завершено	-	20, 15, 9
22	Отладка	-	-
23	Тестирование работы приложения	4	21
24	Исправление ошибок	3	23
25	Отладка завершена	-	24
26	Конец проекта	-	25

Из таблицы видно, что для реализации проекта необходимо выполнить 26 задач. Рассмотрим список затрачиваемых ресурсов (таблица 18).

Таблица 19 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист1	Трудовой	75 Р\ч	125 Р\ч
Программист2	Трудовой	65 Р\ч	125 Р\ч
Компьютер	Материальный	1000 Р	
Сервер	Материальный	350 Р	

В таблице 19 отображен список ресурсов в задачах, необходимых для выполнения проекта.

Таблица 20 – Назначения ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало реализации проекта		
2	Выбор программных средств и архитектуры		

Продолжение таблицы 20

№	Название задачи	Ресурс	Единицы
3	Программирование		
4	Разработка базы данных		
5	Постановка задачи	Программист1	100%
6	Проектирование базы данных	Программист1	50%
7	Разработка логической модели	Программист1 Компьютер	100% 1 шт.
8	Разработка физической модели	Программист1 Компьютер	100% 1 шт.
9	Разработка БД завершена		
10	Разработка клиентской части		
11	Постановка задачи	Программист2	100%
12	Проектирование интерфейса	Программист2	50%
13	Разработка шаблонов html-страниц	Программист2 Компьютер	100% 1 шт.
14	Стилизация и добавление динамики в интерфейс	Программист2 Компьютер	100% 1 шт.
15	Разработка клиентской части завершена		
16	Разработка серверной части		
17	Постановка задачи	Программист1 Программист2	100% 100%
18	Разработка уровня модели	Программист1 Программист2	100% 50%
19	Разработка уровня контроллеров	Программист1 Программист2 Компьютер Сервер	50% 100% 2 шт. 1 шт.
20	Разработка серверной части завершена		
21	Программирование завершено		
22	Отладка		
23	Тестирование работы приложения	Программист1 Компьютер Сервер	100% 1 шт. 1 шт.
24	Исправление ошибок	Программист1 Компьютер Сервер	100% 1 шт. 1 шт.
25	Отладка завершена		
26	Конец проекта		

Из представленных таблиц видно, что решением задач будут заниматься два программиста, которым для этого необходимы рабочие компьютеры, а также выделенный или специализированный компьютер для выполнения сервисного программного обеспечения (сервер).

Требуется рассчитать время, которое необходимо для реализации проекта и вычислим его критический путь. Для этого требуется построить сетевой график проекта. Сетевой график – это ориентированный граф, в котором вершинами обозначены работы проекта, а дугами – временные взаимосвязи работ [11]. Сетевой график отображает взаимосвязи между операциями и порядок их выполнения [5]. Построенный сетевой график представлен на рисунке Ж.1.

Для вычисления критического пути необходимо для каждой задачи определить раннее и позднее время начала работ. Раннее время определяется по формуле:

$$T_{p(i)} = \max_{j \in G} (T_{p(j)} + t_j),$$

где G – множество работ, непосредственно предшествующих работе i ;

$T_{p(j)}$ – раннее время выполнения работы j ;

t_j – время выполнения работы j .

Позднее время начала следует вычислять по формуле:

$$T_{п(i)} = \min_{j \in H} (T_{п(j)} - t_j),$$

где H – множество работ, непосредственно следующих за работой i ;

$T_{п(j)}$ – позднее время выполнения работы j ;

t_j – время выполнения работы j .

Вычисление раннего времени начала работ приведено в таблице 20, вычисление позднего – в таблице 21.

Таблица 21 – Вычисление раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
1	-	-	-	-	0
2	-	-	-	-	0
3	-	-	-	-	0
4	-	-	-	-	0
5	1	0	0	0	0
6	5	0	1	1+0	1

Продолжение таблицы 21

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
7	6	1	3	3+1	4
8	6	1	3	3+1	4
9	7	2	4	Max(4+2;4+2)	6
	8	2	4		
10	-	-	-	-	0
11	1	0	0	0	0
12	11	0	1	1	1
13	12	1	5	5+1	6
14	13	6	6	6+6	12
15	14	12	6	6+12	18
17	9	6	0	Max(0+6;0+18)	18
	15	18	0		
18	17	18	1	1+18	19
19	17	18	1	1+18	19
20	18	19	5	Max(5+19;7+19)	26
	19	19	7		
21	20	26	0	26	26
22	21	26	0	26	26
23	22	26	0	26	26
24	23	26	4	4+26	30
25	24	30	3	3+30	33
26	25	33	0	31	33

Из результатов таблицы видно, что выполнение проекта займет тридцать три рабочих дня.

Таблица 22 – Вычисление позднего времени начала работ

Работа	Последователи		Длительность	Вычисление	Позднее время начала
	№	Позднее время начала			
26	-	-	-	-	33
25	26	33	0	33 – 0	33
24	25	33	3	33 – 3	30
23	24	33	4	30 – 4	26
22	23	30	0	26 – 0	26
21	22	26	0	26	26
20	21	26	0	26	26
19	20	26	7	26 – 7	19
18	20	26	5	26 – 5	21
17	18	21	1	Min(21-1; 19-1)	18
	19	19			
16	17	18	0	18	18
15	16	18	0	18	18
14	15	18	6	18 – 4	12
13	14	12	6	Min(12-6;18-6)	6
	15	18			
12	13	6	5	6-5	1
11	12	1	1	1-1	0
9	17	18	0	19-1	18

Продолжение таблицы 22

Работа	Последователи		Длительность	Вычисление	Позднее время начала
	№	Позднее время начала			
7	9	18	0	18-0	18
8	9	18	0	18-0	18
6	7	18	2	Min(18-2; 18-2)	16
	8	18	2		
5	6	16	3	16-3	13
1	5	13	0	Min(13-1;0-0)	0
	11	0			

Полученные результаты раннего и позднего времени начала работ позволяют вычислить критический путь для нашего проекта (таблица 22).

Таблица 23 – Вычисление критического пути

Работа №	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
5	0	13	13
6	1	16	15
7	4	18	14
8	4	18	14
9	6	18	12
11	0	0	0
12	1	1	0
13	6	6	0
14	12	12	0
15	18	18	0
16	18	18	0
17	18	18	0
18	19	21	2
19	19	19	0
20	45	45	0
21	45	45	0
22	26	26	0
23	26	26	0
24	30	30	0
25	33	33	0
26	33	33	0

Из таблицы видно, что критический путь проходит через работы с номерами 1,11,12,13,14,15,16,17,19,20,21,22,23,24,25,26. Критический путь на сетевом графике представлен на рисунке Ж.2.

Таким образом, были определены оценки продолжительности операций и построена сетевая модель, каждая дуга которой отображает выполняемую работу, а также был найден критический путь.

6.2 Календарное планирование и оценка бюджета

При помощи системы управления проектами Microsoft Project выполним планирование задач проекта (рисунок 16).

	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Ср 22.04.20	Ср 22.04.20		
2	Выбор программных средств и архитектор	0 дней	Ср 22.04.20	Ср 22.04.20		
3	Программирование	26 дней	Ср 22.04.20	Чт 28.05.20		
4	Разработка базы данных	4 дней	Ср 22.04.20	Пн 27.04.20		
5	Постановка задачи	1 день	Ср 22.04.20	Ср 22.04.20	1	Программист 1
6	Проектирование базы данных	1 день	Чт 23.04.20	Чт 23.04.20	5	Программист 1[50%]
7	Разработка логической модели	2 дней	Пт 24.04.20	Пн 27.04.20	6	Программист 1;Компьютер
8	Разработка физической модели	2 дней	Пт 24.04.20	Пн 27.04.20	6	Программист 1;Компьютер
9	Разработка БД завершена	0 дней	Пн 27.04.20	Пн 27.04.20	7,8	
10	Разработка клиентской части	18 дней	Ср 22.04.20	Пн 18.05.20		
11	Постановка задачи	1 день	Ср 22.04.20	Ср 22.04.20	1	Программист 2
12	Проектирование интерфейса	5 дней	Чт 23.04.20	Ср 29.04.20	11	Программист 2[50%]
13	Разработка шаблонов html-страниц	6 дней	Чт 30.04.20	Пт 08.05.20	12	Программист 2;Компьютер
14	Стилизация и добавление динамических элементов	6 дней	Пн 11.05.20	Пн 18.05.20	13	Программист 2;Компьютер
15	Разработка клиентской части завершена	0 дней	Пн 18.05.20	Пн 18.05.20	14	
16	Разработка серверной части	8 дней	Вт 19.05.20	Чт 28.05.20		
17	Постановка задачи	1 день	Вт 19.05.20	Вт 19.05.20	15,9	Программист 1;Программист 2
18	Разработка уровня модели	5 дней	Ср 20.05.20	Вт 26.05.20	17	Программист 1;Программист 2
19	Разработка уровня контроллера	7 дней	Ср 20.05.20	Чт 28.05.20	17	Программист 1;Программист 2
20	Разработка серверной части завершена	0 дней	Чт 28.05.20	Чт 28.05.20	18,19	
21	Программирование завершено	0 дней	Чт 28.05.20	Чт 28.05.20	20	
22	Отладка	7 дней	Пт 29.05.20	Пн 08.06.20		
23	Тестирование работы приложения	4 дней	Пт 29.05.20	Ср 03.06.20	21	Программист 1;Компьютер
24	Исправление ошибок	3 дней	Чт 04.06.20	Пн 08.06.20	23	Программист 1;Компьютер
25	Отладка завершена	0 дней	Пн 08.06.20	Пн 08.06.20	24	
26	Конец проекта	0 дней	Пн 08.06.20	Пн 08.06.20	25	

Рисунок 16 – Планирование задач в Microsoft Project

Microsoft Project по задачам, введенным в системе автоматически выстроил диаграмму Ганта (рисунок И.1), а также построил сетевой график, представленный на рисунках И.2 – И.4.

Сетевой график показывает, что критический путь проходит через задачи с номерами 1,11,12,13,14,15,16,17,19,20,21,22,23,24,25,26. Из этого можно сделать вывод, что построенный программой график и определенный ею критический путь совпадают с теми, что были вычислены вручную.

На рисунке 17 изображен обзор проекта, на котором можно увидеть подсчитанную Microsoft Project длительность реализации проекта – 45 дней (что совпадает с вычисленной ранее вручную), и предполагаемые затраты – 37 870 рублей. Данная сумма является приемлемой для небольшой команды разработчиков.

BookingMedecines diakov.net			
на дату Ср 22.04.20			
Даты			
Начало:	Ср 22.04.20	Окончание:	Пн 08.06.20
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
Длительность			
Текущий план:	33 дней	Оставш ася:	33 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	33 дней	Процент завершения:	0%
Трудозатраты			
Текущий план:	412 ч	Оставш иеся:	412 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	412 ч	Процент завершения:	0%
Затраты			
Текущий план:	37 870,00 Р	Оставш иеся:	37 870,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	37 870,00 Р		
Состояние задачи		Состояние ресурса	
Еще не начатые задачи	26	Трудовые ресурсы:	0
Выполняющиеся задачи	0	Трудовые ресурсы с прев. дост	2
Завершенные задачи:	0	Материальные ресурсы:	2
Итого задач:	26	Итого ресурсов:	4

Рисунок 17 – Обзор проекта

На рисунках И.5 – И.8 представлены графики использования ресурсов.

Таким образом, было выполнено структурное и календарное планирование для разрабатываемой системы заказа медицинских препаратов. Метод анализа сроков (ранних и поздних) начала и окончания

нереализованных частей проекта позволил увязать выполнение различных работ и процессов во времени, получить прогноз общей продолжительности реализации всего проекта. Благодаря календарному планированию удалось детально спланировать задачи проекта до уровня отдельных работ, назначая ответственных, сформировать индивидуальный план и проконтролировать изменения по ходу проекта.

Заключение

В рамках данной выпускной квалификационной работы была разработана информационная система для заказа лекарственных препаратов. Был проведен анализ предметной области и анализ функциональных требований. Были выполнены все поставленные задачи:

- регистрация и авторизация пользователей;
- управлением учетной записью пользователя;
- поиск необходимого лекарства по названию или критерию;
- оформление итогового заказа;
- просмотр и управление заявками заказов пользователя;
- оперативная поддержка информации о медикаментах, заявках, клиентах и сотрудниках (добавление, редактирование, удаление) в базе данных, а также поиск и сортировка;
- управление хранилищем лекарств.

Была спроектирована и реализована база данных для хранения информации о медицинских препаратах, пользователях, сотрудниках, аптеках. Была разработана панель управления хранилищем препаратов, а также механизма навигации по portalу.

Корректность работы программы была проверена с помощью функционального тестирования методами «Черного ящика» и структурного тестирования с помощью методов «Белого ящика». Тестирование показало правильность работы программы.

Сконструированное веб-приложение является удобным средством учета и ведения фармацевтических данных, а также оформления заявок на покупку медикаментов.

В качестве платформы разработки была использована платформа ASP .Net. Приложение разработано в среде программирования Visual Studio 2017 на языке C#.

Список использованных источников

1. Ставская, Н.Е. Методические подходы к совершенствованию деятельности аптечных учреждений на основе использования современных информационных технологий: дис. канд. фарм. наук: 22.05.2020 – Тула, 2009. – 193с.
2. Манвелян Э.А., Богушевич Е.В., Манвелян М.М., Степанян С.А. Информационные технологии в фармации. 2014, № 3 [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/informatsionnye-tehnologii-v-farmatsii-1/viewer> (дата обращения: 01.05.2020)
3. Автоматизация аптек и аптечных сетей // Единая региональная система «Заказ» [Электронный ресурс]. URL: <http://www.qwerty.perm.ru/solutions/systemorders/> (дата обращения: 01.05.2020)
4. Интернет-бронирование лекарств «Фармация» // Оформление заказа [Электронный ресурс]. URL: https://krasnoyarsk.24farmacia.ru/help/oformlenie_zakaza/ (дата обращения: 04.05.2020)
5. Батрова Р.Г., Глухов С.В. Календарное планирование программ сетевыми методами [Электронный ресурс]. URL: <http://www.ict.nsc.ru/ws/Lyap2001/2226/> (дата обращения: 25.03.2020)
6. Садовский И.Д., Глухов С.В. Применение модели COCOMO II для оценки разработки программного обеспечения в Windows проектах [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/primenenie-modeli-cocomo-ii-dlya-otsenki-razrabotki-programmnogo-obespecheniya-v-windows-proektah/viewer> (дата обращения: 14.04.2020)
7. Алексеев А., Савельев А. Технология Microsoft для создания RIA-приложений // ASP.Net 4.0 [Электронный ресурс]. URL: <https://www.intuit.ru/studies/courses/670/526/lecture/11825> (дата обращения: 15.05.2020)
8. Архитектура MVC [Электронный ресурс]. URL: <https://github.com/codedokode/pasta/blob/master/arch/mvc.md> (дата обращения: 19.05.2020)

9. Архитектура Web-приложений, публикующих БД [Электронный ресурс]. URL: http://saukpgp.ru/sauk/Base_UMM/pgp/dis5/t10_1.htm (дата обращения: 22.05.2020)
10. Харламова Т.Н., Герасимов И.Б., Злобина Н.В. Управление затратами на качество продукции: отечественный и зарубежный опыт [Электронный ресурс]. URL: <http://window.edu.ru/catalog/pdf2txt/728/38728/16506> (дата обращения: 14.04.2020)
11. Скороход С. Управление проектами средствами Microsoft Project // Сетевое планирование и управление [Электронный ресурс]. URL: <https://www.intuit.ru/studies/courses/496/352/lecture/8389> (дата обращения: 01.06.2020)
12. Количество строк кода [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Количество_строк_кода (дата обращения: 03.06.2020)
13. Ледовских И. Метрики сложности кода [Электронный ресурс]. URL: https://www.ispras.ru/preprints/docs/prep_25_2013.pdf (дата обращения: 04.06.2020)

Приложение А
СХЕМЫ МОДЕЛИ БАЗЫ ДАННЫХ
(рекомендуемое)

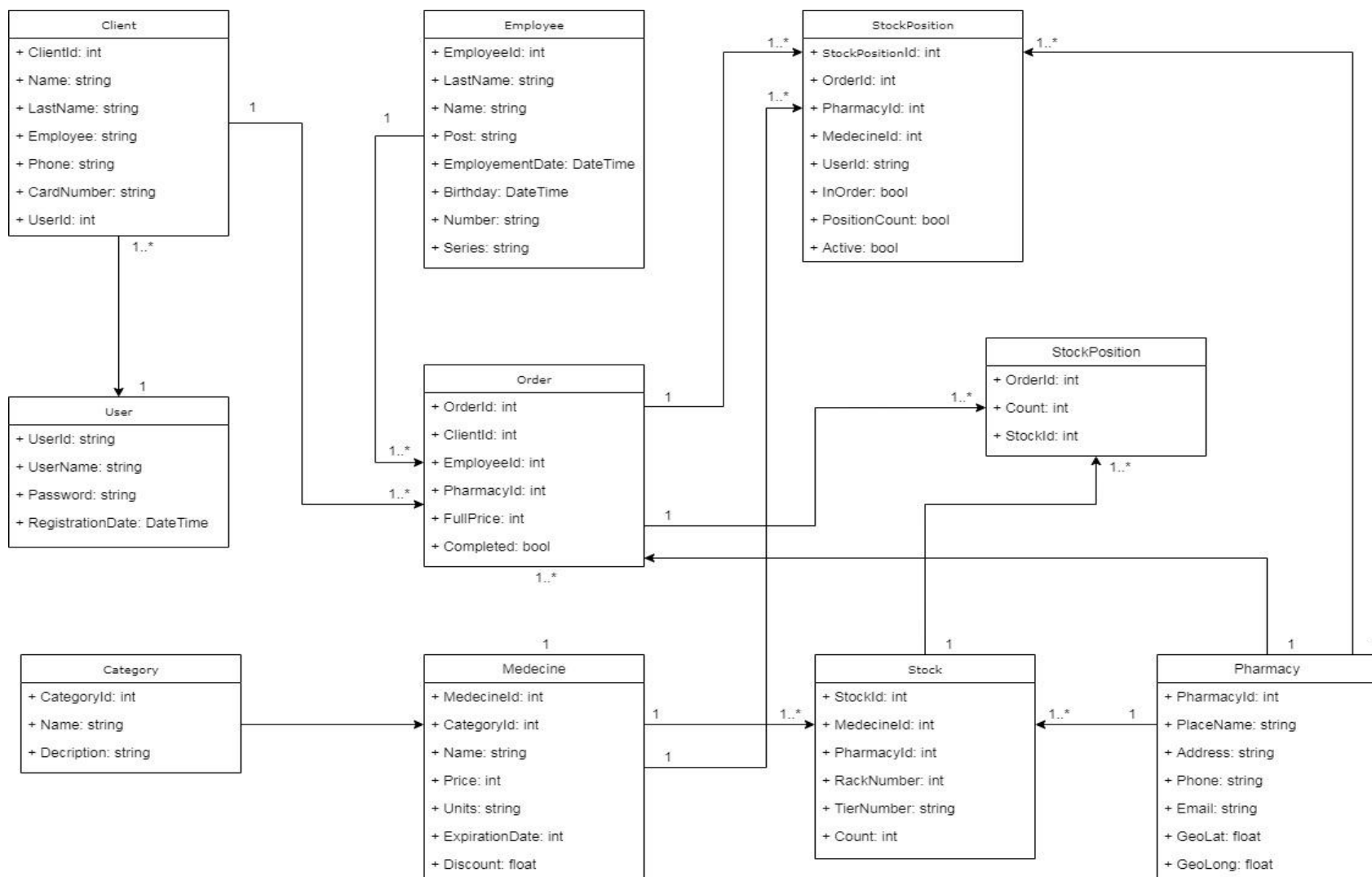


Рисунок А.1 – Диаграмма классов сущностей на концептуальном уровне

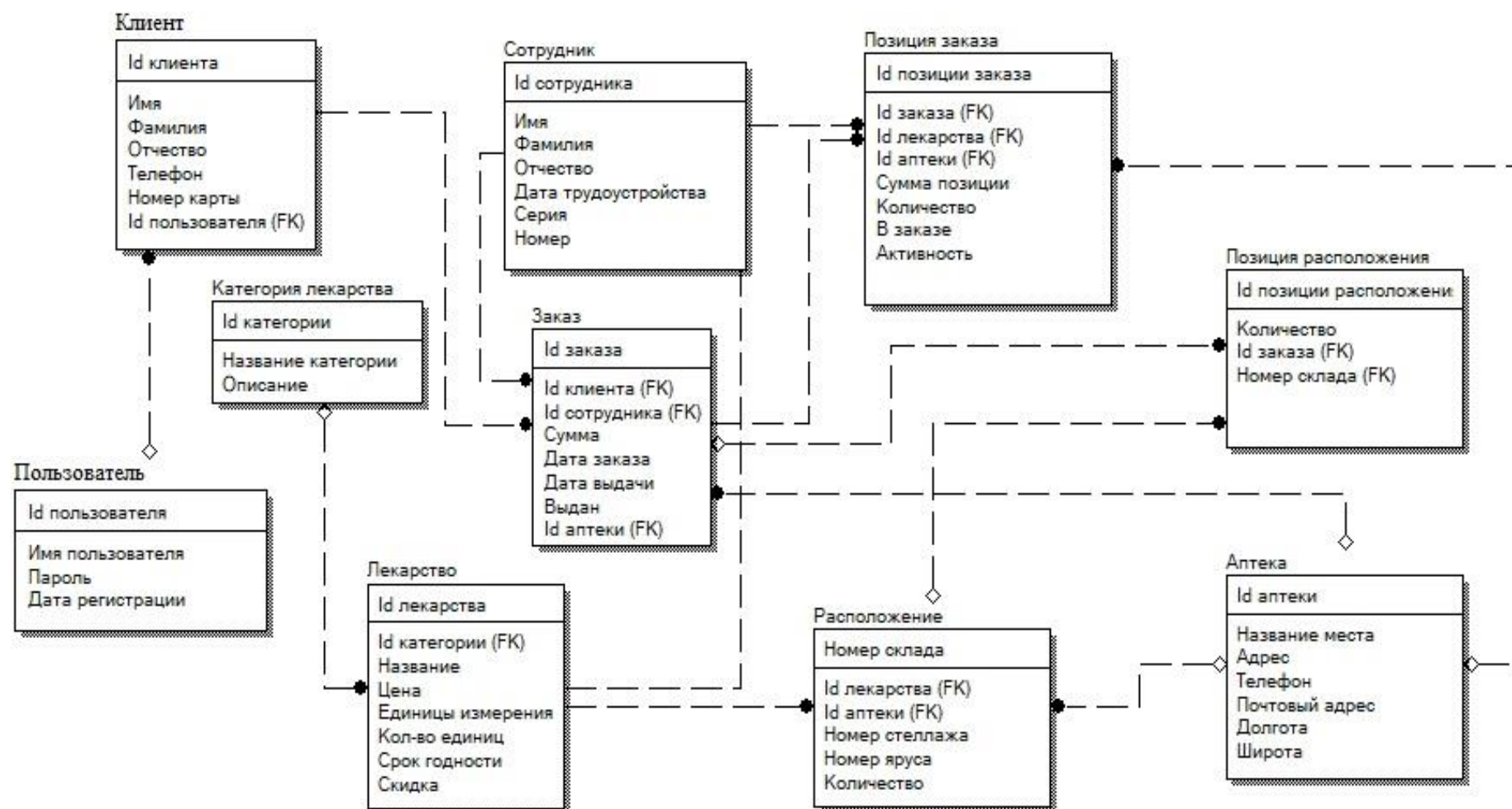


Рисунок А.2 – Логическая модель базы данных

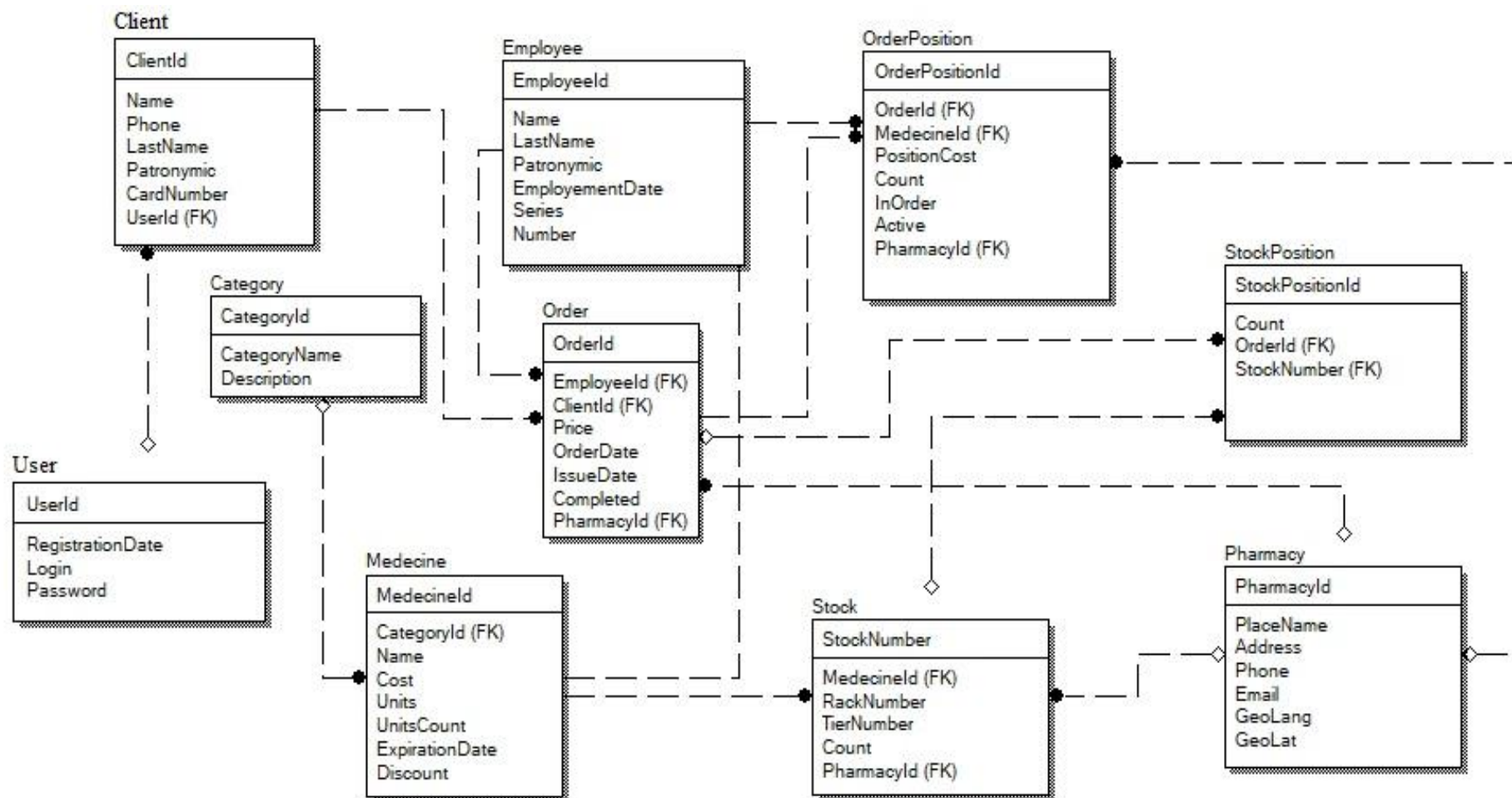


Рисунок А.3 – Физическая модель базы данных

Приложение Б
ДИАГРАММЫ КЛАССОВ УРОВНЯ МОДЕЛИ
(рекомендуемое)

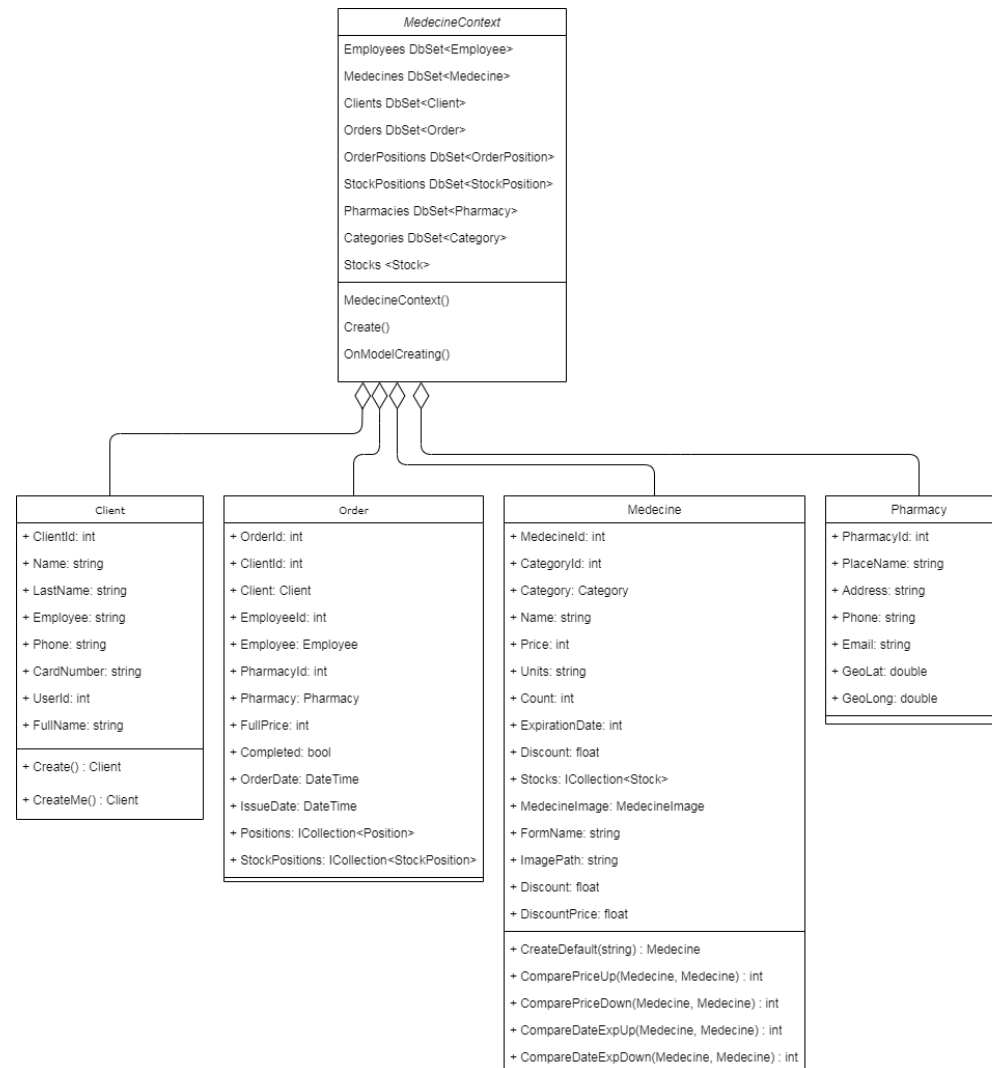


Рисунок Б.1 – Диаграмма классов уровня модели

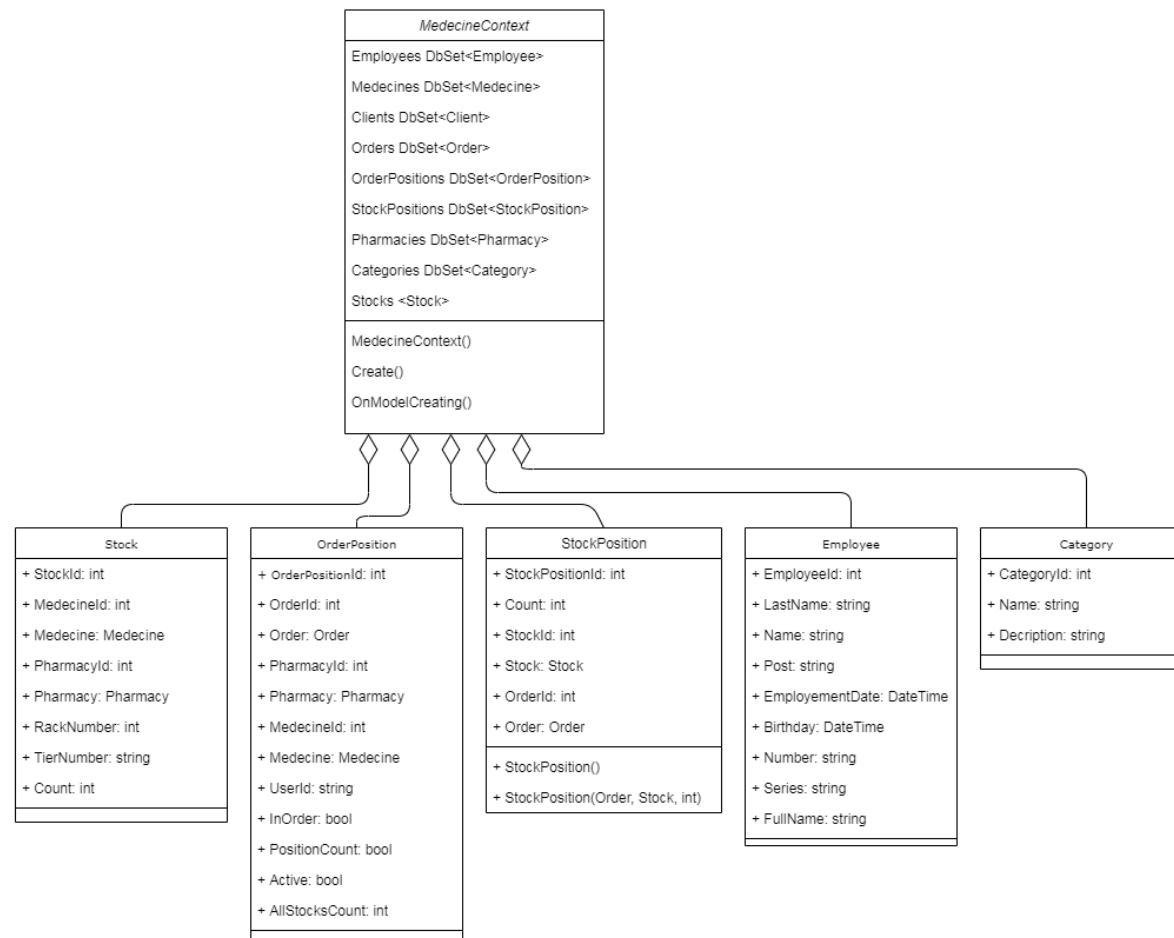


Рисунок Б.2 – Диаграмма классов уровня модели

Приложение В
ДИАГРАММЫ КЛАССОВ УРОВНЯ КОНТРОЛЛЕРОВ
(рекомендуемое)

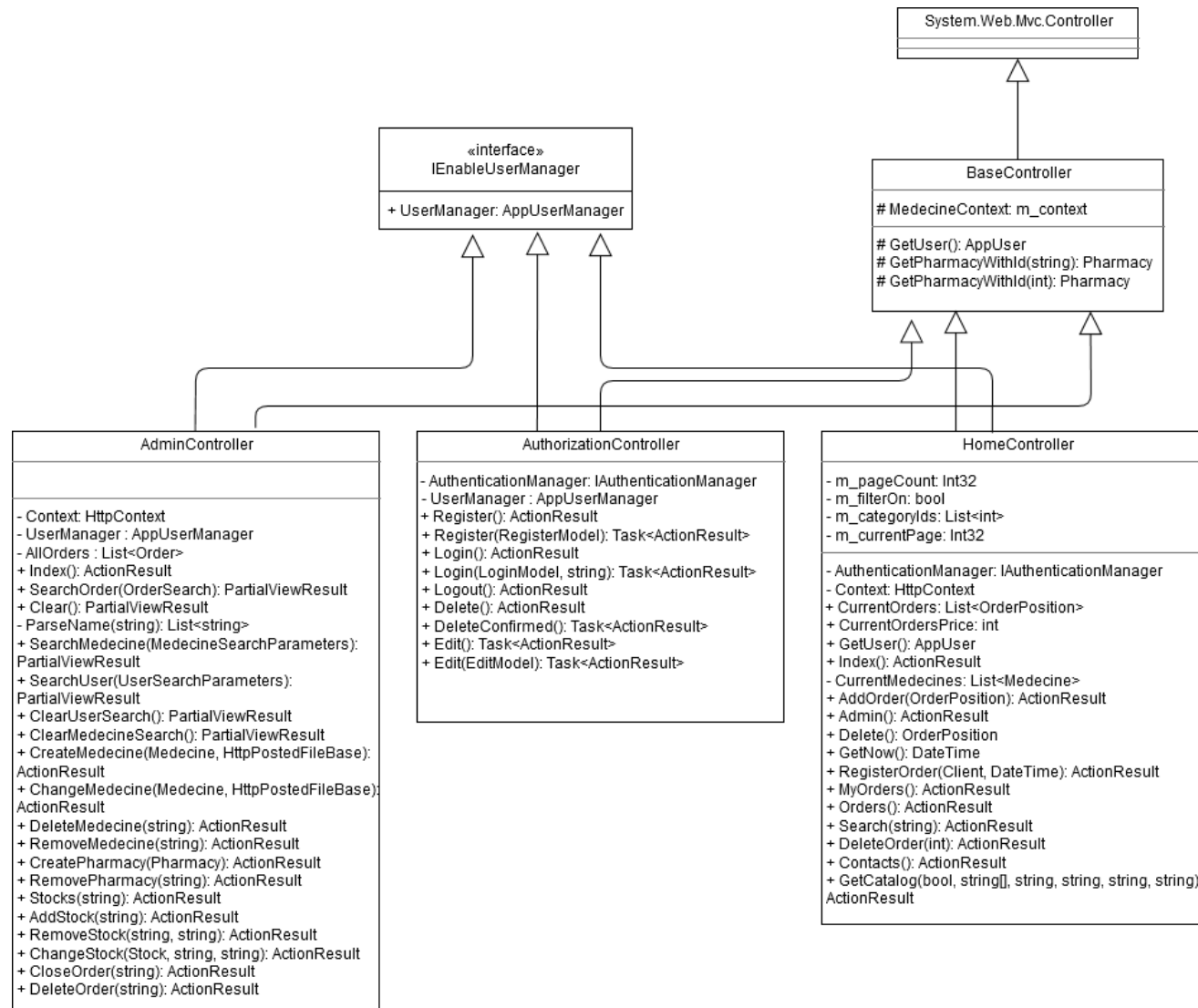


Рисунок В.1 – Диаграмма классов контроллеров

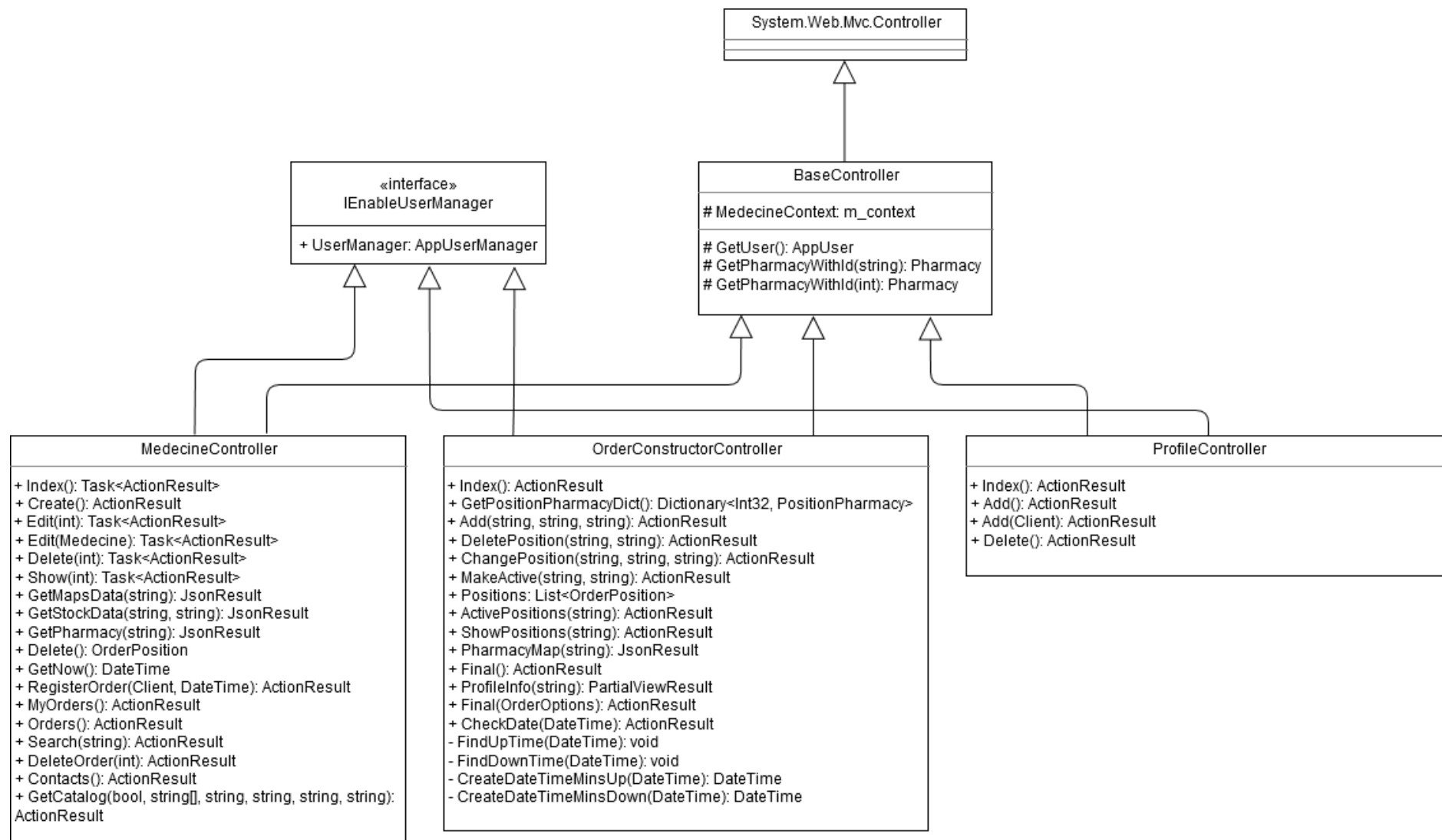


Рисунок В.2 – Диаграмма классов контроллеров

Приложение Г
ИСХОДНЫЙ КОД КЛАССОВ КОНТРОЛЛЕРОВ
(рекомендуемое)

Листинг Г.1 – AdminController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.Mvc;
using System.Web.UI.WebControls;
using BookingMedecines.Models;
using System.Data.Entity;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;

namespace BookingMedecines.Controllers
{
    public class AdminController : BaseController,
        IEnableUserManager
    {
        public AppUserManager UserManager
        {
            get { return HttpContext.GetOwinContext().
                UserManager<AppUserManager>(); }
        }

        private HttpContext Context
        {
            get => HttpContext.ApplicationInstance.Context;
        }

        List<Order> AllOrders
        {
            get => m_context.Orders.
                Include(o => o.Client).
                Include(o => o.Employee).
                Include(o => o.Positions).
                Include(o => o.Pharmacy).
                Include(o => o.StockPositions.Select(sp =>
                    sp.Stock.Medecine)).
                ToList();
        }

        // GET: Admin
        [Authorize(Roles = "admin")]
        public ActionResult Index()
        {
            ViewBag.Stats = GenerateChart();
            ViewBag.Orders = AllOrders;
            var pos = m_context.Positions.
                Include(p => p.Medecine).
                Include(p => p.Order).
                Include(p =>
                    p.Medecine.Category).ToList();

```

```

ViewBag.StockPositions =
    m_context.StockPositions.Include(sp =>
        sp.Stock);
ViewBag.Pharmacies = m_context.Pharmacies;
ViewBag.Medecines = m_context.Medecines.
    Include(m => m.Category).Include(m =>
        m.MedecineImage);
var users = m_context.Users.Include(u =>
    u.Clients).Select(u => (AppUser)u).ToList();

foreach (var user in users)
    user.RoleNames =
        UserManager.GetRoles(user.Id).ToList();

ViewBag.Users = users;
ViewBag.Categories = m_context.Categories;
ViewBag.CategoryId = new
    SelectList(m_context.Categories,
        "CategoryId", "Name");
ViewBag.Employees = m_context.Employees;
return View();
}

[HttpPost]
public PartialViewResult SearchOrder(
    OrderSearch orderParameters)
{
    ViewBag.Stats = GenerateChart();
    var searchOrders = AllOrders;
    var pos = m_context.Positions.
        Include(p => p.Medecine).
        Include(p => p.Order).
        Include(p =>
            p.Medecine.Category).ToList();

    if (orderParameters.ClientName != null)
    {
        var clientNameParts =
            ParseName(orderParameters.ClientName);
        var clientNameOrders = new List<Order>();
        foreach (var order in searchOrders)
        {
            Int32 contains = 0;
            foreach (var namePart in
                clientNameParts)
            {
                if (order.Client.FullName.Contains(
                    namePart))
                    contains++;
            }
            if (contains == clientNameParts.Count)
                clientNameOrders.Add(order);
        }
    }
}

```

```

    }

    searchOrders = clientNameOrders;
}

if (orderParameters.EmployeeName != null)
{
    var employeeNameParts =
        ParseName(orderParameters.EmployeeName);
    var employeeNameOrders = new List<Order>();
    foreach (var order in searchOrders)
    {
        Int32 contains = 0;
        foreach (var namePart in
            employeeNameParts)
        {
            if (order.Employee.FullName.
                Contains(namePart))
                contains++;
        }
        if (contains == employeeNameParts.Count)
            employeeNameOrders.Add(order);
    }

    searchOrders = employeeNameOrders;
}

if (orderParameters.OrderNumber != 0)
searchOrders =
    searchOrders.Where(o => o.OrderId ==
        orderParameters.OrderNumber).ToList();

if (orderParameters.Sum != 0)
    searchOrders = searchOrders.Where(o =>
        o.FullPrice == orderParameters.Sum).
        ToList();

var nullDate = new DateTime();
if (orderParameters.OrderDate != nullDate)
    searchOrders =
        searchOrders.Where(o => o.OrderDate ==
            orderParameters.OrderDate).
            ToList();

if (orderParameters.IssueDate != nullDate)
    searchOrders = searchOrders.Where(o =>
        o.IssueDate ==
            orderParameters.IssueDate).
            ToList();

var sortMode = orderParameters.SortMode;
if (sortMode != null && !sortMode.Equals("all"))
{

```

```

        if (sortMode.Equals("actual"))
            searchOrders = searchOrders.Where(o =>
                o.IssueDate > DateTime.Now).
                ToList();
        else if (sortMode.Equals("closed"))
            searchOrders = searchOrders.Where(o =>
                o.Completed).ToList();
        else if (sortMode.Equals("open"))
            searchOrders = searchOrders.Where(o =>
                !o.Completed).ToList();
        else if (sortMode.Equals("outOfDate"))
            searchOrders = searchOrders.Where(o =>
                o.IssueDate < DateTime.Now &&
                !o.Completed).ToList();

        searchOrders.OrderByDescending(o =>
            o.IssueDate);
    }

    ViewBag.Orders = searchOrders;
    return PartialView("OrderList");
}

public PartialViewResult Clear()
{
    return SearchOrder(new OrderSearch());
}

private List<string> ParseName(string name)
{
    List<string> nameParts = new List<string>();
    while (name.Contains(" "))
    {
        var Index = name.IndexOf(" ");
        nameParts.Add(name.Substring(0, Index));
        name = name.Substring(Index + 1,
            name.Length - Index - 1);
    }

    nameParts.Add(name);
    return nameParts;
}

public PartialViewResult
    SearchMedecine(MedecineSearchParameters
        parameters)
{
    var medecines = m_context.Medecines.Include(m =>
        m.Category).Include(m =>
            m.MedecineImage).ToList();

    if (parameters.MedecineId != 0)
        medecines = medecines.Where(m =>

```

```

        m.MedecineId ==
            parameters.MedecineId).ToList();

    if (parameters.CategoryName != null)
        medecines = medecines.Where(m =>
            m.Category.Name ==
                parameters.CategoryName).ToList();

    if (parameters.Discount != 0)
        medecines = medecines.Where(m => m.Discount
            == parameters.Discount).ToList();

    if (parameters.Price != 0)
        medecines = medecines.Where(m => m.Price ==
            parameters.Price).ToList();

    if (parameters.ExpirationDate != 0)
        medecines = medecines.Where(m =>
            m.ExpirationDate ==
                parameters.ExpirationDate).
            ToList();

    if (parameters.MedecineName != null)
        medecines = medecines.Where(m => m.Name ==
            parameters.MedecineName).ToList();

    ViewBag.Categories = m_context.Categories;
    ViewBag.CategoryId = new
        SelectList(m_context.Categories,
            "CategoryId", "Name");
    ViewBag.Medecines = medecines;
    return PartialView("MedecineList");
}

public PartialViewResult
    SearchUser(UserSearchParameters parameters)
{
    var userListResult = m_context.Users.Include(u =>
        u.Clients).ToList();
    foreach (var user in userListResult)
        user.RoleNames =
            UserManager.GetRoles(user.Id);

    if (parameters.UserName != null)
        userListResult = userListResult.Where(u =>
            u.UserName == parameters.UserName).
            ToList();
    if (parameters.Phone != null)
        userListResult = userListResult.Where(u =>
            u.PhoneNumber == parameters.Phone).
            ToList();

    AppUser[] userListArray;

```

```

if (parameters.CName != null)
{
    userListArray = userListResult.ToArray();
    for (int i = 0;
        i < userListResult.Count(); i++)
    {
        var user = userListArray[i];
        var clients = user.Clients;
        bool result = false;
        foreach (var client in clients)
        {
            if (client.Name.Contains(
                parameters.CName))
            {
                result = true;
                break;
            }
        }

        if (!result)
            userListResult.Remove(user);
    }
}

if (parameters.LastName != null)
{
    userListArray = userListResult.ToArray();
    for (int j = 0;
        j < userListResult.Count(); j++)
    {
        var user = userListArray[j];
        var clients = user.Clients;
        bool result = false;
        foreach (var client in clients)
        {
            if (client.LastName.
                Contains(parameters.LastName))
            {
                result = true;
                break;
            }
        }

        if (!result)
            userListResult.Remove(user);
    }
}

if (parameters.Phone != null)
    userListResult = userListResult.Where(u =>
        u.PhoneNumber == parameters.Phone).ToList();
ViewBag.Users = userListResult;
return PartialView("ClientList");

```

```

    }

    public PartialViewResult ClearUserSearch()
    {
        return SearchUser(new UserSearchParameters());
    }

    public PartialViewResult ClearMedecineSearch()
    {
        return SearchMedecine(
            new MedecineSearchParameters());
    }

    [HttpPost]
    public ActionResult CreateMedecine(Medecine medecine,
        HttpPostedFileBase upload)
    {
        if (upload != null)
        {
            // получаем имя файла
            string filePath = @"..\Content\" +
                System.IO.Path.GetFileName(
                    upload.FileName);
            // сохраняем файл в папку Files
            // в проекте
            upload.SaveAs(Server.MapPath(filePath));
            medecine.MedecineImage = new MedecineImage();
            medecine.MedecineImage.MedecineImageId =
                medecine.MedecineId;
            medecine.MedecineImage.Image = filePath;
        }

        m_context.Medecines.Add(medecine);
        m_context.SaveChanges();
        return RedirectPermanent("/Admin/Index/#panel4c");
    }

    [HttpPost]
    public ActionResult ChangeMedecine(Medecine medecine,
        HttpPostedFileBase upload)
    {
        if (upload != null)
        {
            // получаем имя файла
            string filePath = @"..\Content\" +
                System.IO.Path.GetFileName(upload.
                    FileName);
            // сохраняем файл в папку Files в проекте
            upload.SaveAs(Server.MapPath(filePath));
        }
    }

```

```

        medecine.MedecineImage = new MedecineImage();
        medecine.MedecineImage.MedecineImageId =
            medecine.MedecineId;
        medecine.MedecineImage.Image = filePath;
    }

    m_context.Entry(medecine).State =
        EntityState.Modified;
    m_context.SaveChanges();
    return RedirectPermanent("/Admin/Index/#panel4c");
}

public ActionResult DeleteMedecine(string medId)
{
    var id = Int32.Parse(medId);
    var medecine = m_context.Medecines.First(m =>
        m.MedecineId == id);
    m_context.Medecines.Remove(medecine);

    foreach (var stock in medecine.Stocks)
        m_context.Stocks.Remove(stock);
    m_context.SaveChanges();
    return RedirectPermanent("/Admin/Index/#panel4c");
}

public ActionResult RemoveMedecine(string medId)
{
    var id = Int32.Parse(medId);
    var medecine = m_context.Medecines.Include(m =>
        m.Stocks).First(m => m.MedecineId == id);
    m_context.Medecines.Remove(medecine);

    foreach (var stock in medecine.Stocks)
        m_context.Stocks.Remove(stock);
    m_context.SaveChanges();
    return RedirectPermanent("/Admin/Index/#panel4c");
}

[HttpPost]
public ActionResult CreatePharmacy(Pharmacy pharmacy)
{
    m_context.Pharmacies.Add(pharmacy);
    m_context.SaveChanges();
    return RedirectPermanent("/Admin/Index/");
}

public ActionResult RemovePharmacy(string pharmacyId)
{
    var id = Int32.Parse(pharmacyId);

```

```

        var pharmacy =
            m_context.Pharmacies.
                FirstOrDefault(p => p.PharmacyId == id);
        if (pharmacy != null)
        {
            m_context.Pharmacies.Remove(pharmacy);
            m_context.SaveChanges();
        }

        return RedirectPermanent("/Admin/Index/");
    }

    public ActionResult Stocks(string medId)
    {
        var id = Int32.Parse(medId);

        var stocks = m_context.Stocks.Include(s =>
            s.Pharmacy).Where(s => s.Medecine.MedecineId
                == id).ToList();

        var pharmStock = new Dictionary<Pharmacy,
            List<Stock>>();
        foreach (var stock in stocks)
        {
            if (pharmStock.ContainsKey(stock.Pharmacy))
                pharmStock[stock.Pharmacy].Add(stock);
            else
            {
                var stockList = new List<Stock>();
                stockList.Add(stock);
                pharmStock.Add(stock.Pharmacy,
                    stockList);
            }
        }

        ViewBag.Stocks = pharmStock;
        ViewBag.Medecine = m_context.Medecines.Include(
            m => m.Category).
            First(m => m.MedecineId == id);
        ViewBag.Pharmacies = m_context.Pharmacies;
        return View("StockControl");
    }

    [HttpPost]
    public ActionResult AddStock(Stock stock,
        string placeName, string medecineId)
    {
        stock.Pharmacy =
            m_context.Pharmacies.FirstOrDefault(p =>
                placeName.Contains(p.PlaceName));
        var medId = Int32.Parse(medecineId);
        stock.Medecine =
            m_context.Medecines.FirstOrDefault(m =>

```

```

        m.MedecineId == medId);
m_context.Stocks.Add(stock);
m_context.SaveChanges();
return Stocks(medecineId);
}

public ActionResult RemoveStock(string medecineId,
    string stockId)
{
    var stId = Int32.Parse(stockId);
    var stock = m_context.Stocks.FirstOrDefault(s =>
        s.StockId == stId);
    m_context.Stocks.Remove(stock);
    m_context.SaveChanges();
    return Stocks(medecineId);
}

[HttpPost]
public ActionResult ChangeStock(Stock stock,
    string placeName, string medecineId)
{
    stock.Pharmacy =
        m_context.Pharmacies.FirstOrDefault(p =>
            placeName.Contains(p.PlaceName));
    var medId = Int32.Parse(medecineId);
    stock.Medecine =
        m_context.Medecines.FirstOrDefault(m =>
            m.MedecineId == medId);
    m_context.Stocks.Add(stock);
    m_context.SaveChanges();
    return Stocks(medecineId);
}

public ActionResult CloseOrder(string orderId)
{
    var id = Int32.Parse(orderId);
    var order = m_context.Orders.FirstOrDefault(o =>
        o.OrderId == id);
    var stockPositions =
        order.StockPositions.ToList();
    if (order != null)
    {
        for (int i = 0;
            i < stockPositions.Count; i++)
        {
            var position = stockPositions[i];
            m_context.StockPositions.
                Remove(position);
        }
        order.Completed = true;
    }

    m_context.SaveChanges();
}

```

```

        return RedirectPermanent("/Admin/Index/");
    }

    public ActionResult DeleteOrder(string orderId)
    {
        var id = Int32.Parse(orderId);
        var order = m_context.Orders.FirstOrDefault(o =>
            o.OrderId == id);

        if (order != null)
        {
            var positions = order.Positions.ToList();
            for (int i = 0; i < positions.Count; i++)
            {
                var position = positions[i];
                m_context.Positions.Remove(position);
            }

            var stockPositions =
                order.StockPositions.ToList();
            for (int i = 0;
                i < stockPositions.Count; i++)
            {
                var stockPosition = stockPositions[i];
                m_context.StockPositions.
                    Remove(stockPosition);
            }

            m_context.Orders.Remove(order);
        }

        m_context.SaveChanges();
        return RedirectPermanent("/Admin/Index/");
    }
}

```

Листинг Г.2 – AuthorizationController.cs

```

using BookingMedecines.Models;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;

namespace BookingMedecines.Controllers
{
    public class AuthorizationController : Controller,

```

```

IEnableUserManager
{
    public AppUserManager UserManager
    {
        get
        {
            Return HttpContext.GetOwinContext().
                GetUserManager<AppUserManager>();
        }
    }

    private IAuthenticationManager AuthenticationManager
    {
        get
        {
            return HttpContext.GetOwinContext().
                Authentication;
        }
    }

    public ActionResult Register()
    {
        return View();
    }

    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> Register(
        RegisterModel model)
    {
        if (ModelState.IsValid)
        {
            AppUser user = new AppUser {
                UserName = model.Email,
                Email = model.Email,
                RegistrationDate = DateTime.Now };
            IdentityResult result = await
                UserManager.CreateAsync(user,
                    model.Password);
            if (result.Succeeded)
            {
                await UserManager.
                    AddToRoleAsync(user.Id, "user");
                return RedirectToAction("Login",
                    "Authorization");
            }
            else
            {
                foreach (string error in result.Errors)
                {
                    ModelState.AddModelError("",
                        error);
                }
            }
        }
    }
}

```

```

        }
    }
    return View(model);
}

public ActionResult Login()
{
    ViewBag returnUrl = Request.Url.ToString();
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(LoginModel model,
    string returnUrl)
{
    if (ModelState.IsValid)
    {
        AppUser user = await
            UserManager.FindAsync(model.Email,
                model.Password);
        if (user == null)
        {
            ModelState.AddModelError("",
                "Неверный логин или пароль.");
        }
        else
        {
            ClaimsIdentity claim = await
                UserManager.
                    CreateIdentityAsync(user,
                        DefaultAuthenticationTypes.
                            ApplicationCookie);
            AuthenticationManager.SignOut();
            AuthenticationManager.SignIn(
                new AuthenticationProperties
                {
                    IsPersistent = true
                }, claim);
            return RedirectToAction("Index",
                "Home");
        }
    }
    ViewBag returnUrl = returnUrl;
    return View(model);
}

public ActionResult Logout()
{
    AuthenticationManager.SignOut();
    return RedirectToAction("Index", "Home");
}

```

```

    }

    [HttpGet]
    public ActionResult Delete()
    {
        return View();
    }

    [HttpPost]
    [ActionName("Delete")]
    public async Task<ActionResult> DeleteConfirmed()
    {
        AppUser user = await
            UserManager.FindByEmailAsync(User.Identity.
                Name);
        if (user != null)
        {
            IdentityResult result = await
                UserManager.DeleteAsync(user);
            if (result.Succeeded)
            {
                return RedirectToAction("Logout",
                    "Account");
            }
        }
        return RedirectToAction("Index", "Home");
    }

    public async Task<ActionResult> Edit()
    {
        AppUser user = await
            UserManager.FindByEmailAsync(User.Identity.
                Name);
        if (user != null)
        {
            EditModel model = new EditModel {
                RegistrationDate =
                    user.RegistrationDate };
            return View(model);
        }
        return RedirectToAction("Login", "Account");
    }

    [HttpPost]
    public async Task<ActionResult> Edit(EditModel model)
    {
        AppUser user = await
            UserManager.FindByEmailAsync(User.Identity.
                Name);
        if (user != null)
        {
            user.RegistrationDate =
                model.RegistrationDate;

```

```

        IdentityResult result = await
            UserManager.UpdateAsync(user);
        if (result.Succeeded)
        {
            return RedirectToAction("Index",
                "Home");
        }
        else
        {
            ModelState.AddModelError("",
                "Что-то пошло не так");
        }
    }
    else
    {
        ModelState.AddModelError("",
            "Пользователь не найден");
    }
    return View(model);
}
}
}

```

Листинг Г.3 – HomeController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Caching;
using System.Web.Mvc;
using System.Web.Routing;
using System.Web.UI;
using BookingMedecines.Models;
using BookingMedecines.Utills;
using System.Data;
using System.Data.Entity;
using System.Threading.Tasks;
using System.Net;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;

namespace BookingMedecines.Controllers
{
    public class HomeController : BaseController,
        IEnableUserManager
    {
        public AppUserManager UserManager
        {
            get { return HttpContext.GetOwinContext().
                UserManager<AppUserManager>(); }
        }
    }
}

```

```

private HttpContext Context
{
    get => HttpContext.ApplicationInstance.Context;
}

private List<OrderPosition> CurrentOrders
{
    get { return (HttpContext.Cache["CurrentOrders"]
        as List<OrderPosition>); }
}

public int CurrentOrdersPrice
{
    get => CurrentOrders.Count > 0 ?
        CurrentOrders.Sum(x => x.PositionPrice) : 0;
}

private AppUser GetUser()
{
    return m_context.Users.FirstOrDefault(u =>
        u.UserName == User.Identity.Name);
}

private const Int32 m_pageCount = 6;
private bool m_filterOn = false;
private List<int> m_categoryIds = new List<int>();
private Int32 m_currentPage = 0;

public ActionResult Index()
{
    try
    {
        var medecines = CurrentMedecines;
        var pageCount = Math.
            Ceiling((Double) medecines.Count /
                (Double)m_pageCount);

        Cart cart = new Cart(GetUser()?.Id);
        if (cart.Enabled)
            ViewBag.Orders = cart.GetClientCart();
        else
            ViewBag.Orders = new
                List<OrderPosition>();
        ViewBag.Cost = cart.ComputeTotalValue();
        ViewBag.SearchString = SearchString;
        ViewBag.Categories =
            m_context.Categories.ToList();
        ViewBag.Pharmacies =
            m_context.Pharmacies.ToList();

        var skip = 0;
        var need = m_pageCount;
        ViewBag.Medecines = medecines.

```

```

        Skip(skip).
        Take(need > medecines.Count ?
            m_pageCount -
                (need - medecines.Count) :
                m_pageCount);

        ViewBag.CurrentPage = 0;
        ViewBag.PageCount = pageCount;
        return View();
    }
    catch (Exception e)
    {
        return HttpNotFound();
    }
}

private List<Medecine> CurrentMedecines
{
    get
    {
        List<Medecine> Result =
            SearchString ==
                String.Empty ?
                    m_context.Medecines.
                        Include(m => m.Category).
                        Include(m => m.Stocks).
                        ToList()
                    : m_context.Medecines.Where(
                        m => m.Name.ToLower().
                            Contains(SearchString.
                                ToLower())).
                        Include(m => m.Category).
                        Include(m => m.Stocks).
                        ToList();

        if (m_filterOn)
            Result = Result.
                Where(m => m_categoryIds.
                    Contains(m.CategoryId)).
                    ToList();

        return Result;
    }
}

[HttpGet]
public ActionResult AddOrder(OrderPosition position)
{
    return Index();
}

[HttpGet]
public ActionResult Admin()
{
    IList<string> Roles =

```

```

        new List<string> {"Поль не определена"};
        AppUser user = GetUser();

        Roles = UserManager.GetRoles(user.Id);
        if (user != null && Roles.Contains("admin"))
            return RedirectToRoute(new {controller =
                "Admin", action = "Index"});

        return RedirectToRoute(new {controller = "Home",
            action = "Index"});
    }

    [HttpPost]
    public void Delete(OrderPosition position)
    {
        CurrentOrders?.Remove(position);
    }

    private DateTime GetNow() => DateTime.Now;

    // Упрощение вызовов методов. Скрытие метода
    // public
    private void GetInfo()
    {
        string browser =
            HttpContext.Request.Browser.Browser;
        string user_agent = HttpContext.Request.UserAgent;
        string url = HttpContext.Request.RawUrl;
        string ip = HttpContext.Request.UserHostAddress;
        string referrer =
            HttpContext.Request.UrlReferrer == null
                ? "" :
                HttpContext.Request.UrlReferrer.
                    AbsoluteUri;
    }

    [HttpGet]
    public ActionResult RegisterOrder()
    {
        ViewBag.Orders = CurrentOrders;
        return View();
    }

    [HttpPost]
    public ActionResult RegisterOrder(Client client,
        DateTime issueDate)
    {
        if (User.Identity.IsAuthenticated)
        {
            client.AppUser = GetUser();
            //GetUser().Clients.Add(client);
        }
        else

```

```

    {
        return RedirectToRoute(new {controller =
            "Authorization", action = "Login"});
    }

m_context.Clients.Add(client);
Order order = new Order()
{
    Client = client,
    FullPrice = CurrentOrdersPrice,
    IssueDate = issueDate,
    OrderDate = DateTime.Now,
    EmployeeId = new Random().Next(1,
        m_context.Employees.Count()),
    Completed = false
};
m_context.Orders.Add(order);
foreach (var orderPosition in CurrentOrders)
{
    var position = new OrderPosition()
    {
        OrderId = order.OrderId,
        PositionCount =
            orderPosition.PositionCount,
        PositionPrice =
            orderPosition.PositionPrice,
        MedecineId = orderPosition.MedecineId
    };
    m_context.Positions.Add(position);
}

m_context.SaveChanges();
CurrentOrders.Clear();
return RedirectToAction("Index", "Home");
}

[HttpGet]
public ActionResult MyOrders()
{
    if (!User.Identity.IsAuthenticated)
        return RedirectToRoute(new {controller =
            "Authorization", action = "Login"});

    var CurrentUser = GetUser();
    var Orders = m_context.Orders.Include(o =>
        o.Client).
        Include(o => o.Employee).
        Include(o => o.Pharmacy);
    var pos = m_context.Positions.
        Include(p => p.Medecine).
        Include(p=>p.Order).
        Include(p=>p.Medecine.Category).
        ToList();
}

```

```

        ViewBag.Positions = pos;
        return View(Orders.Where(o =>
            o.Client.AppUser.UserName ==
                CurrentUser.UserName));
    }

    [HttpGet]
    public ActionResult Orders()
    {
        return View(m_context.Orders);
    }

    private static string SearchString = string.Empty;

    [HttpGet]
    public ActionResult Search(string name)
    {
        try
        {
            SearchString = name;
            ViewBag.Medecines = CurrentMedecines;
            ViewBag.SearchString = SearchString;
            ViewBag.Categories =
                m_context.Categories.ToList();
            return View("Index");
        }
        catch (Exception e)
        {
            return HttpNotFound();
        }
    }

    [HttpGet]
    public ActionResult DeleteOrder(int id)
    {
        var Order = m_context.Orders.Include(o =>
            o.StockPositions).
            First(x => x.OrderId == id);

        var PosArray = m_context.Positions.ToArray();
        for (int i = 0;
            i < m_context.Positions.Count(); i++)
        {
            if (PosArray[i].OrderId == id)
                m_context.Positions.Remove(PosArray[i]);
        }

        foreach (var stockPos in Order.StockPositions)
        {
            var stock =
                m_context.Stocks.FirstOrDefault(s =>
                    s.StockId == stockPos.StockId);
            if (stock == null)

```

```

        stock = m_context.Stocks.First();

        stock.StockCount += stockPos.Count;
    }

    m_context.Orders.Remove(Order);
    m_context.SaveChanges();
    return RedirectToAction("MyOrders", "Home");
}

public ActionResult Contacts()
{
    ViewBag.Pharmacies = m_context.Pharmacies;
    return View();
}

public ActionResult GetCatalog(bool filterOn,
    string[] category_ids, string price, string page,
    string pharmId, string sortName)
{
    if (filterOn)
        m_categoryIds.
            AddRange(category_ids.
                Select(s => Int32.Parse(s)).
                    ToArray());
    m_filterOn = filterOn;
    var medicines = CurrentMedecines.
        Where(m => m.Price <= Int32.Parse(price)).
            ToList();

    var pharmacyId = Int32.Parse(pharmId);
    if (pharmacyId != 0)
    {
        var medicinesInPharmacy =
            new List<Medecine>();
        foreach (var medecine in medicines)
        {
            var stock = m_context.Stocks.
                FirstOrDefault(s =>
                    s.MedecineId == medecine.MedecineId
                    && s.PharmacyId ==
                        pharmacyId);
            if (stock != null && stock.StockCount > 0)
                medicinesInPharmacy.Add(medecine);
        }

        medicines = medicinesInPharmacy;
    }

    if (sortName != null && !sortName.Equals("No"))
    {
        if (sortName == "PriceUp")
            medicines.Sort(Medecine.ComparePriceUp);
    }
}

```

```

        else if (sortName == "PriceDown")
            medecines.Sort(Medecine.ComparePriceDown);
        else if (sortName == "DateExpUp")
            medecines.Sort(Medecine.CompareDateExpUp);
        else if (sortName == "DateExpDown")
            medecines.Sort(Medecine.CompareDateExpDown);
    }

    if (page == null)
        page = "0";
    var currentPage = Int32.Parse(page);
    var count = medecines.Count();
    var pageCount = Math.Ceiling((Double)count /
        (Double)m_pageCount);
    var skip = currentPage * m_pageCount;
    var need = skip + m_pageCount;

    ViewBag.Medecines =
        medecines.
            Skip(skip).
                Take(need > count ?
                    m_pageCount - (need - count) :
                    m_pageCount);
    ViewBag.PageCount = pageCount;
    ViewBag.CurrentPage = currentPage;
    return PartialView("Catalog");
}
}
}

```

Листинг Г.4 – OrderConstructorController.cs

```

using BookingMedecines.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace BookingMedecines.Controllers
{
    public class OrderConstructorController : BaseController
    {
        // GET: OrderConstructor
        public ActionResult Index()
        {
            var positions = new List<OrderPosition>();
            var positionPharmacyDict =
                GetPositionPharmacyDict(ref positions);
            var activePositions = positions.Where(p =>
                p.Active && !p.InOrder).ToList();
            ViewBag.ActivePositions = activePositions;
            ViewBag.Sum = activePositions.Sum(p =>

```

```

        p.PositionPrice);
    ViewBag.Positions = positionPharmacyDict;
    return View();
}

public Dictionary<Int32, PositionPharmacy>
    GetPositionPharmacyDict(
        ref List<OrderPosition> positions)
{
    Cart cart = new Cart(GetUser()?.Id);
    if (cart.Enabled)
        positions = cart.GetClientCart();
    else
        positions = new List<OrderPosition>();

    var positionPharmacyDict =
        new Dictionary<Int32, PositionPharmacy>();

    foreach (var position in positions)
    {
        PositionPharmacy posPharmacy;
        if (!positionPharmacyDict.
            ContainsKey(position.PharmacyId))
        {
            posPharmacy = new PositionPharmacy();
            posPharmacy.Pharmacy =
                m_context.Pharmacies.FirstOrDefault(p =>
                    p.PharmacyId ==
                        position.PharmacyId);

            positionPharmacyDict.
                Add(position.PharmacyId,
                    posPharmacy);
        }
        else
        {
            posPharmacy =
                positionPharmacyDict[position.
                    PharmacyId];
        }
        posPharmacy.Positions.Add(position);
    }

    foreach (var positionPharmacy in
        positionPharmacyDict)
    {
        positionPharmacy.Value.ActivePositions =
            positionPharmacy.Value.Positions.
                Where(p => p.Active).ToList();
        positionPharmacy.Value.Sum =
            positionPharmacy.Value.ActivePositions.
                Sum(p => p.PositionPrice);
    }
}

```

```

        foreach (var pos in
            positionPharmacy.Value.Positions)
        {
            var stcks = m_context.Stocks.
                Where(s => s.PharmacyId ==
                    pos.PharmacyId &&
                        s.MedecineId ==
                            pos.MedecineId);
            pos.AllStocksCount =
                stcks.Sum(s => s.StockCount);
        }
    }

    return positionPharmacyDict;
}

[HttpPost]
public ActionResult Add(string mId,
    string pId, string count)
{
    var Count = Int32.Parse(count);
    var medecineId = Int32.Parse(mId);
    var pharmacyId = Int32.Parse(pId);
    if (Count == 0)
        return RedirectToRoute(new { controller =
            "Home", action = "Index" });
    if (GetUser() == null)
        return RedirectToRoute(new { controller =
            "Authorization", action = "Login" });
    Cart cart = new Cart(GetUser()?.Id);
    cart.Add(Count, medecineId, pharmacyId);
    return Json(new { medId = medecineId,
        JsonRequestBehavior.AllowGet });
}

[HttpPost]
public ActionResult DeletePosition(string mId,
    string pId)
{
    var medecineId = Int32.Parse(mId);
    var pharmacyId = Int32.Parse(pId);
    if (GetUser() == null)
        return RedirectToRoute(new { controller =
            "Authorization", action = "Login" });
    Cart cart = new Cart(GetUser()?.Id);
    cart.Remove(medecineId, pharmacyId);
    return Json(new { medId = medecineId,
        JsonRequestBehavior.AllowGet });
}

[HttpPost]
public ActionResult ChangePosition(string mId,

```

```

        string pId, string count_)
    {
        Cart cart = new Cart(GetUser()?.Id);
        var medecineId = Int32.Parse(mId);
        var count = Int32.Parse(count_);
        var pharmacyId = Int32.Parse(pId);
        cart.ChangeCount(medecineId, pharmacyId, count);
        return Json(new { medId = medecineId,
                        pharmId = pharmacyId,
                        JsonRequestBehavior.AllowGet });
    }

    [HttpPost]
    public ActionResult MakeActive(string medecineId,
        string pharmacyId)
    {
        Cart cart = new Cart(GetUser()?.Id);
        cart.MakeActive(Int32.Parse(medecineId),
            Int32.Parse(pharmacyId));
        return Json(new { medId = medecineId, pharmId =
            pharmacyId,
            JsonRequestBehavior.AllowGet });
    }

    public List<OrderPosition> Positions
    {
        get
        {
            Cart cart = new Cart(GetUser()?.Id);
            List<OrderPosition> positions;
            if (cart.Enabled)
                positions = cart.GetClientCart();
            else
                positions = new List<OrderPosition>();
            return positions;
        }
    }

    public ActionResult ActivePositions(string pharmacyId)
    {
        var positions = Positions.
            Where(p => p.PharmacyId ==
                Int32.Parse(pharmacyId));
        ViewBag.Positions = positions;
        var activePositions =
            positions.Where(p => p.Active &&
                !p.InOrder).
                ToList();
        ViewBag.ActivePositions = activePositions;
        ViewBag.Sum =
            activePositions.Sum(p => p.PositionPrice);
        return PartialView("ActiveOrderButton");
    }

```

```

    }

    public ActionResult ShowPositions(string pharmacyId)
    {
        var positions = new List<OrderPosition>();
        var positionPharmacyDict =
            GetPositionPharmacyDict(ref positions);
        var activePositions = positions.Where(p =>
            p.Active && !p.InOrder).ToList();
        ViewBag.ActivePositions = activePositions;
        ViewBag.Sum = activePositions.Sum(p =>
            p.PositionPrice);
        ViewBag.Positions = positionPharmacyDict;
        return PartialView("Positions");
    }

    public JsonResult PharmacyMap(string pharmacyId)
    {
        return Json(GetPharmacyWithId(pharmacyId),
            JsonRequestBehavior.AllowGet);
    }

    [HttpGet]
    public ActionResult Final()
    {
        var positions = Positions;
        var activePositions = positions.
            Where(p => p.Active && !p.InOrder).ToList();
        var pharmacies = activePositions.
            Select(p => p.Pharmacy).Distinct();
        ViewBag.Ready = pharmacies.Count() == 1;
        ViewBag.Pharmacies = pharmacies;
        ViewBag.ActivePositions = activePositions;
        ViewBag.Sum =
            activePositions.Sum(p => p.PositionPrice);
        ViewBag.PharmacyId =
            new SelectList(pharmacies,
                "PharmacyId", "PlaceName");

        var clients = GetUser().Clients;
        ViewBag.Profiles = clients;
        ViewBag.CurrentProfile =
            clients.Count == 0 ? Client.Create() :
                clients.First();

        return View();
    }

    public PartialViewResult ProfileInfo(string clientName)
    {
        var positions = Positions;
    }

```

```

var activePositions = positions.
    Where(p => p.Active && !p.InOrder);
var pharmacies = activePositions.
    Select(p => p.Pharmacy).Distinct();
ViewBag.Ready = pharmacies.Count() == 1;
ViewBag.Pharmacies = pharmacies;
ViewBag.ActivePositions = activePositions;
ViewBag.Sum = activePositions.
    Sum(p => p.PositionPrice);
ViewBag.PharmacyId = new SelectList(pharmacies,
    "PharmacyId", "PlaceName");

var clients = GetUser().Clients;
ViewBag.Profiles = clients;
ViewBag.CurrentProfile = clients.
    FirstOrDefault(c => c.FullName ==
        clientName);

return PartialView("ProfileInfo");
}

private object m_locker = new object();

[HttpPost]
public ActionResult Final([Bind(Include = "Client,
    PharmacyId, IssueDate, IssueTime")]
    OrderOptions options)
{
    lock (m_locker)
    {
        var cart = new Cart(GetUser()?.Id);
        var order = new Order();
        order.Client = options.Client;
        order.Completed = false;
        order.IssueDate = options.IssueDate;
        order.IssueDate = new DateTime(
            options.IssueDate.Year,
            options.IssueDate.Month,
            options.IssueDate.Day,
            options.IssueTime.Hour,
            options.IssueTime.Minute,
            options.IssueTime.Second);
        order.PharmacyId = options.PharmacyId;
        order.Positions = new List<OrderPosition>();

        var positions = Positions;
        var activePositions = positions.
            Where(p => p.Active &&
                p.PharmacyId == options.PharmacyId);

        order.FullPrice =
            activePositions.Sum(p =>
                p.PositionPrice);
    }
}

```

```

order.Employee =
    m_context.Employees.FirstOrDefault();
options.Client.AppUser = GetUser();
order.OrderDate = DateTime.Now;

bool errorCount = false;
foreach (var position in activePositions)
{
    var needCount = position.PositionCount;
    while (needCount > 0)
    {
        var stock =
            m_context.Stocks.
                FirstOrDefault(s =>
                    s.PharmacyId ==
                    options.PharmacyId &&
                    s.MedecineId ==
                    position.MedecineId &&
                    s.StockCount > 0);
        if (stock == null ||
            stock.StockCount <= 0)
            break;

        var selected =
            stock.StockCount < needCount ?
            stock.StockCount : needCount;
        stock.StockCount -= selected;
        needCount -= selected;

        var stockPosition = new
            StockPosition(order,
                stock,
                selected);

        m_context.StockPositions.
            Add(stockPosition);
    }

    if (needCount > 0)
    {
        errorCount = true;
        break;
    }
}

if (errorCount)
{
    return View("ErrorOrder");
}

m_context.Orders.Add(order);
m_context.SaveChanges();
var orderId = order.OrderId;

```

```

        foreach (var activePosition in
            activePositions)
        {
            cart.AddToOrder(activePosition.
                Medecine.MedecineId, orderId);
        }
    }

    return RedirectToRoute(new { controller = "Home",
        action = "MyOrders" });
}

private List<string> m_times = new List<string>();

[HttpPost]
public ActionResult CheckDate(DateTime date)
{
    var upBorder = CreateDateTimeMinsUp(date);
    var downBorder = CreateDateTimeMinsDown(date);
    var order = m_context.Orders.FirstOrDefault(o =>
        o.IssueDate > downBorder &&
        o.IssueDate < upBorder);
    string result;
    if (order != null)
    {
        result = "error";
        var downOrder = m_context.Orders.
            FirstOrDefault(o =>
                o.IssueDate >= downBorder &&
                o.IssueDate <= date);
        if (downOrder != null)
            FindDownTime(downOrder.IssueDate);
        else
        {
            var firstInRightHalf =
                m_context.Orders.
                    FirstOrDefault(o =>
                        o.IssueDate >= date &&
                        o.IssueDate <= upBorder);
            var a =
                CreateDateTimeMinsDown(
                    firstInRightHalf.IssueDate);
            m_times.Add(a.ToString());
        }

        var upRange = m_context.Orders.
            Where(o => o.IssueDate >= date &&
                o.IssueDate <= upBorder).
                ToList();
        upRange = upRange.OrderBy(o =>

```

```

        o.IssueDate).
            ToList();
var upOrder = upRange.Count != 0 ?
    upRange.Last() :
    null;
if (upOrder != null)
    FindUpTime(upOrder.IssueDate);
else
{
    var leftHalf = m_context.Orders.
        Where(o => o.IssueDate >= downBorder &&
            o.IssueDate <= date).
        ToList();
    leftHalf = leftHalf.OrderBy(o =>
        o.IssueDate).
        ToList();
    var lastInLeftHalf =
        upRange.Count != 0 ?
        leftHalf.Last() :
        null;

    m_times.Add(CreateDateTimeMinsUp(
        lastInLeftHalf.IssueDate).
        ToString());
}
}
else
{
    result = "success";
}

return Json(new { Result = result,
    Times = m_times, JsonRequestBehavior.AllowGet });
}

private void FindUpTime(DateTime time)
{
    var upTime = CreateDateTimeMinsUp(time);
    var upOrder = m_context.Orders.
        FirstOrDefault(o => o.IssueDate > time &&
            o.IssueDate <= upTime);
    if (upOrder != null)
        FindUpTime(upOrder.IssueDate);
    else
        m_times.Add(upTime.ToString());
}

private void FindDownTime(DateTime time)
{
    var downTime = CreateDateTimeMinsDown(time);
    var downOrder = m_context.Orders.

```

```

        FirstOrDefault(o => o.IssueDate >= downTime
            && o.IssueDate < time);
    if (downOrder != null)
        FindDownTime(downOrder.IssueDate);
    else
        m_times.Add(downTime.ToString());
}

private DateTime CreateDateTimeMinsUp(DateTime date)
    => date.AddMinutes(5);

private DateTime CreateDateTimeMinsDown(DateTime date)
    => date.AddMinutes(-5);
}
}

```

Приложение Д
ИСХОДНЫЙ КОД МЕТОДА ПОИСКА ПОЛЬЗОВАТЕЛЯ
(рекомендуемое)

Листинг Д.1 – Функция поиска пользователя в базе данных

```

public PartialViewResult SearchUser(UserSearchParameters
parameters)
{
    var userListResult = m_context.Users.Include(u =>
        u.Clients).ToList();
        foreach (var user in userListResult)
            user.RoleNames = UserManager.GetRoles(user.Id);

    if (parameters.UserName != null)
        userListResult = userListResult.Where(u => u.UserName
            == parameters.UserName).ToList();
    if (parameters.Phone != null)
        userListResult = userListResult.Where(u => u.PhoneNumber
            == parameters.Phone).ToList();

    AppUser[] userListArray;
    if (parameters.CName != null)
    {
        userListArray = userListResult.ToArray();
        for (int i = 0; i < userListResult.Count(); i++)
        {
            var user = userListArray[i];
            var clients = user.Clients;
            bool result = false;
            foreach (var client in clients)
            {
                if (client.Name.Contains(parameters.CName))
                {
                    result = true;
                    break;
                }
            }

            if (!result)
                userListResult.Remove(user);
        }
    }

    if (parameters.LastName != null)
    {
        userListArray = userListResult.ToArray();
        for (int j = 0; j < userListResult.Count(); j++)
        {
            var user = userListArray[j];
            var clients = user.Clients;
            bool result = false;
            foreach (var client in clients)
            {
                if (client.LastName.
                    Contains(parameters.LastName))
                {

```

```
        result = true;
        break;
    }
}

    if (!result)
        userListResult.Remove(user);
}

if (parameters.Phone != null)
    userListResult = userListResult.Where(u =>
        u.PhoneNumber == parameters.Phone).ToList();
ViewBag.Users = userListResult;
return PartialView("ClientList");
}
```

Приложение Е
ГРАФ СВЯЗНОСТИ ДЛЯ МЕТОДА ПОИСКА ПОЛЬЗОВАТЕЛЯ
(рекомендуемое)

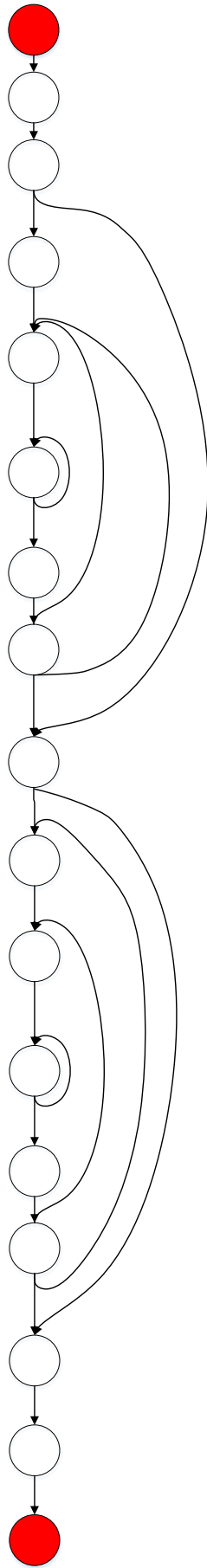


Рисунок Е.1 – Граф связности

Приложение Ж
СЕТЕВОЙ ГРАФИК ПРОЕКТА
(рекомендуемое)

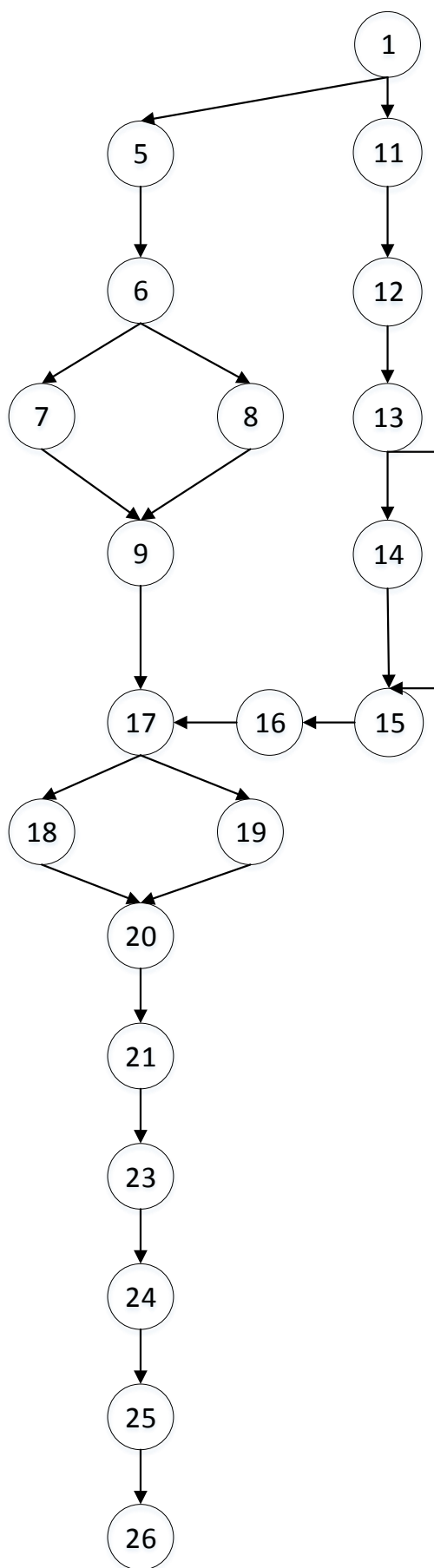


Рисунок Ж.1 – Сетевой график проекта

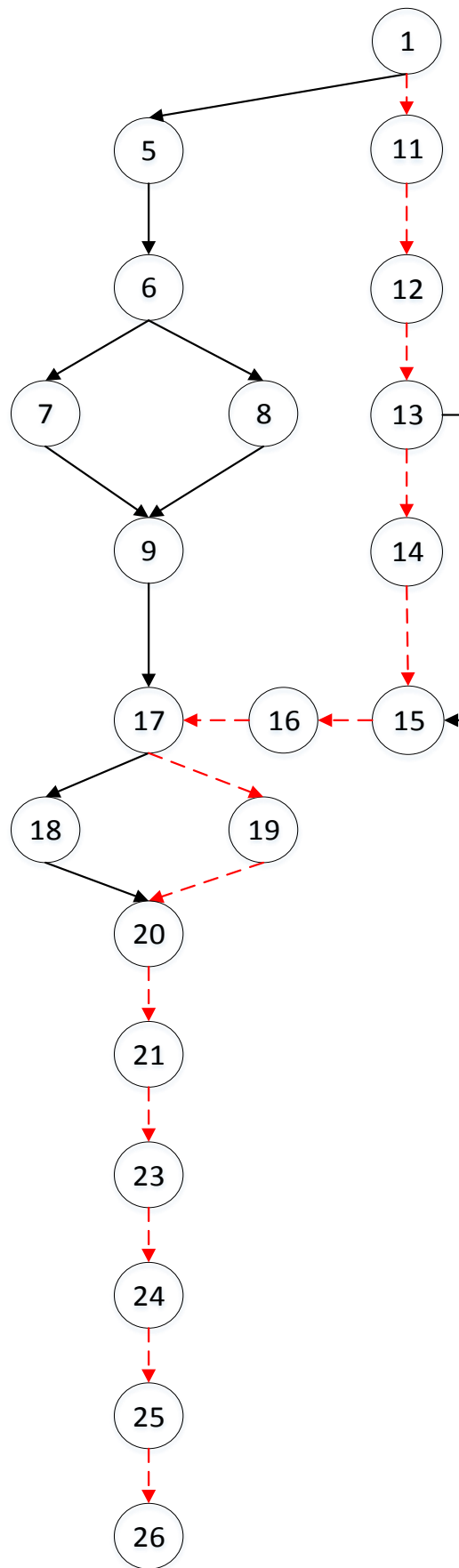


Рисунок Ж.2 – Критический путь проекта

Приложение И
ГРАФИЧЕСКАЯ ЧАСТЬ
(рекомендуемое)

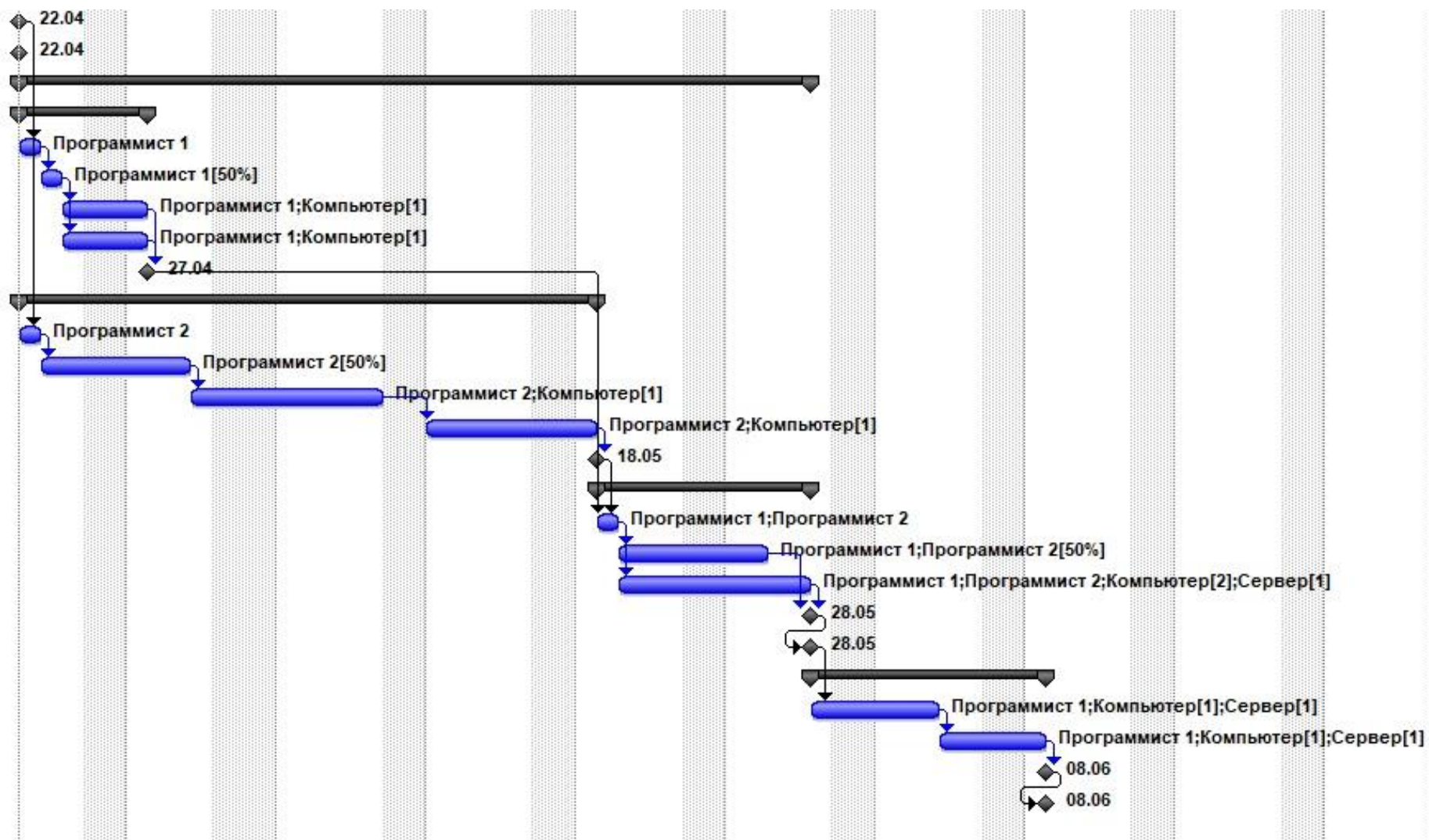


Рисунок И.1 – Диаграмма Ганта по проекту

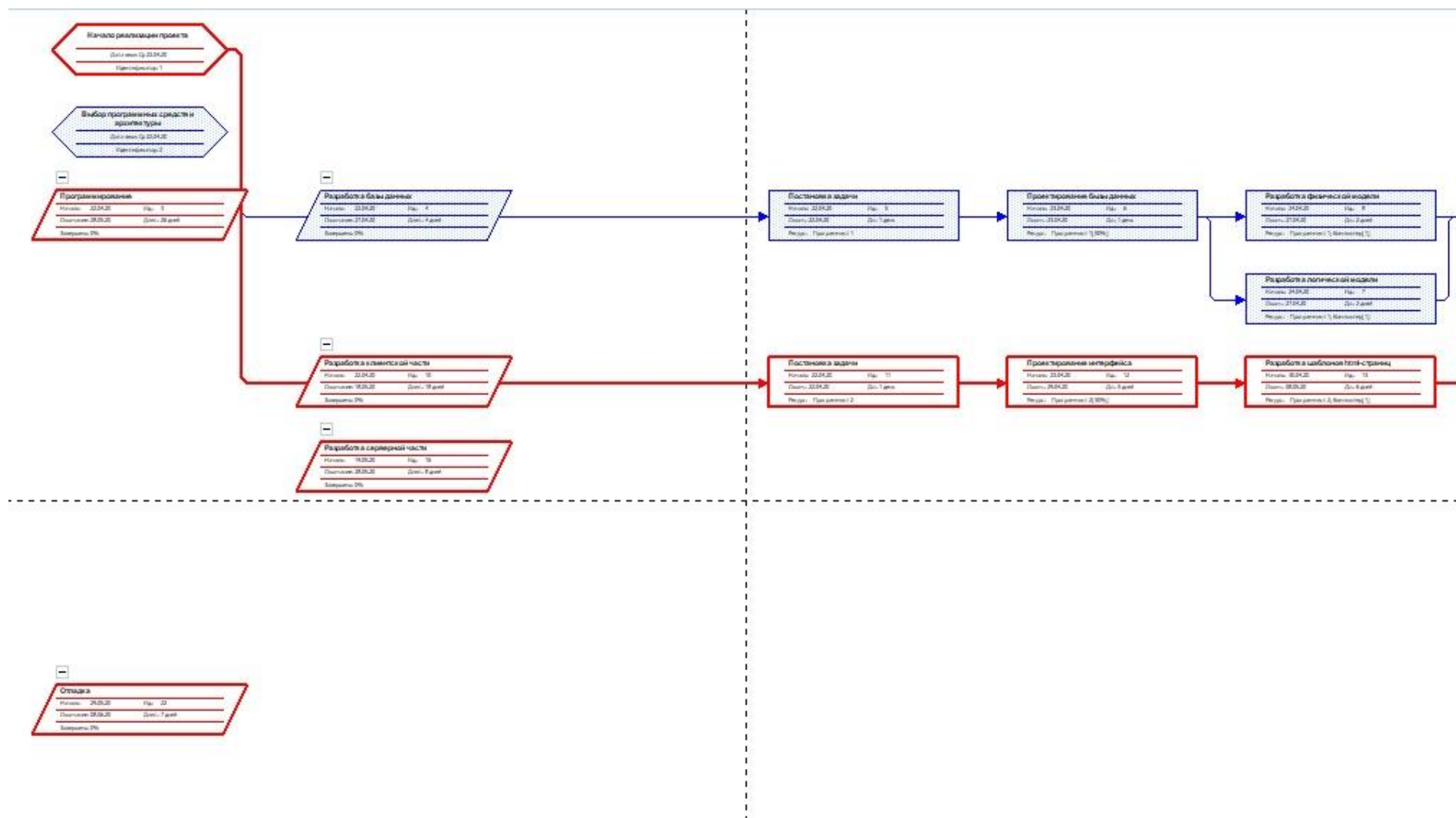


Рисунок И.2 – Сетевой график (начало)

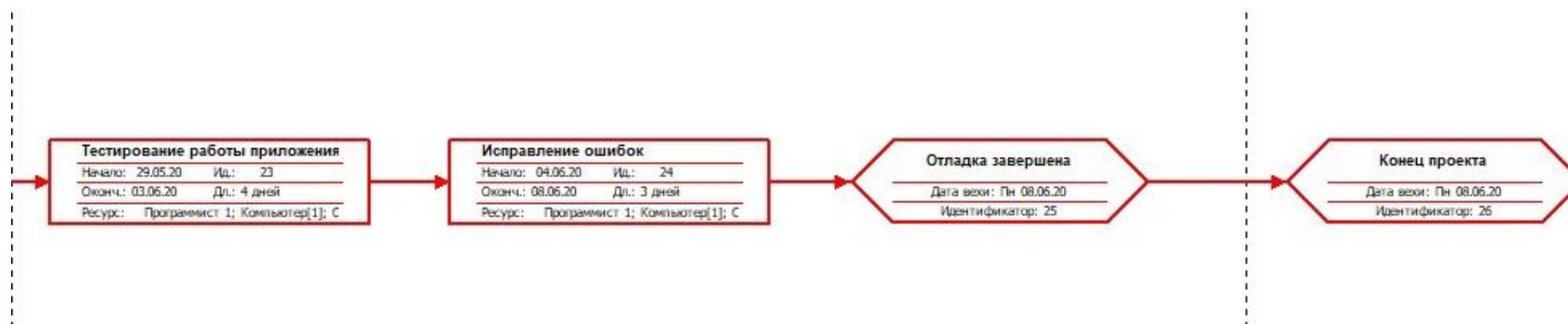


Рисунок И.4 – Сетевой график (окончание)

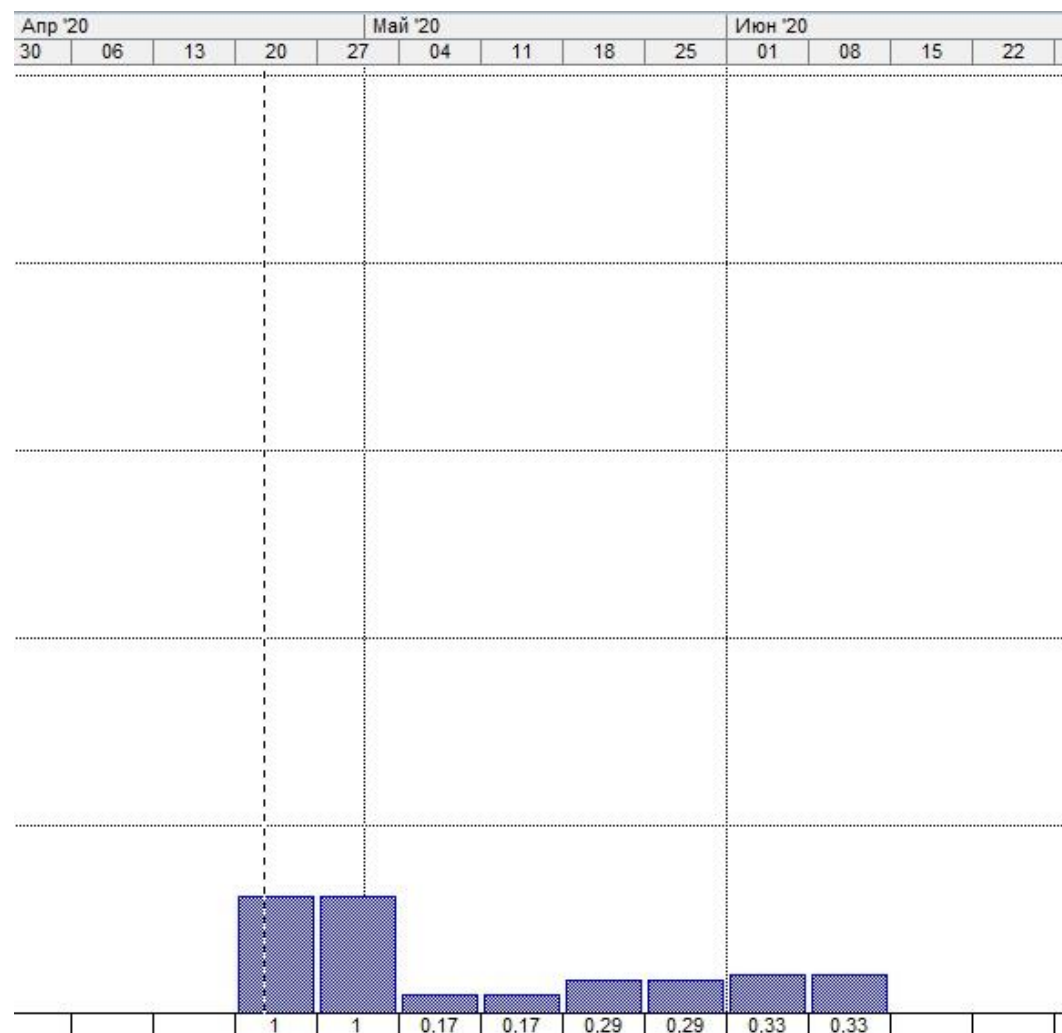


Рисунок И.5 – График ресурсов для ресурса «Компьютер»

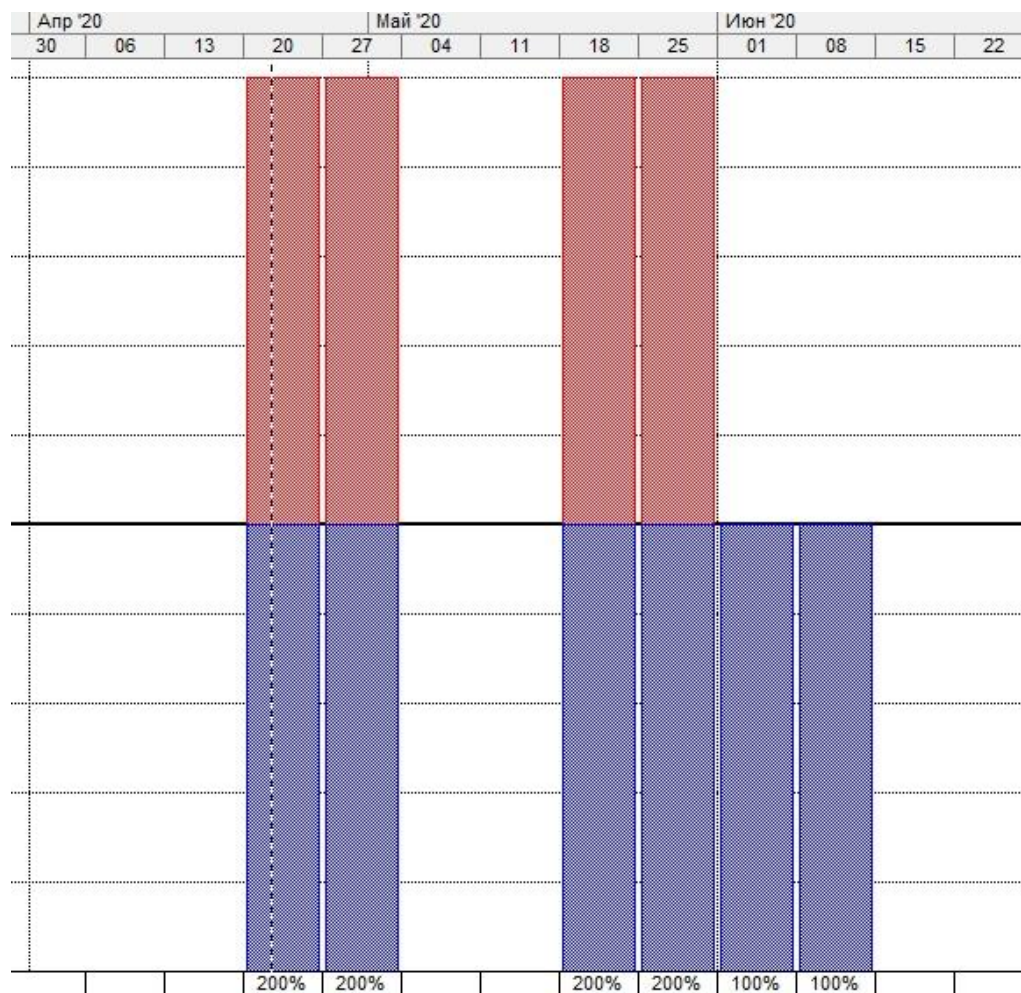


Рисунок И.7 – График ресурсов для ресурса «Программист1»

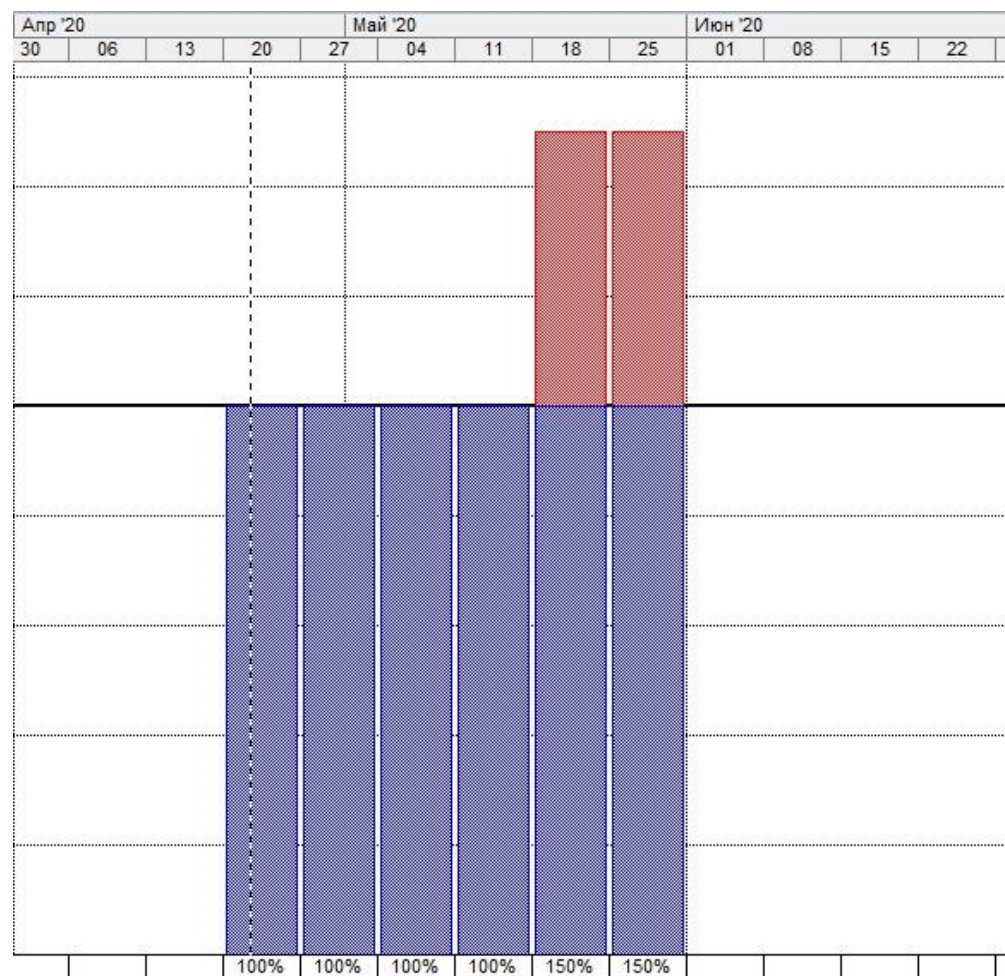


Рисунок И.8 – График ресурсов для ресурса «Программист2»