

## **ВВЕДЕНИЕ**

Во множестве бытовых вопросов передовым помощником впредь является интернет. Данная причина является ключом создания множества программ для упрощения деятельности пользователей компьютеров в любых сферах жизни. Походы в развлекательные центры стали упрощены благодаря бронированию мест через интернет в различные заведения. Из этого следует актуальность выбранной темы.

Цель проекта учебной практики – разработка web-сайта для бронирования билетов в сфере развлечений, который предназначен для самых различных пользователей и администраторов в сфере услуг.

Темой учебной практики по разработке и сопровождению ПО является «Бронирование мест в сфере развлечений». На основании описания предметной области необходимо спроектировать архитектуру программы и разработать готовый продукт для бронирования.

Пояснительная записка состоит из восьми разделов, содержащих необходимую информацию по организации и эксплуатации программного средства.

Первый раздел «Проектирование задачи с позиций методологий IDEF0, IDEF1, IDEF1X» содержит описание указанных моделей, представление в графическом виде, используемого для разработки объектно-ориентированных систем.

Второй раздел «Создание информационной базы проекта. Моделирование данных» описывает базу данных как информационную модель с предлагающимися структурными таблицами, для которой разрабатывается программа.

Третий раздел «Моделирование системы через построение UML-диаграмм» содержит решения по архитектуре и UML-диаграммы.

Четвертый раздел «Программирование базы данных» содержит информацию по реализации базы данных и использованию SQL-команд.

Пятый раздел «Реализация программы» содержит описание назначения программы. Кроме этого, описываются средства защиты и структура справочной системы.

Шестой раздел «Тестирование» содержит описание проверки каждого пункта меню, каждой операции, моделирование всех возможных действий пользователя при работе с программой.

Седьмой раздел «Оценка стоимости разработки программного обеспечения» содержит вычисления по оценочной стоимости программного продукта.

Восьмой раздел «Сопровождение разработанного программного обеспечения» содержит описание по сопровождению.

В заключении подведены итоги о проделанной работе, а также проверено соответствие разработанного программного продукта поставленной задаче.

# **1 ПРОЕКТИРОВАНИЕ ЗАДАЧИ С ПОЗИЦИЙ МЕТОДОЛОГИЙ IDEF0, IDEF1, IDEF1X**

Основным назначением методологии SADT является моделирование предметной области с целью определения требований к разрабатываемой системе или программному средству и с целью их проектирования. Методология SADT может применяться при выполнении ранних работ процесса разработки системы или программного средства.

CASE-технология – это совокупность методологий разработки и сопровождения сложных систем (в том числе программных средств), поддерживаемая комплексом взаимосвязанных средств автоматизации. Основные цели использования CASE-технологий при разработке ПС – отделить анализ и проектирование от программирования и последующих работ процесса разработки, предоставив разработчику соответствующие методологии визуального анализа и проектирования.

Диаграмма DFD основана на методах, ориентированных на потоки данных. Методология является одной из методологий функционального моделирования предметной области. DFD-модель должна иметь единственные цель, точку зрения, субъект и точно определенные границы. DFD-модель отражает перемещение объектов, их хранение, обработку, внешние источники и потребителей данных.

На рисунке Б.1 представлена диаграмма DFD. Хранилищами данных являются «сайт», «сервер базы данных» и «база данных»; функции представляют «бронирование места» и «переформирование мест»; сущности: «пользователь» и «заведение».

Диаграммы являются основными рабочими элементами IDEF0-модели. Диаграммы представляют входные-выходные преобразования и указывают правила и средства этих преобразований.

На рисунке Б.2 представлена диаграмма IDEF0, на которой изображена

функция «забронировать место». К ней относятся механизмы «пользователь» и «сервер базы данных», управления «вид оплаты» и «свободное место», входная информация – «зарегистрированный пользователь» и выходная информация – «забронированное место».

На рисунке Б.3 отображена диаграмма декомпозиции IDEF0. Функции описаны ниже в порядке следования:

- 1) «вход пользователя», где входная информация – «логин» и «пароль», управление – «данные базы данных», выходная информация – «вошедший пользователь», вызов внешней функции – «регистрация»;
- 2) «бронирование билета», где входная информация – «вошедший пользователь», управление – «список свободных мест», выходная информация – «забронированный билет»;
- 3) «отметка брони», где входная информация – «забронированный билет» и «неоплаченная бронь», управление – «оплата», выходная информация – «отмеченное бронирование»;
- 4) «оплата бронирования», где входная информация – «отмеченное бронирование», управление – «безналичный расчет», выходная информация – «неоплаченная бронь» и «оплаченная бронь»;
- 5) «формирование мест», где входная информация – «оплаченная бронь», управление – «занятые места» и выходная информация – «перестроенные места»;
- 6) «выборка данных статистики и финансовый отчет», где входная информация – «перестроенные места», управление – «критерии выборки», выходная информация – «статистика посещений» и «финансовый отчет».

Механизмом для функции «выборка данных статистики и финансовый отчет» является «база данных», для всех остальных – «сервер базы данных». Также у функций «вход пользователя» и «бронирование билета» добавляется механизм «Пользователь».

Диаграмма IDEF1 – методология информационного моделирования производственной среды или системы; отображает структуру и содержание информационных потоков, необходимых для поддержки системы; основана на реляционной теории Кодда и использовании ER-диаграмм.

На рисунке Б.4 представлена диаграмма IDEF1. В неё входят сущности:

- 1) тип заведения, где «тип заведения» – атрибут;
- 2) заведение, где «название заведения» и «адрес заведения» – атрибут;
- 3) мероприятие, где «название мероприятия» и «дата мероприятия» – атрибут;
- 4) билет, где «номер места в билете» и «стоимость билета» – атрибут;
- 5) тип билета, где «тип билета» – атрибут
- 6) статус билета, где «статус билета» – атрибут;
- 7) платеж, где «дата платежа» и «количество билетов в платеже» – атрибут;
- 8) вид платежа, где «вид платежа» – атрибут;
- 9) пользователь, где «логин пользователя» и «пароль пользователя» – атрибут.

В верхней части таблиц, где первое слово «ID», указание первичного ключа соответствующей сущности. Связи, в роли ассоциаций, обозначены комментарием.

Диаграмма IDEF1X – методика информационного моделирования (моделирования данных). Цель его состоит в обеспечении разработчика системы концептуальной схемой базы данных в форме модели сущность – связь, предметной области, которая может быть отображена в любую систему базы данных.

На рисунке Б.5 представлена диаграмма IDEF1X. Представляет аналогию с IDEF1 по сущностям и их атрибутам, комментарием правее или под связью. В верхней части сущностей с началом «ID» и окончанием «(FK)» указаны внешние ключи. Левее или над связью указан порядковый номер после символов «R/», далее комментарий отношения к таблице, содержащимся внешним ключом. Над

графическими компонентами указано название сущности и через символ «/» порядковый номер. Связь со знаком «P» указывает на отношение «1 и более», при «(>= 2)» – отношение «2 и более», при «(>= 4)» – отношение «4 и более», при «3» – отношение к 3. Связь без дополнительного символа означает отношение «0 или 1 или много».

В итоге, были построены диаграммы по методологии SADT, отображающие потоки данных, функциональные преобразования и модель концептуальной схемы базы данных.

## **2 СОЗДАНИЕ ИНФОРМАЦИОННОЙ БАЗЫ ПРОЕКТА. МОДЕЛИРОВАНИЕ ДАННЫХ**

### **2.1 Описание предметной области**

Пользователь (user): Лицо или группа лиц, извлекающих пользу из системы в процессе ее применения [1].

Онлайн-бронирование — бронирование через Интернет, в интерактивном режиме. Термин применяется по отношению к бронированию номеров в гостиницах, билетов (авиа, железнодорожных, автобусных и т.п), мест в ресторанах и театрах, прокату автомобилей и т. д.

Общие принципы онлайн-бронирования:

- 1) поиск доступных предложений;
- 2) заполнение формы с контактной и платежной информацией;
- 3) осуществление платежа;
- 4) получение документа, подтверждающего бронирование.

По способу размещения заявки можно выделить два типа систем бронирования билетов:

- 1) истинное онлайн-бронирование — предполагает выбор мест на графическом плане зрительного зала. На плане отображаются доступные для выбранного мероприятия места, план размечается в соответствии со схемой расписки, отображаются цены на билеты. Системы, обеспечивающие истинное онлайн-бронирование нередко предоставляют зрителям дополнительные возможности, такие как онлайн-оплата заказанных билетов с помощью кредитных карт или электронных платежных систем, печать билетов («электронный билет»), выбор дополнительных услуг (доставка билетов, рекомендация аналогичных мероприятий) и пр. Основным преимуществом систем бронирования этого типа является наглядность: зритель видит, какие места доступны для

приобретения, в какой части зала находятся выбранные им места, сколько стоят те или иные билеты. К недостаткам следует отнести сравнительно высокую сложность и, соответственно, — стоимость таких систем;

- 2) псевдоонлайн-бронирование по сути является формой предварительной заявки. В такой форме зритель может выбрать мероприятие, на которое он хочет приобрести билет, иногда — выбрать, в какой части зала он предпочел бы занять места, указать количество мест и ввести контактную информацию. Отправленная заявка обрабатывается сотрудником театрально-концертной организации, который, в случае подтверждения заказа, связывается со зрителем для уточнения деталей и фактического размещения брони. Основное преимущество такого рода систем — простота реализации и дешевизна. Главным недостатком следует назвать высокую нагрузку на сотрудников отдела продаж учреждения: процесс оформления брони практически не автоматизирован и по сути аналогичен приему заявок по телефону.

Предметная область представлена следующей информацией: разработать Web-сайт, который позволяет бронировать билеты для мероприятий в заведениях различного типа. Требуется хранить следующую информацию: тип заведения, заведение, мероприятие, билет: статус и тип, платеж и его вид, данные пользователей.

Необходимо реализовать функции программы для незарегистрированных пользователей:

- 1) регистрация пользователя;
- 2) авторизация пользователя;
- 3) просмотр афиши заведений;
- 4) просмотр мест с информацией о доступности брони;
- 5) просмотр стоимости места.

Для зарегистрированных пользователей доступны также:



- 1) вход авторизованного пользователя в личный кабинет;
- 2) редактирование профиля пользователя;
- 3) удаление пользователя;
- 4) просмотр забронированных мест данным пользователем с полной информацией;
- 5) просмотр истории платежей;
- 6) добавление бронирования;
- 7) редактирование бронирования;
- 8) удаление бронирования;
- 9) оплата брони;
- 10) добавление комментариев;
- 11) редактирование комментариев;
- 12) удаление комментариев.

Функции администратора, помимо вышеописанных:

- 1) автоматическое снятие брони;
- 2) добавление в черный список пользователя;
- 3) удаление пользователя из черного списка;
- 4) добавление заведений;
- 5) редактирование заведений;
- 6) удаление заведений;
- 7) добавление данных в афишу заведений;
- 8) редактирование данных афиши заведений;
- 9) удаление данных из афиши заведений.

В качестве готового продукта будет представлен сайт с вышеуказанными возможностями.

## **2.2 Анализ существующих аналогов**

Букинг.ком (1996г.) - нидерландская компания, занимающаяся бронированием гостиниц онлайн. Дочка ГК Priceline Group, насчитывающей

15 000 сотрудников, 204 офисов 70 стран мира.

Booking.com предлагает широкий ассортимент вариантов проживания: апартаментов, гостевых домов, семейных пансионатов, хостелов. Клиентам сайта доступна возможность бронирования авиабилетов онлайн, аренды авто, заказа ресторанного столика, такси до аэропорта. Программа Booking.com распознает 40 языков. Общее количество предложений составляет 1 476 702 номеров, 121 276 направлений 228 уголков мира.

Ежедневно сервис регистрирует более 1,5 млн. номеров онлайн, помогая путешественникам подобрать идеальный вариант отдыха, командировки, познавательного путешествия. Услуги агентства предоставляются бесплатно. Клиенты, отыскавшие более выгодную стоимость, получают возмещение разницы. Служба поддержки сервиса работает круглосуточно.

Отличительные характеристики Booking.com:

- 1) внесение оплаты при заселении;
- 2) возможность отмены бронирования номера за 24 часа до заезда;
- 3) описания отелей мира: месторасположения, предоставляемого сервиса, цен;
- 4) отзывы пользователей;
- 5) удобство интерфейса;
- 6) выбор оптимальных мест проживания - вилл, отелей, хостелов, номеров бизнес-класса;
- 7) фильтрация цен онлайн, отзывов, количество звезд, вариантов питания, размещения, удобств;
- 8) предоставление уникальных предложений, акций, скидок онлайн.
- 9) дополнительные опции (близкое расположение аэропорта, возможность автопроката);
- 10) предоставление выгодных условий турагентам. Программа обеспечивает простоту бронирования номеров клиентов турагентств, зарабатывая комиссию.

Рамблер/касса — онлайн-сервис, где можно купить билеты в кино, театры, на концерты и спортивные мероприятия в Москве, Санкт-Петербурге и ещё 90 городах России;

- 1) удобный сайт и приложение для iOS и Android;
- 2) билеты без комиссии в 400+ кинотеатров;
- 3) оплата банковской картой, со счёта мобильного телефона или через PayPal и WebMoney;
- 4) всегда актуальная информация о сеансах и наличии мест;
- 5) внимательная служба поддержки пользователей, которая поможет в любой ситуации.

Международная компания Ticketpro работает с 1992 года, и на нынешний день уверенно занимает лидирующие позиции на рынке распространения билетов на различные мероприятия в 17 странах мира. Компания Ticketpro.by, работающая в Республике Беларусь, использует технологию и компетенцию своего партнера – Ticketpro International.

Ticketpro International - ведущий оператор продажи билетов, действующий в настоящее время в Болгарии, Венгрии, Греции, Индии, США, Канаде, Малайзии, Польше, Республике Беларусь, Словацкой Республике, Хорватии, Чешской Республике.

Основная цель Ticketpro.by - предоставление клиентам в Республике Беларусь услуг мирового класса и самого высокого качества. Используя программное обеспечение по продаже билетов собственной разработки, мы гарантируем постоянные инновации и улучшения, согласно мировым тенденциям в данном сегменте рынка.

Ticketpro использует лучшее программное обеспечение в мире для организации продаж билетов с использованием самых последних технологий и имеет огромный опыт успешной работы. Поэтому организаторы мероприятий мирового уровня в новых, незнакомых для себя странах (которой для многих из них и является Республика Беларусь), предпочитают сотрудничество с уже зарекомендовавшей себя компанией Ticketpro.

Ежегодно продавая миллионы билетов по всему миру и постоянно руководствуясь принципами эффективности, надежности и достоверности, Ticketpro является выбором номер один для организатора мероприятия в любом месте и в любое время.

Несмотря на большое количество подобных сервисов, создание данного сайта сможет найти реализацию в более узких областях Беларуси для объединения множества малоизвестных заведений и создания в них услуг такого рода.

## 2.3 Модель данных

Информационная модель – модель объекта, представленная в виде информации, описывающей существенные для данного рассмотрения параметры и переменные величины объекта, связи между ними, входы и выходы объекта и позволяющая путём подачи на модель информации об изменениях входных величин моделировать возможные состояния объекта.

База данных – это информационная модель, позволяющая упорядоченно хранить данные о группе объектов, обладающих одинаковым набором свойств.

Наиболее распространенное средство моделирования данных – это диаграммы "сущность-связь" (ERD), представленная на рисунке Б.6, на которой отображены сущности, ключи и связи. Структурные таблицы, представляющие данную диаграмму, описаны в таблице 1, таблице 2, таблице 3, таблице 4, таблице 5, таблице 6, таблице 7, таблице 8 и таблице 9.

Таблица 1 – Структурная таблица «Тип заведения»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_type_institution	integer	+	Код типа заведения
type_institution	varchar(255)		Тип заведения

Таблица 2 – Структурная таблица «Заведение»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_institution	integer	+	Код заведения

*id_type_institution	integer		Код типа заведения
name_institution	varchar(255)		Название заведения
address_institution	varchar(255)		Адрес заведения

Таблица 3 – Структурная таблица «Мероприятие»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_event	integer	+	Код мероприятия
*id_institution	integer		Код заведения
name_institution	varchar(255)		Название заведения
address_institution	varchar(255)		Адрес заведения

Таблица 4 – Структурная таблица «Билет»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_ticket	integer	+	Код билета
*id_event	integer		Код мероприятия
*id_type_ticket	integer		Код типа билета
*id_status_ticket	integer		Код статуса билета
seat_number_ticket	integer		Номер места в билете
cost_ticket	decimal		Стоимость билета

Таблица 5 – Структурная таблица «Тип билета»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_type_ticket	integer	+	Код типа билета
type_ticket	varchar(255)		Тип билета

Таблица 6 – Структурная таблица «Статус билета»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_status_ticket	integer	+	Код статуса билета
status_ticket	varchar(255)		Статус билета

Таблица 7 – Структурная таблица «Вид платежа»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_type_payment	integer	+	Код вида платежа
type_payment	varchar(255)		Вид платежа

Таблица 8 – Структурная таблица «Платеж»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_payment	integer	+	Код платежа
*id_user	integer		Код пользователя
*id_type_payment	integer		Код типа платежа
*id_ticket	integer		Код билета
date_payment	date		Дата платежа
count_ticket	integer		Количество билетов в платеже

Таблица 9 – Структурная таблица «Пользователь»

Имя поля	Тип данных	Ключевое поле	Описание поля
#id_user	integer	+	Код пользователя
login_user	varchar(255)		Логин пользователя
password_user	Varchar(255)		Пароль пользователя

Таким образом, была спроектирована модель базы данных, содержащая девять таблиц. Были определены первичные и вторичные ключи, атрибуты каждой таблицы и их типы данных.

## 2.4 Требования к приложению

Требования к программному обеспечению – совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации.

Интерфейс программного средства должен быть интуитивно понятен для пользователя, а также удобен для просмотра данных. Необходима реализация разграничения прав с делением на категории администрации, зарегистрированных и обычных пользователей.

В качестве защиты программного средства требуется авторизация прав администратора, а также пароль на сервере базы данных в виде хэш-значения.

### 3 МОДЕЛИРОВАНИЕ СИСТЕМЫ ЧЕРЕЗ ПОСТРОЕНИЕ UML-ДИАГРАММ

#### 3.1 Диаграмма вариантов использования

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе. Важно понимать, что диаграммы вариантов использования не предназначены для отображения проекта и не могут описывать внутреннее устройство системы. Диаграммы вариантов использования предназначены для упрощения взаимодействия с будущими пользователями системы, с клиентами, и особенно пригодятся для определения необходимых характеристик системы. Другими словами, диаграммы вариантов использования говорят о том, что система должна делать, не указывая сами применяемые методы.

Диаграмма претендентов отображена на рисунке Б.7. В таблице 10 представлены и описаны объекты диаграммы.

Таблица 10 – Таблица объектов диаграммы претендентов

Актеры	Варианты использования	Отношения
1. Администратор	1. Редактирование данных мероприятия	1. Отношения ассоциации
	2. Редактирование данных заведения	
	3. Редактирование данных типа заведения	
	4. Редактирование данных билета	

	5. Редактирование данных типа билета	2. Отношения расширения
	6. Добавление типа билета	
	7. Изменение типа билета	
	8. Удаление типа билета	
	9. Изменение данных афиши	
	10. Добавление билета	
	11. Удаление билета	
	12. Добавление типа заведения	
	13. Изменение типа заведения	
	14. Удаление типа заведения	
	15. Добавление мероприятия	
	16. Изменение мероприятия	
	17. Удаление мероприятия	
	18. Удаление заведения	
	19. Изменение заведения	
	20. Добавление заведения	



2. Зарегистрированный пользователь	1. Редактирование платежа	1. Отношения ассоциации
	2. Бронирование билета	
	3. Изменение билета	2. Отношения расширения и включения
	4. Удаление платежа	3. Отношения включения
	5. Изменение платежа	
	6. Добавление платежа	
3. Незарегистрированный пользователь	1. Регистрация	1. Отношение ассоциации
	2. Авторизация	
	3. Просмотр данных	
	4. Добавление пользователя	2. Отношения включения

Таким образом, вариант использования описывает, с точки зрения действующего лица, группу действий в системе, которые приводят к конкретному результату. Варианты использования являются описаниями типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать.

### 3.2 Диаграмма классов

Диаграмма классов – это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании - описании существующих и используемых систем. Информация с диаграммы

классов напрямую отображается в исходный код приложения - в большинстве существующих инструментов UML-моделирования возможна кодогенерация для определенного языка программирования. Таким образом, диаграмма классов – конечный результат проектирования и отправная точка процесса разработки.

Диаграмма классов отображена на рисунке Б.8. Она включает в себя классы и атрибуты, аналогичные сущностям и атрибутам, IDEF1-модели, изображенной на рисунке Б.4. Каждый класс, за исключением «Type payment», содержит операции добавления, удаления и редактирования. Классы «Institution», «Event», «Ticket» и «Payment» также включают поиск и сортировку.

Класс «User» находится в связи «зависимость» по отношению к «Payment», также как «Payment» к «Type payment». Отношения «агрегации» прослеживаются от «Ticket» к «Payment» и от «Institution» к «Event». Оставшиеся классы связаны отношением «композиция»: «Type institution» к «Institution»; «Event», «Type ticket» и «Status ticket» к «Ticket».

Полное списочное представление представлено ниже:

- 1) type institution (атрибут «type\_institution»; операции: «добавление», «редактирование», «удаление»; связь композиции к классу institution);
- 2) institution (атрибуты: «name\_institution» и «address\_institution»; операции: «добавление», «редактирование», «удаление», «поиск», «сортировка»; связь агрегации к классу event);
- 3) ticket (атрибуты: «set\_number\_ticket» и «cost\_ticket»; операции: «добавление», «редактирование», «удаление», «поиск», «сортировка»; связь агрегации к классу payment);
- 4) type ticket (атрибут «type\_ticket»; операции: «добавление», «редактирование», «удаление»; связь композиции к классу ticket);
- 5) status ticket (атрибут «status\_ticket»; связь композиции к классу ticket);
- 6) payment (атрибуты: «date\_payment» и «count\_ticket»; операции: «добавление», «редактирование», «удаление», «поиск»,

«сортировка»; связь типа «зависимость» к классу type payment);

- 7) type payment (атрибут «type\_payment»; операции: «добавление», «редактирование», «удаление»);
- 8) user (атрибуты: «login\_user» и «password\_user»; операции: «добавление», «редактирование», «удаление»; связь типа «зависимость» к классу payment).

Классы используются в процессе анализа предметной области для составления словаря предметной области разрабатываемой системы. Это могут быть как абстрактные понятия предметной области, так и классы, на которые опирается разработка и которые описывают программные или аппаратные сущности.

### **3.3 Диаграмма состояния**

Объекты характеризуются поведением и состоянием, в котором находятся. Диаграммы состояний применяются для того, чтобы объяснить, каким образом работают сложные объекты. Состояние - ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условию, выполняет определенную деятельность или ожидает какого-то события. Состояние объекта определяется значениями некоторых его атрибутов и присутствием или отсутствием связей с другими объектами.

Таким образом, диаграмма состояний показывает, как объект переходит из одного состояния в другое, что видно из рисунка Б.9. Сервер исполняет «вход» и «выход» пользователя с сайта, далее переходит к блоку подсостояния бронирования билетов. В него входят следующие состояния: «ожидание ввода логина и пароля», «проверка на соответствие», «ожидание выбора заведения», «ожидание выбора места», «обработка запроса на бронирование», «ожидание платежа», блок подсостояния платежа. К нему относятся: «ожидание выбора платежной системы», «ожидание ввода данных», «проверка на соответствие», «подтверждение платежа», «запрет оплаты», «выдача электронного билета»,

«ожидание выбора пользователя», «печать» и «завершение транзакции».

Диаграмма также дополнена односторонними направленными стрелками и пояснительными записками по отношению к ним. Блоки подсостояния и некоторые состояния работают по принципу цикличности, что позволяет оптимизировать работу web-приложения.

### **3.4     Диаграмма деятельности**

Диаграмма деятельности – UML-диаграмма, на которой показаны действия, состояния которых описано на диаграмме состояний. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграмме, изображённая на рисунке Б.10, состоит из следующих действий:

- 1)    выбор заведения;
- 2)    выбор мероприятия;
- 3)    выбор мест/места;
- 4)    выбор платежной системы;
- 5)    ввод данных;
- 6)    подтверждение платежа;
- 7)    сохранение данных в базе данных;
- 8)    выдача электронного билета;
- 9)    распечатанный билет.

Также на ней отображены линии синхронизации, блоки на проверку истинности условий и два конечных узла, завершающих все потоки диаграммы: «оформление платежа не удалось» и «билет заказан». Таким образом, была спроектированная диаграмма деятельности.

### 3.5 Диаграмма последовательности действий

Диаграмма последовательностей относится к диаграммам взаимодействия UML, описывающим поведенческие аспекты системы, но рассматривает взаимодействие объектов во времени. Данная диаграмма отображает временные особенности передачи и приема сообщений объектами.

На диаграмме, изображённой на рисунке Б.11, отображена последовательность действий администратора для добавления данных в базу данных. Актер – администратор – нажимает на кнопку, в следствии чего web-страница обновляется. Предыдущее действие повторяется. Далее администратор заполняет поля, после чего нажимает на кнопку сохранения и результат добавляется в базу данных, после чего происходит обновление сервера базы данных.

Таким образом, была спроектированная диаграмма последовательности действий для данного web-приложения.

### 3.6 Диаграмма кооперации

Диаграмма кооперации – диаграмма, на которой изображаются взаимодействия между частями композитной структуры или ролями кооперации.

Компоненты диаграммы: объекты и линия связи.

Диаграмма, отображённая на рисунке Б.12, включает в себя следующие объекты:

- 1) зарегистрированный пользователь (актер) – «global» плательщик;
- 2) включающие мультиобъекты:
  - a. заведение – «global»;
  - b. мероприятие – «local»;
  - c. билет – «local»;
  - d. платежная система – «global»;
- 3) платеж;

- 4) сервер базы данных – «global»;
- 5) принтер – «local».

Мультиобъекты связаны с соответствующими объектами при помощи композиции. Каждый объект, являющийся для программы переменной, пронумерован в алфавитном порядке в ходе следования.

Действия происходят в следующем порядке: «выбрать заведение» - «отобразить мероприятия заведения», «выбрать мероприятие» - «отобразить билеты мероприятия», «выбрать билет» - «оформить билета», «отобразить платежные системы» - «выбрать платежную систему» - «ввести данные» - «оплатить», «выбрать билет» - «обновить сервер базы данных», «печать».

### **3.7 Диаграмма развертывания**

Диаграмма развертывания (deployment diagram) – диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

На рисунке Б.13, отображена диаграмма развёртывания. Диаграмма, включает в себя следующие устройства:

- 1) «принтер»;
- 2) «рабочая станция»;
- 3) «сервер сайта»;
- 4) «сервер базы данных»;
- 5) «сервер платежной системы».

В том числе узлы: «web browser» и «сеть»; а также пояснительные записки: «протокол», «протокол» HTTP и «device».

Узлы представляются прямоугольными параллелепипедами с артефактами, расположенными в них, изображенными в виде прямоугольников. Узлы могут иметь подузлы, как вложенные прямоугольные параллелепипеды. Один узел диаграммы развертывания может концептуально представлять множество физических узлов, таких как кластер серверов баз данных.

### 3.8 Диаграмма компонентов




Диаграмма компонентов – диаграмма физического уровня, которая служит для представления программных компонентов и зависимостей между ними.

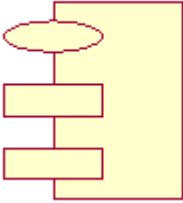
Диаграмма компонентов разрабатывается для следующих целей:

- 1) визуализация общей структуры исходного кода программной системы;
- 2) спецификация исполнимого варианта программной системы;
- 3) обеспечение многократного использования отдельных фрагментов программного кода;
- 4) представление концептуальной и физической схем баз данных.

На изображении Б.14 представлена диаграмма компонентов. Ниже, в табличном представлении, описаны данные компоненты.

Таблица объектов диаграммы компонентов

Графическое изображение и имя	Названия компонентов
NewSubprogBody 	urls.py, settings.py
NewMainSubprog 	views.py, admin.py, models.py
NewTaskBody 	оплата, бронирование

<p>NewPackageSpec</p>  <p>The diagram shows a package named 'NewPackageSpec' represented by a large yellow rectangle. To its left, there are three smaller yellow shapes: an oval at the top and two rectangles below it, all connected to the main package by a vertical line, indicating dependencies or sub-packages.</p>	<p>тип билета, статус билета, мероприятие, заведение, тип заведения, билет, пользователь, платеж, тип платежа</p>
---	---

А также основной компонент «Server page», представляющий собой серверную страницу.

Диаграмма компонентов наглядно демонстрирует необходимые объекты для осуществления web-проекта по бронированию билетов.



## 4 ПРОГРАММИРОВАНИЕ БАЗЫ ДАННЫХ

### 4.1 Организация данных

Организация данных – представление данных и управление данными в соответствии с определенными соглашениями.

Система управления базами данных (СУБД) — специализированная программа (чаще комплекс программ), предназначенная для организации и ведения базы данных. Для создания и управления информационной системой СУБД необходима в той же степени, как для разработки программы на алгоритмическом языке необходим транслятор.

Встраиваемая СУБД – библиотека, которая позволяет унифицированным образом хранить большие объёмы данных на локальной машине. Доступ к данным может происходить через геоинформационные системы.

SQLite – это программная библиотека, которая реализует автономный, безсерверный, нулевой конфигурации транзакционный механизм СУБД SQL. SQLite – наиболее широко распространенный механизм СУБД SQL в мире. Исходный код находится в открытом доступе.

Таблицы предметной области:

- 1) тип заведения;
- 2) заведение;
- 3) мероприятие;
- 4) билет;
- 5) статус билета;
- 6) тип билета;
- 7) платеж;
- 8) тип платежа;
- 9) пользователь.

Структура таблиц представлена в модуле python. Листинг приведен в приложении А: «models.py».

Используемые типы полей:

- 1) id;
- 2) text;
- 3) integer;
- 4) char;
- 5) dateTime;
- 6) foreignKey (внешние ключи).

К выходной информации относятся все значения, кроме таблицы «User»:

- 1) type\_institution;
- 2) name\_institution;
- 3) adress\_institution;
- 4) institution;
- 5) name\_event;
- 6) date\_added;
- 7) type\_ticket;
- 8) status\_ticket;
- 9) seat\_number\_ticket;
- 10) cost\_ticket;
- 11) type\_payment;
- 12) date\_payment;
- 13) count\_ticket.

Входной информацией являются следующие значения:

- 1) login;
- 2) password;
- 3) status\_ticket;
- 4) count\_ticket;
- 5) date\_payment.

Организация данных исходила из построенных диаграмм: ERD, IDEF1 и IDEF1X. Таким образом была создана наиболее надежная и качественная база данных.

## 4.2 Построение физической модели данных

После создания модели, Django автоматически создает API для работы с базой данных, которая позволяет создавать, получать, изменять и удалять объекты. Для представления данных таблицы в виде объектов Python, Django использует интуитивно понятную систему: класс модели представляет таблицу, а экземпляр модели - запись в этой таблице.

При создании моделей они наследуют поведение от класса `django.db.models.Model`, который предоставляет ряд базовых операций с данными.

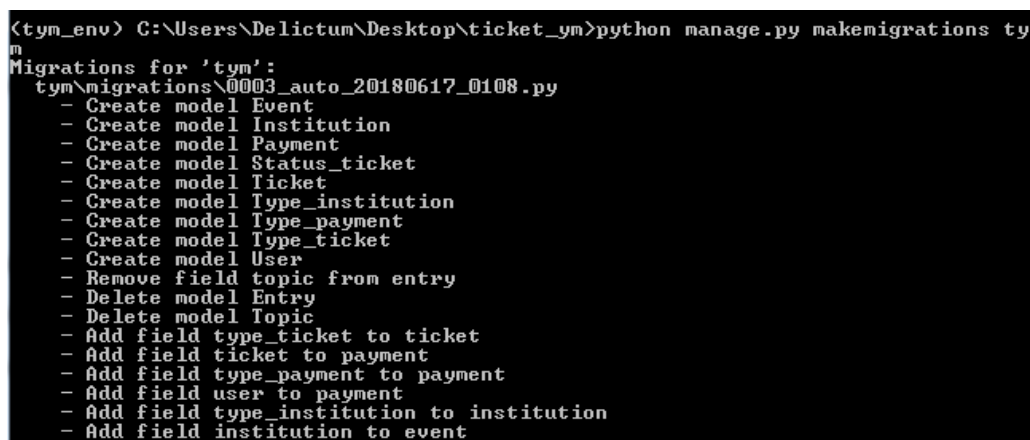
Используя связанные объекты в модели (`ForeignKey`, `OneToOneField` или `ManyToManyField`), в объект модели будет добавлен API для работы со связанными объектами. Если модель содержит `ForeignKey`, объект этой модели может получить связанный объект через атрибут модели.

Подключение физической модели происходит в модуле `settings.py` на строках 80-85

```
DATABASES = { 'default': { 'ENGINE': 'django.db.backends.sqlite3',  
    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), }}
```

А также в строке 43 внутри списка «INSTALLED\_APPS».

Далее происходит осуществление миграции приложения в построенной виртуальной среде, как отображено на рисунке 1.



```
<tym_env> C:\Users\Delictum\Desktop\ticket_yn>python manage.py makemigrations tym  
Migrations for 'tym':  
  tym\migrations\0003_auto_20180617_0108.py  
    - Create model Event  
    - Create model Institution  
    - Create model Payment  
    - Create model Status_ticket  
    - Create model Ticket  
    - Create model Type_institution  
    - Create model Type_payment  
    - Create model Type_ticket  
    - Create model User  
    - Remove field topic from entry  
    - Delete model Entry  
    - Delete model Topic  
    - Add field type_ticket to ticket  
    - Add field ticket to payment  
    - Add field type_payment to payment  
    - Add field user to payment  
    - Add field type_institution to institution  
    - Add field institution to event
```

Рисунок 1 – Миграция приложения

После миграции самого приложения осуществляется дальнейшая миграция с базой данных, как видно из рисунка 2.

```
<tym_env> C:\Users\Delictum\Desktop\ticket_ym>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, tym
Running migrations:
  Applying tym.0003_auto_20180617_0108... OK
<tym_env> C:\Users\Delictum\Desktop\ticket_ym>_
```

Рисунок 2 – Миграция базы данных

Данные действия Django заменяет SQL-запросами и преобразует в полноценную базу данных. Таким образом, представлено программирование базы данных и корректное осуществление информационной модели.

## 5 РЕАЛИЗАЦИЯ ПРОГРАММЫ

### 5.1 Проектирование интерфейса

HTML (HyperText Markup Language) – язык разметки гипертекста. Предназначен для создания Web-страниц. Под гипертекстом в этом случае понимается текст, связанный с другими текстами указателями-ссылками.

Основными компонентами HTML являются:

- 1) тег (tag) – тег HTML это компонент, который командует Web-браузеру выполнить определенную задачу типа создания абзаца или вставки изображения;
- 2) атрибут (или аргумент) – изменяет тег;
- 3) Значение – присваиваются атрибутам и определяют вносимые изменения.

Теги представляют собой зарезервированные последовательности символов, начинающиеся с < (знака меньше) и заканчивающиеся > (знаком больше). Закрытие тега отличается от открытия только наличием символа '/'.

Проектирование интерфейса основано на HTML с использованием модуля «Django-bootstrap3», использующий библиотеки JavaScript. Описание интерфейса web-ссылок изложено в таблице 11.

Таблица 11 – Таблица интерфейса web-ссылок

HTML-документ	Описание
base_generic.html	Базовый документ, являющийся основной для прочих
events.html	Документ отображения мероприятий
index.html	Документ главной страницы
institutions.html	Документ отображения заведений
ticket_booked.html	Документ бронирования билета
ticket_details.html	Документ отображения подробной информации о билете

tickets.html	Документ отображения билетов
type_institutions.html	Документ отображения типов заведений

Файл «base\_generic.html» включает в себя навигационную панель с типом кнопок, как видно из рисунка 3.

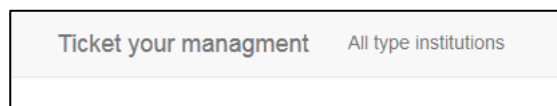


Рисунок 3 – Навигационная панель «base\_generic.html»

В данный файл входят следующие обозначения, отображенные в листинге А.10:

- 1) `<!DOCTYPE html>` – указание типа текущего документа;
- 2) `{% load bootstrap 3 %}` – импорт библиотеки, включающий синтаксис и компоненты JavaScript;
- 3) `<html lang="en">` – указание локализации html;
- 4) `<head>` – контейнер для метаданных;
- 5) `<meta charset="utf-8">` – подключение кодировки UTF-8;
- 6) `<meta http-equiv="X-UA-Compatible" content="IE=edge">` – работоспособность для различных версий InternetExplorer;
- 7) `<meta name="viewport" content="width=device-width, initial-scale=1">` – указывает браузеру, что ширина экрана должна рассматриваться независимо от ширины устройства;
- 8) `{% block title %} <title>` – обозначение наименования вкладки;
- 9) `{% bootstrap_css %}` – подключение css-файла;
- 10) `{% bootstrap_javascript %}` – подключение JavaScript;
- 11) `{% load static %}` – подключение статических файлов;
- 12) `<link rel="stylesheet" href="{% static 'css/style.css' %}">` – ссылка на внешний документ стиля;
- 13) `<body>` – определяет тело документа;
- 14) `<nav class="navbar navbar-default navbar-static-top">` – определение

заголовка навигации в верхней части страницы;

15) `<div class="container">` – задает контейнер для последующих элементов;

16) `<div class="navbar-header">` – определяет заглавную часть в панели навигации;

17) `<button...>` – задает параметры для компонента кнопки;

18) `<... href="{% url 'index' %}">` – задает гиперссылку для кнопки на главную страницу;

19) `<li>` – определяет элемент списка;

20) `<ul>` – устанавливает маркированный список;

21) `<nav>` – задает навигацию по сайту.

Последующие html-файлы также включают в себя вышеописанные теги и атрибуты.

Страница отображения для листинга A.11 представлена на рисунке 4. В нее входят маркированный список с гиперссылками.

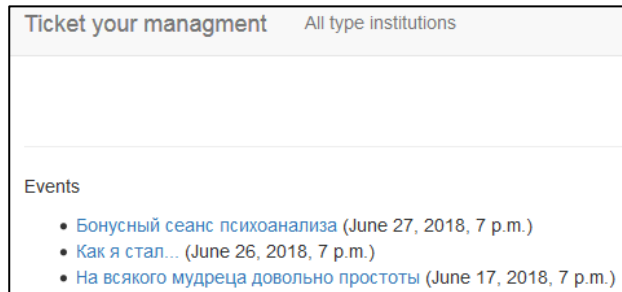


Рисунок 4 – отображение «events.html»

Пользовательское представление web-вкладки для листинга A.13 отображена на рисунке 5. Страница состоит из мероприятий выбранного заведения, оформленных в виде списка с гиперссылками.

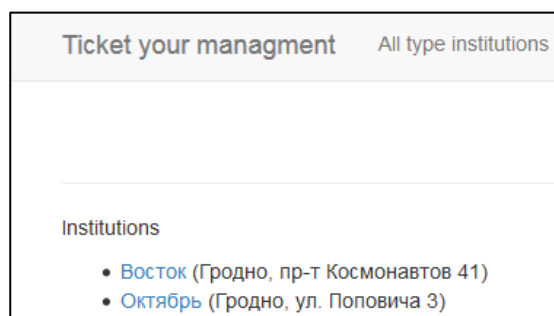


Рисунок 5 – отображение «institutions.html»

Интерфейс основной страницы web-приложения описан в листинге A12. На данной странице размещены заглавие проекта и динамическое содержание, оформленное в виде маркированного списка, как видно из рисунка 6.

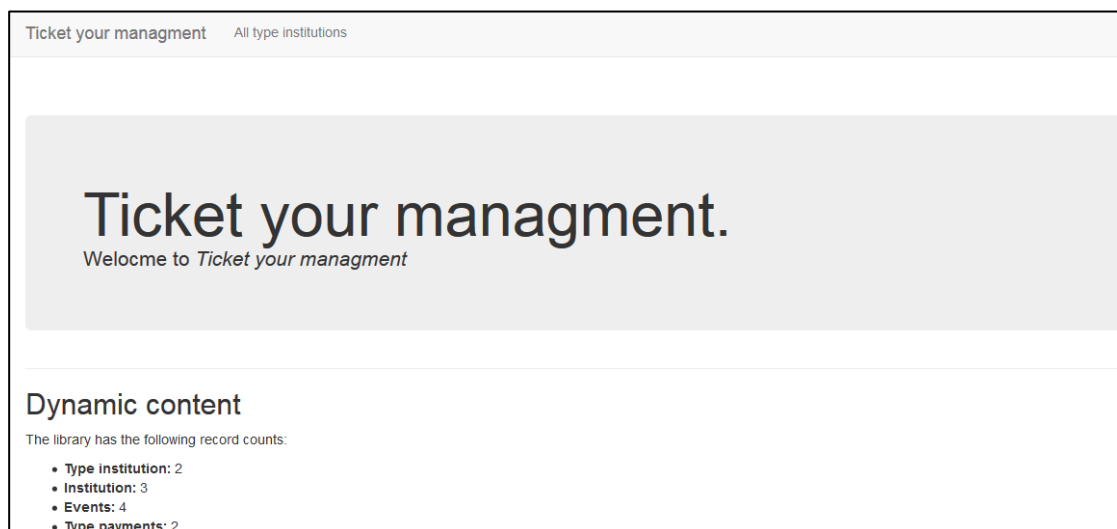


Рисунок 6 – отображение «index.html»

Страница отображения для листинга A.14 представлена на рисунке 7. В ней содержится информация, что пользователь забронировал выбранный билет.

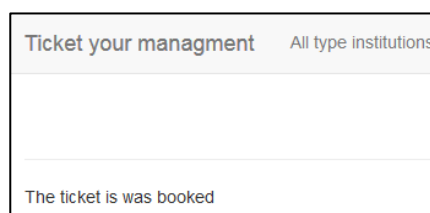


Рисунок 7 – отображение «ticket\_booked.html»

Пользовательское представление вкладки с подробной информацией о билете находится в листинге A.15, отображение странице на рисунке 8.

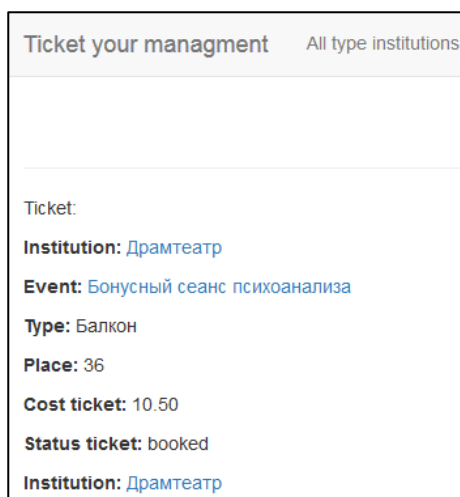


Рисунок 8 – отображение «ticket\_details.html»



Страница отображения для листинга A.16 представлена на рисунке 9. На странице используется список из типов билетов, а также из вложенного списка по билетам.

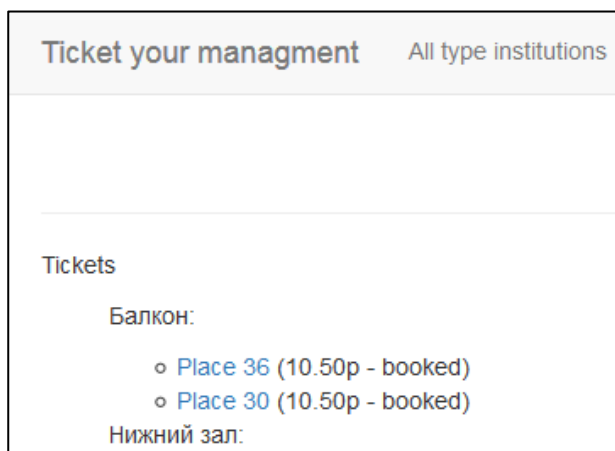


Рисунок 9 – отображение «tickets.html»

Пользовательское представление web-вкладки для листинга A.17 отображена на рисунке 10. Страница состоит из заведений выбранного типа заведения, оформленных в виде списка с гиперссылками.

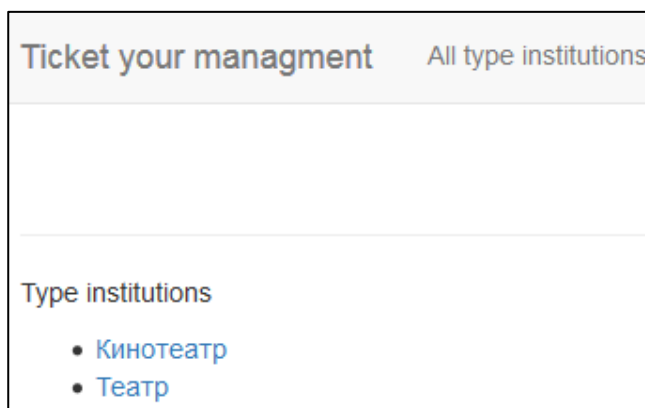


Рисунок 10 – отображение «type\_institutions.html»

В результате на языке HTML с использованными библиотеками и css-файлами был спроектирован интерфейс web-приложения.

## 5.2 Инструменты разработки приложения

Средства разработки программного обеспечения – совокупность приемов, методов, методик, а также набор инструментальных программ, используемых разработчиком для создания программного кода Программы,

отвечающего заданным требованиям.

Python — это простой в освоении, мощный язык программирования. Он имеет эффективные высокоуровневые структуры данных и простой, но эффективный подход к объектно-ориентированному программированию. Элегантный синтаксис и динамическая типизация Python вместе с его интерпретируемой природой делают его идеальным языком для сценариев и быстрой разработки приложений во многих областях на большинстве платформ.

Интерпретатор Python и обширная стандартная библиотека свободно доступны в исходной или двоичной форме для всех основных платформ и могут свободно распространяться. Интерпретатор Python легко дополняется новыми функциями и типами данных, реализованными на C или C ++ (или на других языках, вызываемых с C). Python также подходит как язык расширения для настраиваемых приложений.

Django бесплатный продукт с открытым исходным кодом. Включает в себя десятки дополнительных функций, которые можно использовать для решения общих задач веб-разработки. Решает вопросы об аутентификации пользователей, управлении контентом, карт сайтов, RSS-каналах и многих других задач. Также является абсолютно универсальным и масштабируемым.

Данный выбор является оптимальным, так как разработка и разворачивание web-приложения возможна на различных платформах операционных систем. Помимо этого, продукты представляют собой полностью бесплатный функционал и семантическую простоту разработки. Также не требует больших затрат ресурсов и позволяет осуществить разработку исключительно в консольном окне.

Для разработки использованы следующие библиотеки:

- 1) `dj-database-url` 0.5.0 — реализует поддержку гиперссылок с web-приложение;
- 2) `dj-static` 0.0.6 — обслуживание статических активов сервера WSGI;
- 3) `django` 2.0.6 — основная библиотека разработки приложения;
- 4) `gunicorn` 19.8.1 — HTTP-сервер Python WSGI для UNIX.;

- 5) pytz 2018.4 – позволяет выполнять точные и кросс-платформенные вычисления часовых поясов;
- 6) static3 0.7.0 – включает статический контент в приложения WSGI;
- 7) psycopg2 2.6.1 – адаптер базы данных PostgreSQL для Python;
- 8) django-bootstrap3 – вывод шаблона в код;
- 9) virtualenv – инструмент для создания изолированной среды Python.

Данный перечень библиотек позволяет максимально упростить разработку web-приложения и предоставить возможность развернуть его на сервер.

### **5.3 Функции: физическая и логическая организация**

Проект – это специальная структура управления и группировки файлов, предназначенных для одной среды программирования.

Функции, реализованные в программе, отображены ниже:

- 1) def index(request) – возвращает начальную страницу приложения;
- 2) def institutions(request, id) – возвращает страницу заведений приложения;
- 3) def events(request, id) – возвращает страницу мероприятий заведения приложения;
- 4) def tickets(request, id) – возвращает страницу билетов мероприятия приложения;
- 5) def ticket\_details(request, id) – возвращает страницу детальной информации билета приложения;
- 6) def ticket\_booked(request, id) – возвращает страницу бронирования билета приложения с исполнением соответствующего действия;
- 7) def \_\_str\_\_(self) – возвращает строковое представление класса;
- 8) def \_\_int\_\_(self) – возвращает целочисленное представление класса;
- 9) def \_\_datetime\_\_(self) – возвращает тип даты представления класса.

Данным образом, была осуществлена реализация программы, где

интерфейс представляют web-ссылки. Также указан перечень инструментов разработки и осуществленных функций программного средства.

## 6 ТЕСТИРОВАНИЕ

### 6.1 Функциональное тестирование

При запуске появилась главная страница web-приложения, изображенная на рисунке 5.

В навигационной панели нажали на «All type institutions», после чего произошло перенаправление на страницу, отображенную на рисунке 10. Там же нажали на «Ticket your management», после чего вернулись на домашнюю страницу, указанную на рисунке 5.

Исходя из ссылки [http://localhost:8000/tym/type\\_institutions](http://localhost:8000/tym/type_institutions) выбрали «Театр», после чего отобразилась страница как на рисунке 11.

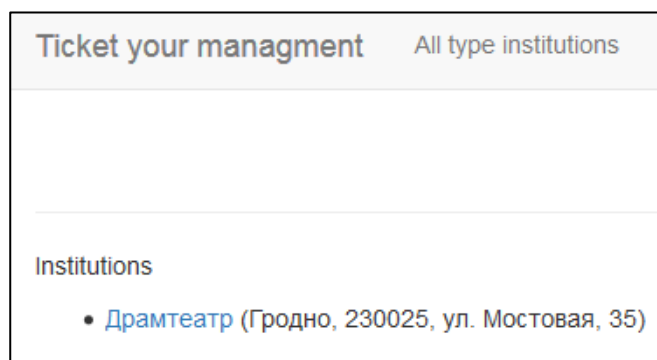


Рисунок 11 – Страница типа заведений «Театр»

По нажатию гиперссылки произошло перенаправление на страницу мероприятий, как видно из рисунка 4. После выбора «На всякого мудреца довольно простоты» отобразилась вкладка, изображенная на рисунке 12.

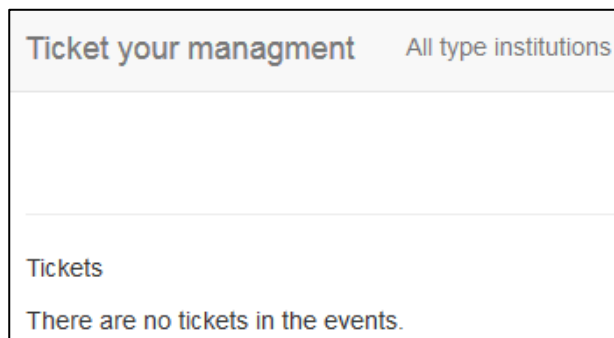


Рисунок 12 – Страница билетов мероприятия «На всякого мудреца  
довольно простоты»

Вернулись на предыдущую вкладку и выбрали «Бонусный сеанс

психоанализа», тем самым отобразилась страница как на рисунке 9. Выбрали «Place 30», отобразилось окно как на изображении 13.

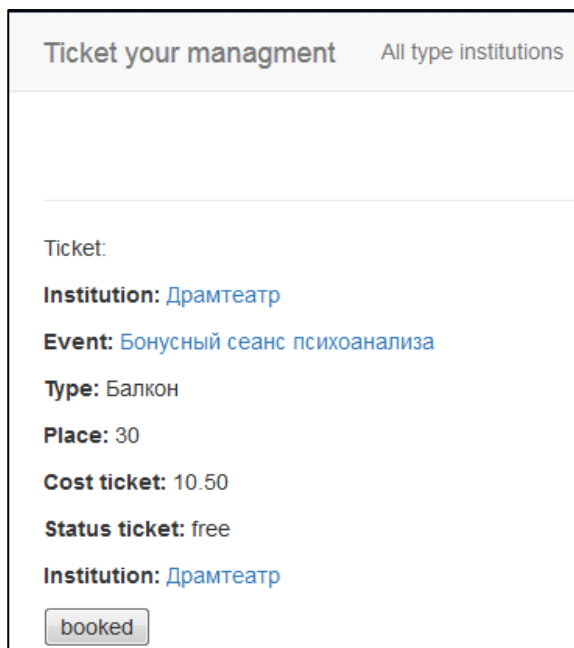


Рисунок 13 – Страница подробной информации тридцатого билета мероприятия «На всякого мудреца довольно простоты»

Нажали на кнопку «booked», после чего произошло перенаправление на страницу, изображенную на рисунке 7. Вернулись на предыдущую вкладку и увидели обновленный результат, как видно на изображении 14.

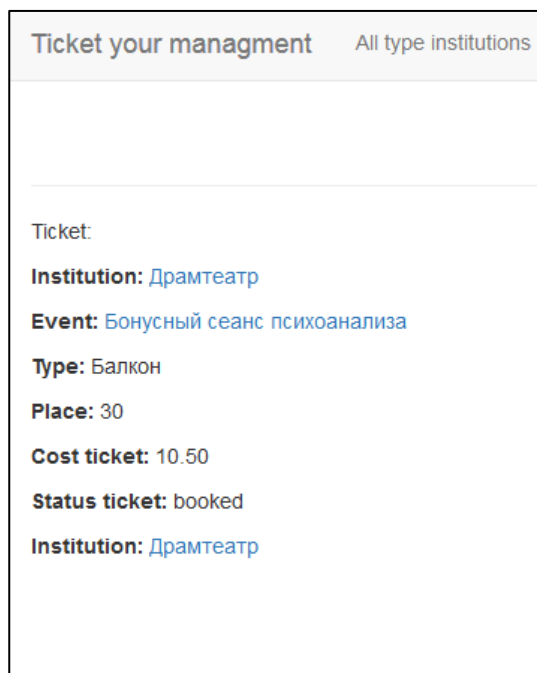
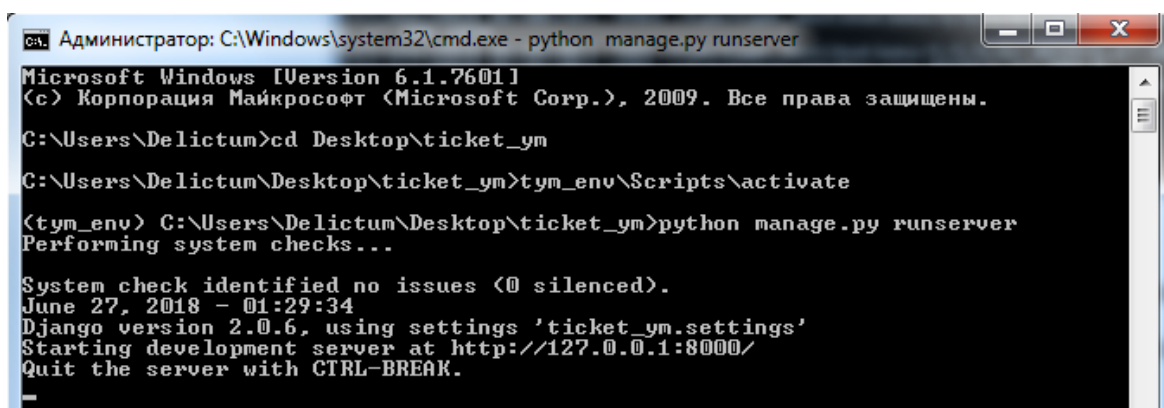


Рисунок 14 – Обновленный результат детальной информации билета после нажатия кнопки бронирования

В конечном итоге, можно утверждать, что функциональное тестирование было успешно пройдено и выявление ошибочных данных не было обнаружено.

## 6.2 Полное тестирование

В качестве полного тестирования произвели изменение записи билета. Для этого произвели запуск сервера. Был запущен cmd.exe, после чего в папке с проектом активировали виртуальную среду командой «tym\_env\Scripts\activate». Далее ввели «python manage.py runserver». Убедились, что сервер включен и находится в рабочем состоянии, как видно из рисунка 15.

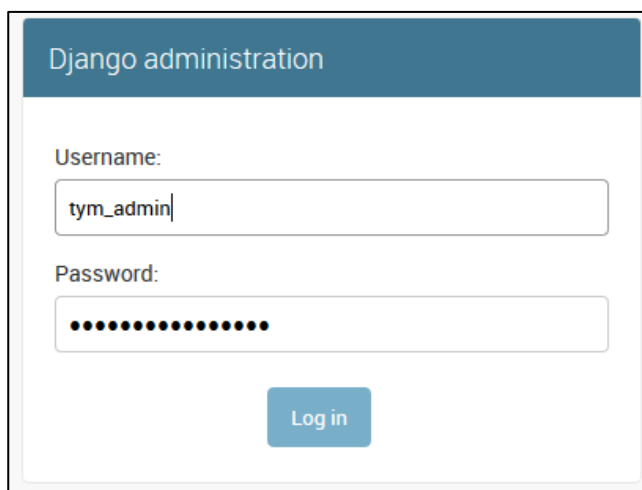


```
Администратор: C:\Windows\system32\cmd.exe - python manage.py runserver
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.
C:\Users\Delictum>cd Desktop\ticket_ym
C:\Users\Delictum\Desktop\ticket_ym>tym_env\Scripts\activate
(tym_env) C:\Users\Delictum\Desktop\ticket_ym>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
June 27, 2018 - 01:29:34
Django version 2.0.6, using settings 'ticket_ym.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Рисунок 15 – Запуск сервера

В браузерной строке ввели <http://localhost:8000/admin/>, после чего были направлены на страницу авторизации. Заполнили данные: поле «Username» — «tym\_admin», поле «Password» — «keb1wagenfo7ack72», как видно на изображении 16.



Django administration

Username:

Password:

Рисунок 16 – Авторизация администратора

Вкладка обновилась, как видно из рисунка 17.

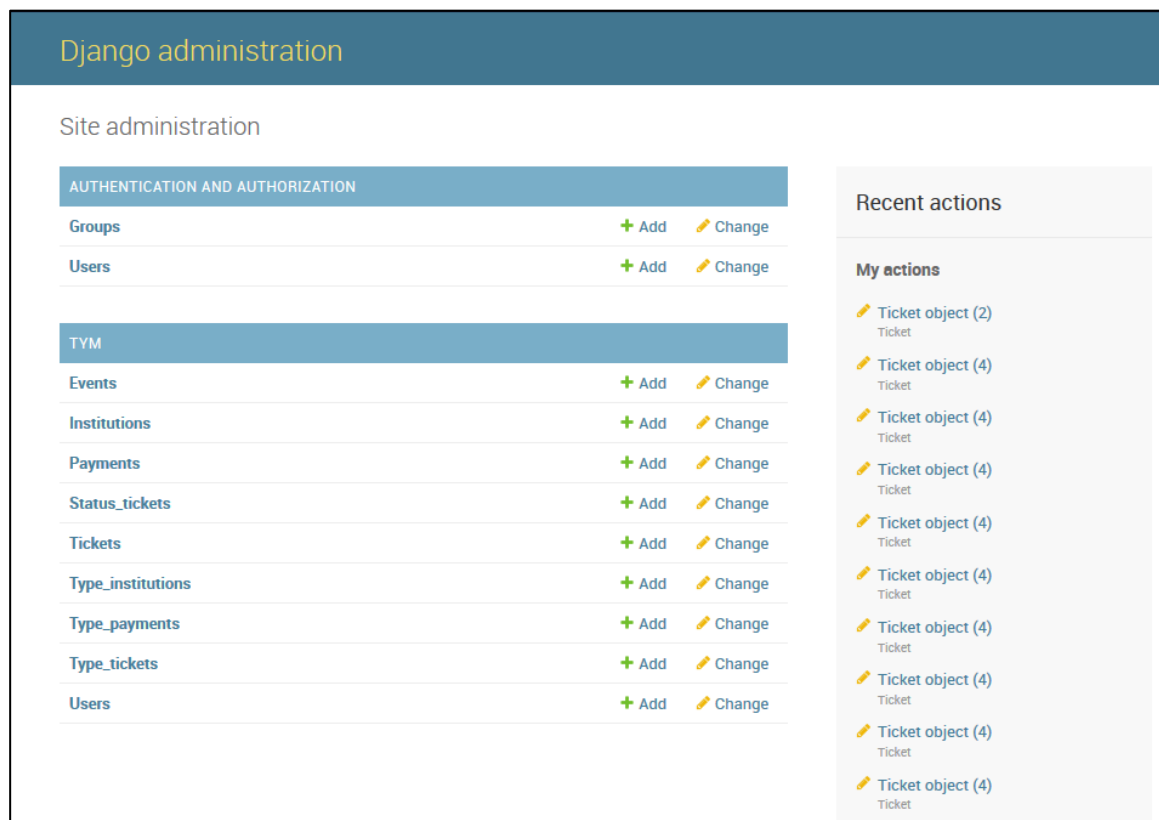


Рисунок 17 – Страница администратора

Выбрали «Tickets», после чего отобразилось окно с существующими объектами данной таблицы, как на изображении 18.

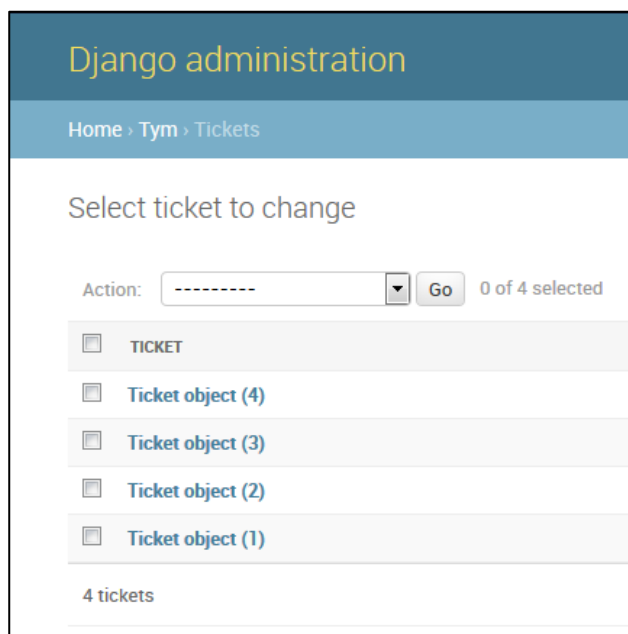


Рисунок 18 – Страница объектов таблицы «Ticket»

Нажали на «Ticket object (1)», после чего страница обновилась, как видно из рисунка 19.



Рисунок 19 – Страница объекта «Ticket object (1)»

Изменили данные: «Type ticket» - «Нижний зал», «Status ticket» - «free», «Seat number ticket» - 35, «Cost ticket» - 8. Поле «Event» оставили без изменений. Результат отображен на рисунке 20.

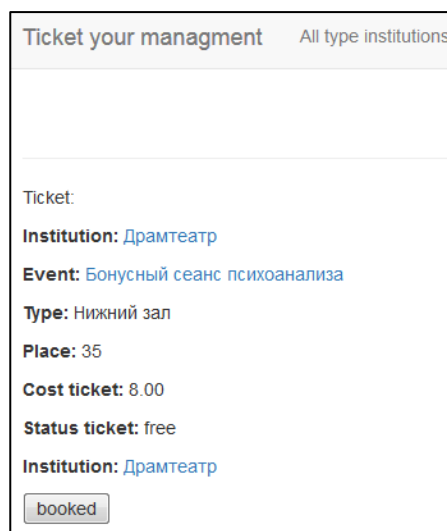
Рисунок 20 – Измененные поля объекта «Ticket object (1)»

Нажали «SAVE». Вкладка вернулась на предыдущую, сверху появилось уведомление об успешном изменении записи «ticket: «Ticket object (1)»».

Рисунок 21 – Уведомление об успешном изменении «Ticket object (1)»

Перешли на ссылку с детальной информацией измененного билета по

адресу [http://localhost:8000/tym/type\\_institutions/1/1/1/1/](http://localhost:8000/tym/type_institutions/1/1/1/1/). Результат виден на рисунке 22.



The screenshot shows a web application interface with a header bar containing the text 'Ticket your management' and 'All type institutions'. Below the header, there is a form displaying ticket details. The details are as follows:

Ticket:
Institution: Драмтеатр
Event: Бонусный сеанс психоанализа
Type: Нижний зал
Place: 35
Cost ticket: 8.00
Status ticket: free
Institution: Драмтеатр

At the bottom of the form, there is a button labeled 'booked'.

Рисунок 22 – Детальная информация измененного билета

В конечном итоге, web-приложение «Ticket your management» была полностью протестирована, ошибок не выявлено.

### 6.3 Применение

Web-приложение «Ticket your management» предназначено для бронирования и заказа билетов. Данное приложение может применяться повсеместно любыми пользователями, обладающими выходом в Интернет, а также наличием браузера.

Для осуществления и упрощения деятельности все функции автоматизированы. Пользователю требуется войти (зарегистрироваться) на сайте, после чего выбрать билет на желаемое мероприятие, затем нажать на необходимую кнопку.

Администратор сайта, помимо вышеперечисленных возможностей, также имеет доступ к полному функционалу web-приложения, позволяющему осуществлять по любым из записей базы данных добавление, изменение и удаление.

В результате программное средство было разработано, протестировано на ошибки и готово к эксплуатации.

## **7        ОЦЕНКА СТОИМОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

### **7.1    Определение объема и трудоемкости программного обеспечения**

В рыночных условиях программное обеспечение выступает преимущественно в виде продукции научно-технических организаций, представляющей собой функционально завершенные и имеющие товарный вид программные средства, поставляемые заказчикам и продаваемые покупателям по рыночным ценам. Все завершенные разработки ПО являются научно-

Экономический эффект у пользователя выражается в экономии трудовых, материальных и финансовых ресурсов, которая в конечном итоге также через уровень затрат, цену и объем продаж выступает в виде роста ЧД или ЧДД пользователя.

Стоимостная оценка ПО и определение экономического эффекта у работника предполагают составление сметы затрат, которая в денежном выражении включает следующие статьи расходов:

- 1) заработную плату исполнителей, основную ( $Z_o$ ) и дополнительную;
- 2) отчисления в фонд социальной защиты населения ( $Z_{сз}$ );
- 3) материалы и комплектующие ( $M$ );
- 4) спецоборудование ( $P_c$ );
- 5) машинное время ( $P_m$ );
- 6) расходы на научные командировки ( $P_{нк}$ );
- 7) прочие прямые расходы ( $P_3$ );
- 8) накладные расходы ( $P_n$ ).

На основе общей суммы расходов по всем статьям ( $C_p$ ) и результатов маркетинговых исследований на рынке ПО определяется плановая отпускная цена ( $Ц_o$ ) с учетом прибыли (рентабельности) и налогов, включаемых в цену.

Объем ПО. Базовой для расчёта плановой сметы затрат на разработку ПО

является объем ПО.

Общий объем ( $V_0$ ) программного продукта определяется исходя из количества и объема функций, реализуемых программой:

$$V_0 = \sum_{i=1}^n V_i \quad (1)$$

Где  $V_i$  – объем отдельной функции ПО;

$n$  – общее число функций.

Единицы измерения объема ПО. Оценивание объема программного продукта связано с выбором наиболее подходящей единицы измерения размера продукта. В зарубежной практике получили распространение следующие единицы измерения:

- 1) количество строк в исходного кода (Lines Of Code, LOC);
- 2) функциональные точки (Function point, FP);
- 3) точки свойств (Property point, PP);
- 4) количество сущностей на диаграмме сущностей (Entity relationship diagram, ERD);
- 5) количество сущностей на диаграмме сущностей (Entity relationship diagram, DFD);
- 6) количество «квадратиков», соответствующих процессу/контролю (PSPEC/CSPEC);
- 7) количество различных элементов в составе управленческой спецификации (element);
- 8) объем документации (количество строк, quantity lines);
- 9) количество объектов, атрибутов и служб на объектной диаграмме (subjects, attributes, services).

В данном методическом пособии в качестве единиц измерения объема ПО используется строка исходного кода (LOC). Преимущества использования строк кода как единиц измерения заключаются в том, что эти единицы:

- 1) отражают сущность труда программистов;
- 2) широко распространены и могут легко адаптироваться позволяют выполнять сопоставление размеров ПО и производительности в

различных группах разработчиков;

- 3) непосредственно связаны с конечным продуктом;
- 4) могут использоваться для оценки работ до завершения проекта; позволяют автоматизировать сбор данных о количестве LOC от начала до конца проекта;
- 5) дают возможность учитывать мнение разработчика об объеме ПО на основе количества написанных строк кода.

Строка исходного кода (LOC) является универсальной метрикой, так как может применяться при создании любых программных продуктов. При подсчете LOC следует придерживаться следующих рекомендаций:

- 1) учитывать «строку исходного кода» как одну, если в ней содержится лишь один оператор (если в одной строке содержатся два выполняемых оператора, разделяемых точкой с запятой, то нужно считать две строки, а если один выполняемый оператор разбит на две «физические» строки, то он будет учитываться как один оператор);
- 2) учитывать все имеющиеся выполняемые операторы, поддерживаемые данным продуктом;
- 3) определение данных учитывать лишь один раз;
- 4) не учитывать строки, содержащие комментарии;
- 5) не учитывать отладочный код или другой временный код (пробное ПО, средства тестирования, инструменты разработки и прототипирования и другие инструментальные средства);
- 6) учитывать каждую инициализацию, вызов или включение макроса в качестве части исходного кода;
- 7) не учитывать повторно используемые операторы исходного кода.

Среда разработки ПО – Python3.

ПО функциональные назначения.  $V_0 = 5000$  LOC.

Расчет объема программного продукта (количества строк исходного кода) предполагает определение типа программного обеспечения, всестороннее

техническое обоснование функций ПО и определение объема каждой функции. На стадии технико-экономического обоснования проекта невозможно рассчитать точный объем функций могут быть получены только ориентировочные (прогнозные) оценки на основе имеющихся фактических данных по аналогичным проектам, выполненным ранее, или путем применения действующих нормативов, которые в организациях должны периодически обновляться, уточняться и утверждаться как нормативы. На основании информации о функциях разрабатываемого ПО каталогу функций определяется объем функций и общий объем ПО, который уточняется (корректируется) с учетом условий разработки ПО в организации.

Таблица 12 – Перечень и объем функций программного модуля

№	Наименование (содержание)	Объем функции (LOC)
		По каталогу $V_i$
1	2	3
203	Формирование баз данных	2180
707	Графический вывод результатов	423
506	Обработка ошибочных и сбойных ситуаций	200
608	Функции и методы	1770
	Итого	4573

Трудоемкость разработки ПО. По общему объему ПО и нормативам затрат труда в расчете на единицу объема определяются нормативная и общая трудоемкость разработки ПО.

На основании принятого к расчету объема ( $V_0$ ) и категории сложности определяется нормативная трудоемкость ПО,  $T_n = 23$  человеко- дней.

Общая трудоемкость небольших проектов рассчитывается по форме

$$T_0 = T_n * K_c * K_T * K_n \quad (2)$$

где  $K_c$  - коэффициент, учитывающий сложность ПО;

$K_T$  – поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;

$K_n$  – коэффициент, учитывающий степень новизны ПО.

Категория сложности ПО. Все ПО принято подразделять на три категории сложности в зависимости от наличия следующих характеристик:

- 1) высокий уровень языкового интерфейса с пользованием;
- 2) режим работы в реальном времени;
- 3) управление удаленными объектами;
- 4) машинная графика, многомашинные комплексы;
- 5) существенное распараллеливание вычислений;
- 6) нестандартная конфигурация технических средств;
- 7) оптимизационные и особо сложные инженерные и научные расчеты;
- 8) переносимость ПО.

Коэффициент сложности ( $K_c$ ). Посредством коэффициента сложности учитываются дополнительные затраты труда, связанные со сложностью разрабатываемого программного продукта. Коэффициент сложности рассчитывается по формуле:

$$K_c = 1 + \sum_{i=1}^n K_i \quad (3)$$

где  $K_i$  – коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

$n$  – количество учитываемых характеристик.

Коэффициент сложности определяется по формуле 3:

$$K_c = 1 + 0,12 + 0,07 + 0,08 + 0,06 = 1,33$$

Степень охвата реализуемых функций разрабатываемого ПО стандартным модулям, типовыми программами и ПО – от 40% до 60%, соответственно  $K_t = 0,6$ .

Влияние фактора новизны на трудоемкость учитывается путем умножения трудоемкости на соответствующий коэффициент новизны, таким образом  $K_n = 1,2$ .

Общая трудоемкость определяется по формуле 2:

$$T_o = 23 * 1,33 * 0,6 * 1,2 = 21,4 \text{ человеко-дней}$$

Общая трудоемкость для крупных проектов. При решении сложных задач

с длительным периодом разработки ПО трудоемкость определяется по стадиям разработки:

- 1) техническое задание (ТЗ) – исследование;
- 2) эскизный проект (ЭП) – анализ требований;
- 3) технический проект (ТП) – проектирование;
- 4) рабочий проект (РП) – разработка (кодирование, тестирование);
- 5) внедрение (ВН) – ввод в действие.

При этом на основании нормативной трудоемкости рассчитывается общая трудоемкость с учетом распределения ее по стадиям ( $T_o$ ):

$$T_o = \sum_{i=1}^n T_i \quad (4)$$

где  $T_i$  – трудоемкость разработки ПО на  $i$ -й стадии (чел./дн.);

$n$  – количество стадий разработки.

Трудоемкость стадий определяется на основе нормативной трудоемкости с учетом сложности, новизны, степени использования в разработке стандартных модулей ПО и удельного веса трудоемкости каждой стадий в общей трудоемкости ПО:

$$T_{yi} = T_n * d_{cti} * K_c * K_t * K_n \quad (5)$$

где  $T_{yi}$  – уточнённая трудоемкость разработки ПО на  $i$ -й стадии (технического задания, эскизного проекта, технического проекта, рабочего проекта и внедрения);

$d_{cti}$  – удельный вес трудоемкости  $i$ -й стадии разработки ПО в общей трудоемкости разработки ПО;

$K_c$  – коэффициент, учитывающий сложность ПО, вводится на всех стадиях;

$K_t$  – коэффициент, учитывающий степень использования стандартных модулей ПО, вводится только на стадии рабочего проекта;

$K_n$  – коэффициент, учитывающий степень новизны ПО, вводится на всех стадиях.

$$T_{trp} = 0.85 * T_{tp} + 1 * T_{rp} \quad (6)$$

где  $T_{trp}$  – трудоемкость стадии «Технорабочий проект»;



$K_{тп}$  – трудоемкость стадии «Технический проект»;

$K_{рп}$  – трудоемкость стадии «Рабочий проект».

Трудоемкость ПО по стадиям. Все стадии разработки ПО различаются трудоемкостью. Трудоемкость разработки стадий ПО ( $T_{уз}$ ,  $T_{уэ}$ ,  $T_{уг}$ ,  $T_{ур}$ ,  $T_{ув}$ ) определяется с учетом удельного веса трудоемкости стадии в общей трудоемкости ПО ( $d$ ), сложности ( $K_c$ ), новизны ПО ( $K_n$ ) и степени использования стандартных модулей ( $K_t$ ). При этом коэффициент ( $K_t$ ) используется только на стадии «Рабочий проект» при написании исходного кода (разработки программы). Трудоемкость стадий ПО рассчитывается по следующим формулам:

$$\text{трудоемкость стадии ТЗ: } T_{уз} = T_n * K_c * d_3 * K_n = 21,4 * 0,11 * 1,33 * 1,2 = 3,8 \quad (7)$$

$$\text{трудоемкость стадии ЭП: } T_{уэ} = T_n * K_c * d_э * K_n = 21,4 * 0,09 * 1,33 * 1,2 = 3,1 \quad (8)$$

$$\text{трудоемкость стадии ТП: } T_{уг} = T_n * K_c * d_t * K_n = 21,4 * 0,11 * 1,33 * 1,2 = 3,8 \quad (9)$$

$$\text{трудоемкость стадии РП: } T_{ур} = T_n * K_c * d_p * K_n * K_t = 21,4 * 0,55 * 1,33 * 1,2 * 0,7 = 13,1 \quad (10)$$

$$\text{трудоемкость стадии ВН: } T_{ув} = T_n * K_c * d_b * K_n = 21,4 * 0,14 * 1,33 * 1,2 = 4,8. \quad (11)$$

Общая трудоемкость определяется как сумма трудоемкостей по стадиям:

$$T_y = T_{уз} + T_{уэ} + T_{уг} + T_{ур} + T_{ув} \quad (12)$$

Общая трудоемкость по стадиям, исходя из формулы 12:

$$T_y = 3,8 + 3,1 + 3,8 + 13,1 + 4,8 = 28,6 \text{ человеко-дней.}$$

В конечном итоге, общая трудоёмкость составляет 28,6 человеко-дня.

## **7.2 Расчёт эффективного фонда времени и численности работников**

Эффективный фонд времени одного работника ( $\Phi_{эф}$ ) рассчитывается по формуле:

$$\Phi_{эф} = D_r - D_p - D_b - D_o \quad (13)$$

где  $D_r$  – количество дней в году;

$D_p$  – количество праздничных дней в году;

$D_b$  – количество выходных дней в году;

$D_o$  – количество дней отпуска.

Эффективный фонд времени определяется по формуле 13:

$$\Phi_{\text{эф}} = 365 - 8 - 104 - 24 = 229 \text{ дн.};$$

На основании уточненной трудоемкости программного средства и установленного периода разработки рассчитывается общая плановая численность исполнителей по формуле:

$$Ч_{\text{раз}} = T_{\text{ут}} / (T_{\text{пл.}} * \Phi_{\text{эф}}) \quad (14)$$

где  $T_{\text{ут}}$  – уточненная трудоёмкость программного средства;

$T_{\text{пл.}}$  – плановая продолжительность разработки программного средства (лет);

$Ч_{\text{раз}}$  – общая плановая численность разработчиков (человек);

$\Phi_{\text{эф}}$  – эффективный фонд времени одного работника.

### 7.3 Расчёт основной и дополнительной заработной платы

Основной статьей расходов на создание ПО является заработная плата работников (исполнительного) проекта, в число которых принято включать инженеров-программистов, участвующих в написании кода, руководителей проекта, системных архитекторов, дизайнеров, разрабатывающих пользовательский интерфейс, разработчиков баз данных, web-мастеров и других специалистов, необходимых для решения специальных задач в команде.

Месячная тарифная ставка каждого исполнителя ( $T_m$ ) определяется путем умножения действующей месячной тарифной ставки 1-го разряда ( $T_{m1}$ ) на тарифный коэффициент ( $T_k$ ), соответствующий установленному тарифному разряду:

$$T_m = T_{m1} * T_k \quad (15)$$

Часовая тарифная ставка рассчитывается путем деления месячной тарифной ставки на установленную при 40-часовой недельной норме рабочего времени расчётную среднемесячную норму рабочего времени в часах ( $\Phi_p$ ):

$$T_{\text{ч}} = T_m / \Phi_p \quad (16)$$

где  $T_{\text{ч}}$  – часовая тарифная ставка (д.е.);

$T_{\text{м}}$  – месячная тарифная ставка (д.е.).

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле:

$$Z_{oi} = \sum_{i=1}^n T_{\text{чи}} * T_{\text{ч}} * \Phi_{\text{п}} * K \quad (17)$$

где  $n$  – количество исполнителей, занятых разработкой конкретного ПО;

$T_{\text{чи}}$  – часовая тарифная ставка  $i$ -го исполнителя (д.е.);

$\Phi_{\text{п}}$  – плановый фонд рабочего времени  $i$ -го исполнителя (дн.);

$T_{\text{ч}}$  – количество работы в день (ч);

$K$  – коэффициент премирования.

В соответствии со штатным расписанием на разработке будут заняты:

- 1) программист 2 категории – тарифный разряд – 12; тарифный коэффициент – 2.84.

Расчётные данные представим в виде таблицы.

Таблица 13 – Расчёт основной заработной платы

Исполнители	Тарифная ставка 1-го разряда ( $T_{\text{м1}}$ ), тыс.р.	Тарифный коэффициент ( $T_{\text{к}}$ )	Тарифная ставка данного разряда ( $T_{\text{м}}$ ). Тыс.р.	Эффективный фонд работы за месяц ( $H_{\text{мес}}$ )	Тарифная ставка часовая ( $T_{\text{ч}}$ ) тыс.р.	Тарифная ставка дневная ( $T_{\text{дн.}}$ ) тыс.р.	Продолжительность участия в разработке, дн.	Коэффициент премирования	Заработная плата основная ( $Z_o$ )
Программист второй категории	34	2,84	96,56	160	0,60	4,8	170	1,8	492,48
Итого фонд основной заработной платы	-	-	-	-	-	-	-	-	492,48

Дополнительная заработная плата на конкретное ПО ( $Z_{\text{дп}}$ ) включает выплаты, предусмотренные законодательством о труде (оплата отпусков,

льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной плате:

$$З_{дi} = \frac{З_{oi} * Н_d}{100} \quad (18)$$

где  $З_{дi}$  – дополнительная заработная плата исполнителей на конкретное ПО (д.е.);

$Н_d$  – норматив дополнительной заработной платы ( $Н_d = 17$ );

$З_o$  – основная заработная плата в целом по организации.

Дополнительная заработная плата исполнителей исходя из формулы 18:

$$З_{дi} = \frac{492,48 * 17}{100} = 83,72 \text{ рублей}$$

Отчисления в фонд социальной защиты населения ( $З_{сzi}$ ) определяются в соответствии с действующими законодательными актами по нормативу процентном отношении к фонду основной и дополнительной зарплаты исполнителей, определенной по нормативу, установленному в целом по организации:

$$З_{сzi} = \frac{(З_{oi} + З_{qi}) * Н_{сз}}{100} \quad (19)$$

где  $Н_{сз}$  – норматив отчислений в фонд социальной защиты населения (%).

Отчисления в фонд социальной защиты населения, высчитанный из формулы 19:

$$З_{сzi} = \frac{(492,48 + 83,72) * 34}{100} = 195,74 \text{ рублей}$$

Отчисления в Белгосстрах ( $З_{бгс}$ ) определяются в соответствии действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей.

$$З_{бгс} = \frac{(З_{oi} + З_{qi}) * Н_{бгс}}{100} \quad (20)$$

$Н_{бгс}$  – норматив отчислений Белгосстрах (%).

Отчисления в Белгосстрах, исходя из формулы 20:

$$З_{бгс} = \frac{(492,48 + 83,72) * 0,6}{100} = 3,45 \text{ рублей}$$

Налоги, рассчитываемые от фонда оплаты труда, определяются в соответствии с действующими законодательными актами по нормативам в процентном отношении к сумме всей заработной платы, относимой на ПО.

#### 7.4 Расчёт затрат на материалы и спецоборудование

Расходы по статье «Материалы» определяются на основании сметы затрат, разрабатываемой на ПО с учетом действующих нормативов. По статье «Материалы» отражаются расходы на магнитные бумагу, красящие ленты и другие материалы, необходимые разработки ПО. Нормы расхода для материалов в суммарном выражении ( $H_m$ ) определяются 100 в расчете на строки исходного кода или по нормативу в процентах к фонду основной заработной платы разработчиков (но, который устанавливается организацией (3-5%). Сумма затрат на расходные материалы ( $M_i$ ) рассчитывается по формуле:

$$M_i = H_m * (V_{oi} / 100) \quad (21)$$

где  $H_m$  – норма расхода материалов в расчете на 100 строк исходного кода ПО (д.е.);

$V_{oi}$  – общий объем ПО (строк исходного кода) на конкретное ПО.

Или по формуле:

$$M_i = \frac{V_{oi} * H_{мз}}{100} \quad (22)$$

где  $H_{мз}$  – норма расхода материалов от основной заработной платы (%).

Сумма затрат на расходные материалы, исходя из формулы 22:

$$M_i = \frac{492,48 * 3}{100} = 14,77 \text{ рублей}$$

Данная статья включается в смету расходов на разработку в том случае, когда приобретаются специальное оборудование или специальные программы, предназначенные для разработки и создания только данного ПО:

$$P_{Ci} = \sum_{i=1}^n C_{ci} \quad (23)$$

где  $C_{ci}$  – стоимость конкретного специального оборудования (д.е.);

$n$  – количество применяемого специального оборудования.

## 7.5 Расчёт расходов на оплату машинного времени

Расходы по статье «Машинное время» ( $P_{mi}$ ) включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам (в машино-часах) на 100 строк исходного, когда ( $H_{mb}$ ) машинного времени в зависимости от характера решаемых задач и типа ПК:

$$P_{mi} = C_{mi} * V_{oi} / 100 * H_{mb} \quad (24)$$

где  $C_{mi}$  – цена одного машино-часа (д.е.);

$V_{oi}$  – общий объём ПО (строк исходного кода);

$H_{mb}$  – норматив расхода машинного времени на отладку 100 строк исходного кода (машино-часов).

Расходы по статье «Машинное время» по формуле 24:

$$P_{mi} = 4573 / 100 * 0.1 = 4,57 \text{ рублей}$$

## 7.6 Расчет прочих затрат

Расходы по статье «Научные командировки» ( $P_{nki}$ ) на конкретное ПО определяются по нормативу, разрабатываемому в целом по организации, в процентах к основной заработной плате:

$$P_{nki} = (Z_{oi} * H_{rnk}) / 100 \quad (25)$$

где  $H_{rnk}$  – норматив расходов на командировки в целом по организации (%).

Расходы по статье «Научные командировки», исходя из формулы 25:

$$P_{nki} = (492,48 * 25) / 100 = 123,12 \text{ рублей}$$

$P_{nk}$  – расходы на командировки в целом по организации (д.е.).

Расходы по статье «Прочие затраты» ( $P_{zi}$ ) на конкретное ПО включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по нормативу, разрабатываемому в целом по организации, в процентах к основной заработной плате:

$$П_{зi}=(З_{oi}*Н_{пз})/100 \quad (26)$$

где  $H_{пз}$  – норматив прочих затрат в целом по организации;

$П_з$  – прочие затраты в целом по организации.

Расходы по статье «Прочие затраты», высчитанные по формуле 26:

$$П_{зi}=(492,48*20)/100 =98,50 \text{ рублей}$$

Затраты по статье «Накладные расходы» ( $P_{ни}$ ), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды ( $P_{ни}$ ), относятся на конкретное ПО нормативу ( $H_{рн}$ ) в процентном соотношении к основной заработной плате исполнителей. Норматив устанавливается в целом по организации:

$$P_{ни}= (З_{oi}*H_{рн})/100 \quad (27)$$

где  $P_{ни}$  – накладные расходы на конкретную ПО (д.е.);

$H_{рн}$  – норматив накладных расходов в целом по организации (Прил. 7)

где  $P_n$  – накладные расходы в целом по организации (д.е.).

Расходы по статье «Накладные расходы», исходя из формулы 27:

$$P_{ни}=(492,48*10)/100 = 48,84 \text{ рублей}$$

## 7.7 Составление сметы затрат

Себестоимость продукции представляет собой стоимостную оценку используемых в процессе производства продукции природных ресурсов, сырья, материалов, топлива, энергии, основных фондов, трудовых ресурсов и других затрат на ее производство и реализацию.

По элементам затрат составляются сметы затрат на производство и реализацию всей товарной продукции, представленные в таблице 14.

Таблица 14 – Смета затрат

Наименование статей затрат	Обозначение	Сумму
Основная заработная плата	$З_{oi}$	492,48
Дополнительная заработная плата	$З_{дi}$	83,72

Отчисления в фонд социальной защиты населения	$Z_{сzi}$	195,74
Отчисления в Белгосстрах	$Збгс$	3,45
Материалы	$M_i$	14,77
Спецоборудование	$P_{ci}$	-
Машинное время	$P_{mi}$	4,57
Научные командировки	$P_{нki}$	123,12
Прочие затраты	$П_{zi}$	98,50
Накладные расходы	$P_{ni}$	48,84
Итого производственная себестоимость	$C_{пр}$	1065,19

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию программного средства ( $P_{ci}$ ). Эти затраты определяются по нормативу в процентах от производственной себестоимости по формуле:

$$P_{ci} = C_{пр} * H_{ca} / 100 \quad (28)$$

$C_{пр}$  – производственная себестоимость;

$H_{ca}$  – норматив отчисления на сопровождение и адаптацию программного средства.

Затраты на сопровождение и адаптацию программного средства, высчитанные по формуле 28:

$$P_{ci} = 1065,19 * 16 / 100 = 170,43 \text{ рублей}$$

Таким образом, полная себестоимость разработки программного средства ( $C_{пол}$ ) определяется по формуле:

$$C_{пол} = C_{пр} + P_{ci} \quad (29)$$

Полная себестоимость разработки программного средства, высчитанная по формуле 29:

$$C_{пол} = 1065,19 + 170,43 = 1235,62 \text{ рублей}$$

Таким образом, были произведены расчёты по производственной себестоимости программного продукта.



Сопровождение программного обеспечения – процесс улучшения, оптимизации и устранения дефектов программного обеспечения после передачи в эксплуатацию.

Данное web-приложение в будущем будет локально адаптировано, расширит диапазон включенных записей базы данных, более детальную обработку платежей. Оформление общей стилистики сайта будет усовершенствовано. В том числе сформируется маркетинговый отдел.

Для обеспечения эффективного сопровождения программных систем необходимо решать целый комплекс вопросов и проблем, связанных с соответствующими работами. Именно процесс сопровождения позволяет улучшить удовлетворенность пользователей внедренным программным обеспечением.

Существующее программное обеспечение никогда не бывает полностью завершенным и продолжает эволюционировать в течение всего срока эксплуатации. В процессе развития программная система становится все более сложной до тех пор, пока не прикладываются специальные усилия по уменьшению ее сложности.

В качестве повышения оптимизации, база данных будет переведена на более мощную СУБД – PostgreSQL. Также будет осуществление мобильной версии сайта и мобильного приложения под Android и iOS.

Тем самым, IT-рынок — структура быстро растущая и динамичная, поэтому помимо поддержки программного обеспечения будет осуществлено доработка программного средства, а именно:

- 1) реинжиниринг;
- 2) интернационализация и локализация;
- 3) портирование и миграцию программного обеспечения.

Миграция будет осуществляться постепенно, что позволит избежать

рисков.

Данный подход к сопровождению web-приложения «Ticket your management» позволит максимально реализовать идею проекта и осуществлять его деятельность в течении длительного периода времени.

## ЗАКЛЮЧЕНИЕ

Задачей данного проекта по учебной практике являлась разработка web-приложения по бронированию билетов различных заведений.

Для реализации программного средства был выбран язык программирования Python v.3.6.5 на библиотеке «Django» v.2.0.6 со встроенной библиотекой СУБД SQLite при помощи консоли и IDLE Python Windows.

На проектирование методологий IDEF0, IDEF1, IDEF1X были созданы функциональные модели в соответствии с данным программным продуктом, на котором отобразили необходимые данные. В проектировании информационной базы проекта была смоделирована схема «Сущность-связь», на которой указали требуемые таблицы и атрибуты для дальнейшей успешной реализации программы.

При проектировании следующих диаграмм: вариантов использования, классов, состояний, деятельности, последовательности, кооперации, компонентов и развертывания, — использовали язык моделирования UML и программное кейс-средство «Rational Rose», а также «EDRAW MAX», «Microsoft Office Word» и «Microsoft Office Visio». Результатом проектирования были созданы указанные диаграммы.

В разделе «Реализация программы» был спроектирован интерфейс web-приложения. Была осуществлена миграция базы данных, а также осуществлены все функции программы. Результатом данного раздела стал готовый программный продукт.

После вся программа неоднократно тестировалась на правильность получаемых результатов, все возникающие ошибки были устранены в ходе работы. Тестирование происходило с помощью метода «Чёрного ящика». После проведенного тестирования можно установить, что программа удовлетворяет всем поставленным перед ней требованиям и является готовым программным средством.

Следующим этапом произвели расчёт затрат на создание программного

продукта. Итогом данного расчёта выявили цену создания – 1235,62 рублей.

На этапе сопровождения программного продукта описали этапы, а также, основные концепции, процесс сопровождения, ключевые вопросы и техники сопровождения. Основные модификации по дальнейшему улучшению заключаются в:

- 1) реинжиниринге;
- 2) интернационализации и локализации;
- 3) портировании и миграции программного обеспечения.

Достоинством проекта является простота и интуитивность программного средства, а также для использования требуется только подключение к сети-Интернет и наличие браузера.

Данное web-приложение является простым и полезным в использовании для осуществления дистанционного бронирования и заказа билетов.

Исходя из вышесказанного можно утверждать о выполненной цели учебной практики. Данный проект – «Ticket your management» – реализовал поставленные задачи в соответствии с заданным условием.

## ЛИТЕРАТУРА

- 1 Гринзоу, Л. Философия программирования для Windows / Л. Гринзоу. – Москва: Вильямс, 2002. – 528 с.
- 2 Левин, А. Самоучитель работы Windows / А. Левин. – Минск: Омега, 2005. – 215 с.
- 3 Мэтиз, Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Э. Мэтиз. – СПб.: Питер, 2017. — 496 с.
- 4 Доусон, М. Програмируем на Python / М. Доусон. – СПб.: Питер, 2014. – 416 с.
- 5 Любанович, Б. Простой Python. Современный стиль программирования / Б. Любанович. – Питер, 2017. – 480с.
- 6 Саммерфилд, М. Программирование на Python 3. Подробное руководство / М. Саммерфилд. – Издательство «Символ», 2015. – 608с.
- 7 Лутц, М. Python. Карманный справочник /М. Лутц. – Издательство «Вильямс», 2015. – 320с.
- 8 Дронов, В. Django: практика создания Web-сайтов на Python / В. Л– Издательство «ВНУ», 2016. – 528с.

## ПРИЛОЖЕНИЕ А

### (обязательное)

#### Текст программы

##### Листинг 1 – файл «manage.py»:

```
#!/usr/bin/env python
import os
import sys
if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "ticket_ym.settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
```

##### Листинг 2 – файл «views.py»:

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponse, HttpResponseRedirect
from .models import Institution, Type_institution, Event, Type_payment
from .models import Ticket, Type_ticket, Status_ticket
from django.db import models
def index(request):
    num_type_institution = Type_institution.objects.all().count()
    num_institution = Institution.objects.all().count()
    num_event = Event.objects.all().count()
    num_type_payment = Type_payment.objects.all().count()
    return render(
        request,
        'index.html',
        context={
            'num_institution':num_institution,
            'num_type_institution':num_type_institution,
            'num_event':num_event, 'num_type_payment':num_type_payment},
        )
def type_institutions(request):
```

```

type_institutions = Type_institution.objects.order_by('type_institution')
return render(
    request,
    'type_institutions.html',
    context={'type_institutions':type_institutions}
)

def institutions(request, id):
    institution_id = Type_institution.objects.get(id=id)
    institutions = Institution.objects.filter(type_institution=institution_id)
    return render(
        request,
        'institutions.html',
        context={'institutions':institutions}
    )

def events(request, id):
    event_id = Institution.objects.get(id=id)
    events = Event.objects.filter(institution=event_id)
    return render(
        request,
        'events.html',
        context={'events':events}
    )

def tickets(request, id):
    ticket_id = Event.objects.get(id=id)
    tickets = Ticket.objects.filter(event=ticket_id)
    type_tickets = Type_ticket.objects.order_by('type_ticket')
    type_tickets_count = Type_ticket.objects.count()
    return render(
        request,
        'tickets.html',
        context={'tickets':tickets, 'type_tickets':type_tickets,
        'type_tickets_count':type_tickets_count}
    )

def ticket_details(request, id):
    ticket_id = Ticket.objects.get(id=id)
    status_ticket = Status_ticket.objects.order_by('status_ticket')
    return render(
        request,
        'ticket_details.html',
        context={'ticket_id':ticket_id, 'status_ticket':status_ticket}
    )

def ticket_booked(request, id):

```

```

ticket = get_object_or_404(Ticket, id=id)
ticket.status_ticket = Status_ticket.objects.get(status_ticket='booked')
ticket.save()
return render(
    request,
    'ticket_booked.html',
    context={'ticket':ticket}
)

```

### Листинг 3 – файл «models.py»:

```

from django.db import models
import uuid

class Type_institution(models.Model):
    """Тип заведения"""
    type_institution = models.CharField(max_length=100)
    def __str__(self):
        return self.type_institution

class Institution(models.Model):
    """Информация о заведении"""
    type_institution = models.ForeignKey(Type_institution,
                                         on_delete=models.CASCADE, )
    name_institution = models.CharField(max_length=200)
    adress_institution = models.CharField(max_length=200)
    def __str__(self):
        return self.name_institution

class Event(models.Model):
    institution = models.ForeignKey(Institution,
                                    on_delete=models.CASCADE, )
    name_event = models.TextField()
    date_added = models.DateTimeField()
    def __str__(self):
        return self.name_event

class Type_ticket(models.Model):
    type_ticket = models.CharField(max_length=100)
    def __str__(self):
        return self.type_ticket

class Status_ticket(models.Model):
    status_ticket = models.CharField(max_length=30)
    def __str__(self):
        return self.status_ticket

class Ticket(models.Model):
    event = models.ForeignKey(Event,
                              on_delete=models.CASCADE, )

```



```

type_ticket = models.ForeignKey(Type_ticket,
                                on_delete=models.CASCADE, )
status_ticket = models.ForeignKey(Status_ticket,
                                on_delete=models.CASCADE, )
seat_number_ticket = models.IntegerField()
cost_ticket = models.DecimalField(max_digits=7, decimal_places=2)
def __int__(self):
    return self.seat_number_ticket
class Type_payment(models.Model):
    type_payment = models.CharField(max_length=100)
    def __str__(self):
        return self.type_payment
class User(models.Model):
    login_user = models.TextField()
    password_user = models.DateTimeField()
    def __str__(self):
        return self.login_user
class Payment(models.Model):
    user = models.ForeignKey(User,
                            on_delete=models.CASCADE, )
    type_payment = models.ForeignKey(Type_payment,
                                    on_delete=models.CASCADE, )
    ticket = models.ForeignKey(Ticket,
                              on_delete=models.CASCADE, )
    date_payment = models.DateField()
    count_ticket = models.IntegerField()
    def __datetime__(self):
        return self.date_payment

```

#### Листинг 4 – файл «settings.py»:

```

"""
Django settings for ticket_ym project.
Generated by 'django-admin startproject' using Django 2.0.6.
For more information on this file, see
https://docs.djangoproject.com/en/2.0/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/

```

```

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'cg#p$g+j9tax!#a3cup@1$8obt2_+&k3q+pmu)5%asj6yjpkg'
# SECURITY WARNING: don't run with debug turned on in production!
#DEBUG = True
DEBUG = bool( os.environ.get('DJANGO_DEBUG', True) )
ALLOWED_HOSTS = []
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bootstrap3',
    #Мои приложения
    'ym',
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'ticket_ym.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

```

```

]
WSGI_APPLICATION = 'ticket_ym.wsgi.application'

# Database
# https://docs.djangoproject.com/en/2.0/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Password validation
# https://docs.djangoproject.com/en/2.0/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/2.0/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.0/howto/static-files/
STATIC_URL = '/static/'

# Heroku: Update database configuration from $DATABASE_URL.
import dj_database_url
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)

BOOTSTRAP3 = {
    'include_jquery': True,

```

```
}
```

## Листинг 5 – файл «urls.py»:

```
"""ticket_ym URL Configuration
```

The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/2.0/topics/http/urls/>

Examples:

Function views

1. Add an import: from my\_app import views
2. Add a URL to urlpatterns: path("", views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home
2. Add a URL to urlpatterns: path("", Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

```
"""
```

```
from django.conf.urls import url
```

```
from django.urls import path, re_path
```

```
from . import views
```

```
urlpatterns = [
```

```
    path("", views.index, name='index'),
```

```
    path('type_institutions', views.type_institutions,  
         name='type_institutions'),
```

```
    re_path(r'^type_institutions/(?P<id>\d+)/$',  
           views.institutions, name='institutions'),
```

```
    re_path(r'^type_institutions/(\d+)/(?P<id>\d+)/$',  
           views.events, name='events'),
```

```
    re_path(r'^type_institutions/(\d+)/(\d+)/(?P<id>\d+)/$',  
           views.tickets, name='tickets'),
```

```
    re_path(r'^type_institutions/(\d+)/(\d+)/(\d+)/(?P<id>\d+)/$',  
           views.ticket_details, name='ticket_details'),
```

```
    re_path(r'^type_institutions/(\d+)/(\d+)/(\d+)/(?P<id>\d+)/ticket_booked/$',  
           views.ticket_booked, name='ticket_booked'),
```

```
]
```

## Листинг 6 – файл «admin.py»:

```
from django.contrib import admin
```

```
from .models import Type_institution, Institution, Event, Type_payment
```

```
from .models import Type_ticket, Status_ticket, Payment, Ticket, User
```

```
admin.site.register(Type_institution)
```

```
admin.site.register(Institution)
```

```
admin.site.register(Event)
```

```
admin.site.register(Type_payment)
admin.site.register(Type_ticket)
admin.site.register(Status_ticket)
admin.site.register(Payment)
admin.site.register(Ticket)
admin.site.register(User)
```

### Листинг 7 – файл «0005\_auto\_20180617\_1026.py»:

```
# Generated by Django 2.0.6 on 2018-06-17 07:26
from django.db import migrations, models
import django.utils.timezone

class Migration(migrations.Migration):

    dependencies = [
        ('tym', '0004_remove_status_ticket_imprint'),
    ]

    operations = [
        migrations.AlterModelOptions(
            name='status_ticket',
            options={},
        ),
        migrations.RemoveField(
            model_name='status_ticket',
            name='due_back',
        ),
        migrations.RemoveField(
            model_name='status_ticket',
            name='status',
        ),
        migrations.AddField(
            model_name='status_ticket',
            name='status_ticket',
            field=models.CharField(default=django.utils.timezone.now, max_length=30),
            preserve_default=False,
        ),
        migrations.AlterField(
            model_name='status_ticket',
            name='id',
            field=models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID'),
        ),
    ]
```

### Листинг 8 – файл «0003\_auto\_20180617\_0108.py»:

```
# Generated by Django 2.0.6 on 2018-06-16 22:08
```

```

from django.db import migrations, models
import django.db.models.deletion
import uuid
class Migration(migrations.Migration):
    dependencies = [
        ('tym', '0002_entry'),
    ]
    operations = [
        migrations.CreateModel(
            name='Event',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('name_event', models.TextField()),
                ('date_added', models.DateTimeField()),
            ],
        ),
        migrations.CreateModel(
            name='Institution',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('name_institution', models.CharField(max_length=200)),
                ('adress_institution', models.CharField(max_length=200)),
            ],
        ),
        migrations.CreateModel(
            name='Payment',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('date_payment', models.IntegerField()),
                ('count_ticket', models.IntegerField()),
            ],
        ),
        migrations.CreateModel(
            name='Status_ticket',
            fields=[
                ('id', models.UUIDField(default=uuid.uuid4, help_text='Unique ID for this particular ticket across
whole event', primary_key=True, serialize=False)),
                ('imprint', models.CharField(max_length=200)),
                ('due_back', models.DateField(blank=True, null=True)),
            ],
        ),
    ]

```

```

        ('status', models.CharField(blank=True, choices=[('f', 'Free'), ('b', 'Booked'), ('s', 'Sold')], default='f',
help_text='Book availability', max_length=1)),
    ],
    options={
        'ordering': ['due_back'],
    },
),
migrations.CreateModel(
    name='Ticket',
    fields=[
        ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
        ('seat_number_ticket', models.IntegerField()),
        ('cost_ticket', models.DecimalField(decimal_places=2, max_digits=7)),
        ('event', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='tym.Event')),
        ('status_ticket', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
to='tym.Status_ticket')),
    ],
),
migrations.CreateModel(
    name='Type_institution',
    fields=[
        ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
        ('type_institution', models.CharField(max_length=100)),
    ],
),
migrations.CreateModel(
    name='Type_payment',
    fields=[
        ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
        ('type_payment', models.CharField(max_length=100)),
    ],
),
migrations.CreateModel(
    name='Type_ticket',
    fields=[
        ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
        ('type_ticket', models.CharField(max_length=100)),
    ],

```

```

    ),
    migrations.CreateModel(
        name='User',
        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
            ('login_user', models.TextField()),
            ('password_user', models.DateTimeField()),
        ],
    ),
    migrations.RemoveField(
        model_name='entry',
        name='topic',
    ),
    migrations.DeleteModel(
        name='Entry',
    ),
    migrations.DeleteModel(
        name='Topic',
    ),
    migrations.AddField(
        model_name='ticket',
        name='type_ticket',
        field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='tym.Type_ticket'),
    ),
    migrations.AddField(
        model_name='payment',
        name='ticket',
        field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='tym.Ticket'),
    ),
    migrations.AddField(
        model_name='payment',
        name='type_payment',
        field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
to='tym.Type_payment'),
    ),
    migrations.AddField(
        model_name='payment',
        name='user',
        field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='tym.User'),
    ),
    migrations.AddField(

```



```

        model_name='institution',
        name='type_institution',
        field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
to='tym.Type_institution'),
    ),
    migrations.AddField(
        model_name='event',
        name='institution',
        field=models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='tym.Institution'),
    ),
]

```

### Листинг 9 – файл «apps.py»:

```

from django.apps import AppConfig

class TymConfig(AppConfig):
    name = 'tym'

```

### Листинг 10 – файл «base\_generic.html»:

```

<!DOCTYPE html>
{ % load bootstrap3 % }
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    { % block title % }<title>Ticket your managment</title>{ % endblock % }
    { % bootstrap_css % }
    { % bootstrap_javascript % }
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <!-- Добавление дополнительного статического CSS файла -->
    { % load static % }
    <link rel="stylesheet" href="{ % static 'css/styles.css' % }">
</head>
<body>
    <nav class="navbar navbar-default navbar-static-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#navbar" aria-expanded="false" aria-controls="navbar">

```

```

        </button>
        <a class="navbar-brand" href="{% url 'index' %}">
            Ticket your managment</a>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li><a href="{% url 'type_institutions' %}">All type institutions</a></li>
            </ul>
        </div><!--/.nav-collapse -->
    </div>
</nav>
</nav>
<div class="container">
    <div class="page-header">
        {% block header %}{% endblock header %}
    </div>
    <div>
        {% block content %}{% endblock content %}
    </div>
</div> <!-- /container -->
</body>
</html>

```

### Листинг 11 – файл «events.html»:

```

{% extends "base_generic.html" %}
{% block content %}
    <p>Events</p>
    {% if events %}
    <ul>
        {% for event in events %}
            <li>        <a href="{% event.id %}">{{ event.name_event }}</a> ({{ event.date_added }})    </li>
            {% endfor %}
        </ul>
    {% else %}
        <p>There are no events in the institution.</p>
    {% endif %}
{% endblock %}

```

### Листинг 12 – файл «index.html»:

```

{% extends "base_generic.html" %}
{% block header %}
    <div class='jumbotron'>
        <h1>

```

```

Ticket your managment.
<p>Welocme to <em>Ticket your managment</em><p>
</h1>
</div>
{% endblock header %}
{% block content %}
<h2>Dynamic content</h2>
<p>The library has the following record counts:</p>
<ul>
<li><strong>Type institution:</strong> {{ num_type_institution }}</li>
<li><strong>Institution:</strong> {{ num_institution }}</li>
<li><strong>Events:</strong> {{ num_event }}</li>
<li><strong>Type payments:</strong> {{ num_type_payment }}</li>
</ul>
{% endblock %}

```

### Листинг 13 – файл «institutions.html»:

```

{% extends "base_generic.html" %}
{% block content %}
<p>Institutions</p>
<ul>
{% for institution in institutions %}
<li>
<a href="{{ institution.id }}">{{ institution.name_institution }}</a> ({{
institution.adress_institution }})
</li>
{% endfor %}
</ul>
{% endblock %}

```

### Листинг 14 – файл «ticket\_booked.html»:

```

{% extends "base_generic.html" %}
{% block content %}
<p>The ticket is was booked</p>
{% endblock %}

```

### Листинг 15 – файл «ticket\_details.html»:

```

{% extends "base_generic.html" %}
{% block content %}
<p>Ticket: </p>
<p><strong>Institution:</strong> <a href="/type_institutions/{{
ticket_id.event.institution.type_institution.id }}/{{ ticket_id.event.institution.id }}">{{ ticket_id.event.institution }}</a></p>
<p><strong>Event:</strong> <a href="/type_institutions/{{ ticket_id.event.institution.type_institution.id

```

```

}}/{{ ticket_id.event.institution.id }}/{{ ticket_id.event.id }}">{{ ticket_id.event }} </a></p>
    <p><strong>Type:</strong> {{ ticket_id.type_ticket }}</p>
    <p><strong>Place:</strong> {{ ticket_id.seat_number_ticket }}</p>
    <p><strong>Cost ticket:</strong> {{ ticket_id.cost_ticket }}</p>
    <p><strong>Status ticket:</strong> {{ ticket_id.status_ticket }}</p>
    <p><strong>Institution:</strong> <a href="ticket_booked/">{{ ticket_id.event.institution }} </a></p>
    {% if status_ticket.1 == ticket_id.status_ticket %}
        <form action="ticket_booked/" method="post">
            {% csrf_token %}
            <input type="submit" value="booked" />
        </form>
    {% endif %}
{% endblock %}

```

**Листинг 16 – файл «tickets.html»:**

```

{% extends "base_generic.html" %}
{% block content %}
    <p>Tickets</p>
    {% if tickets %}
        <ul>
            {% for type_ticket in type_tickets %}
                <p>{{ type_ticket }}:</p>
                {% for ticket in tickets %}
                    {% if ticket.type_ticket == type_ticket %}
                        <ul><li>
                            <a href=" {{ ticket.id }}">Place {{ ticket.seat_number_ticket }}</a> ({{
ticket.cost_ticket }}p - {{ ticket.status_ticket }})
                        </li></ul>
                    {% endif %}
                {% endfor %}
            {% endfor %}
        </ul>
    {% else %}
        <p>There are no tickets in the events.</p>
    {% endif %}
{% endblock %}

```

### Листинг 17 – файл «type\_institutions.html»:

```

{% extends "base_generic.html" %}
{% block content %}
    <p>Type institutions</p>
    <ul>
        {% for type_institution in type_institutions %}

```

```
<li>
    <a href="type_institutions/{ { type_institution.id } }/">{ { type_institution.type_institution } }</a>
</li>
{ % endfor % }

</ul>
{ % endblock % }
```

## ПРИЛОЖЕНИЕ Б

(справочное)

### Экранные формы

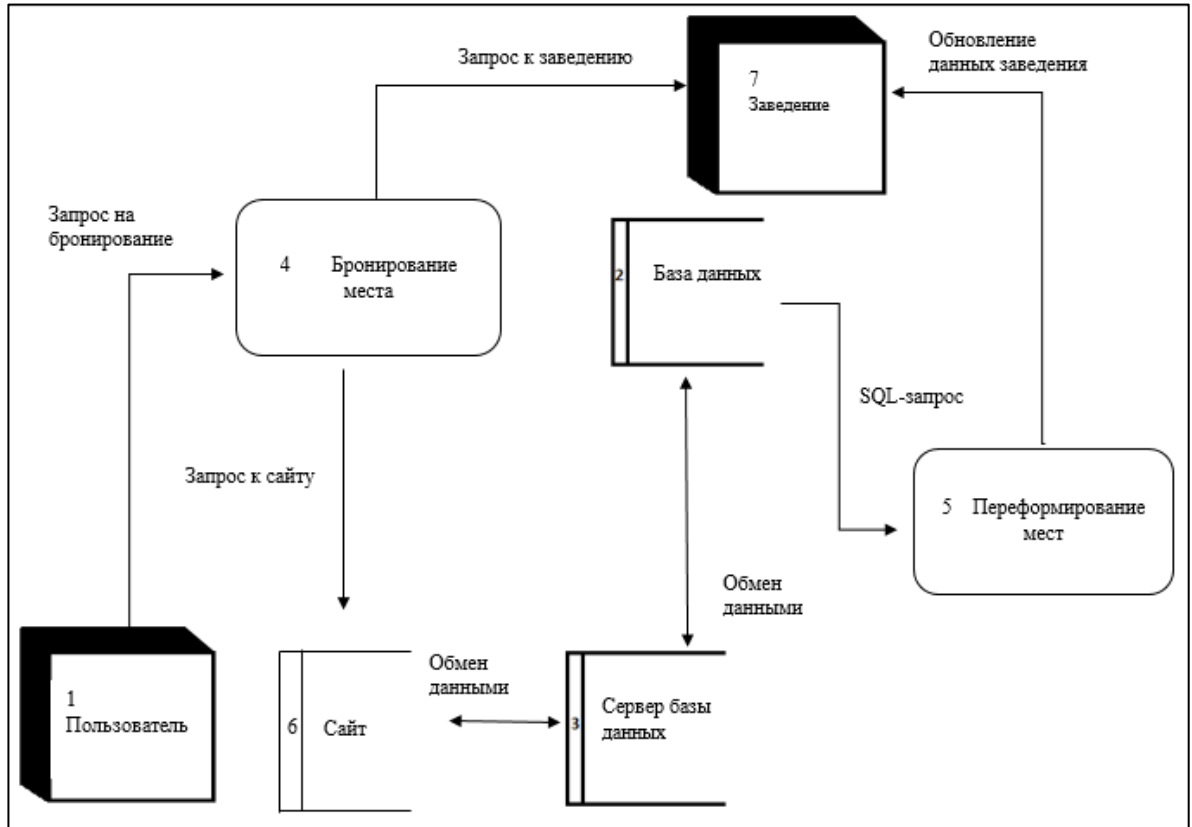


Рисунок Б.1 – Диаграмма потоков данных

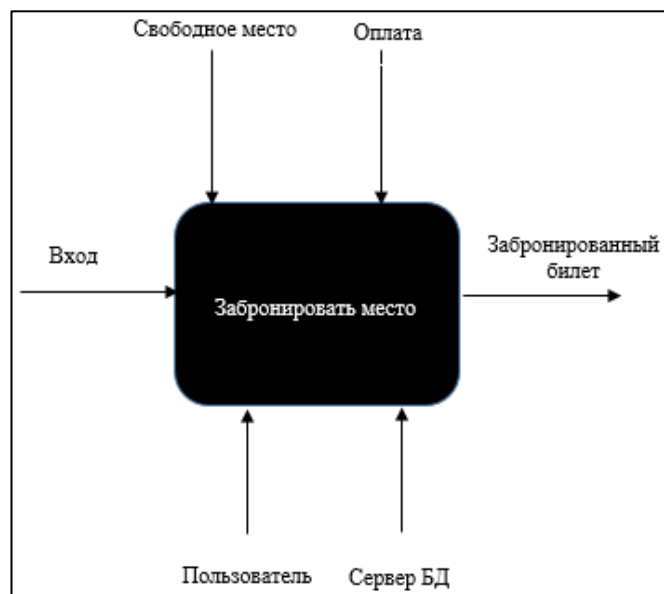


Рисунок Б.2 – Диаграмма IDEF0

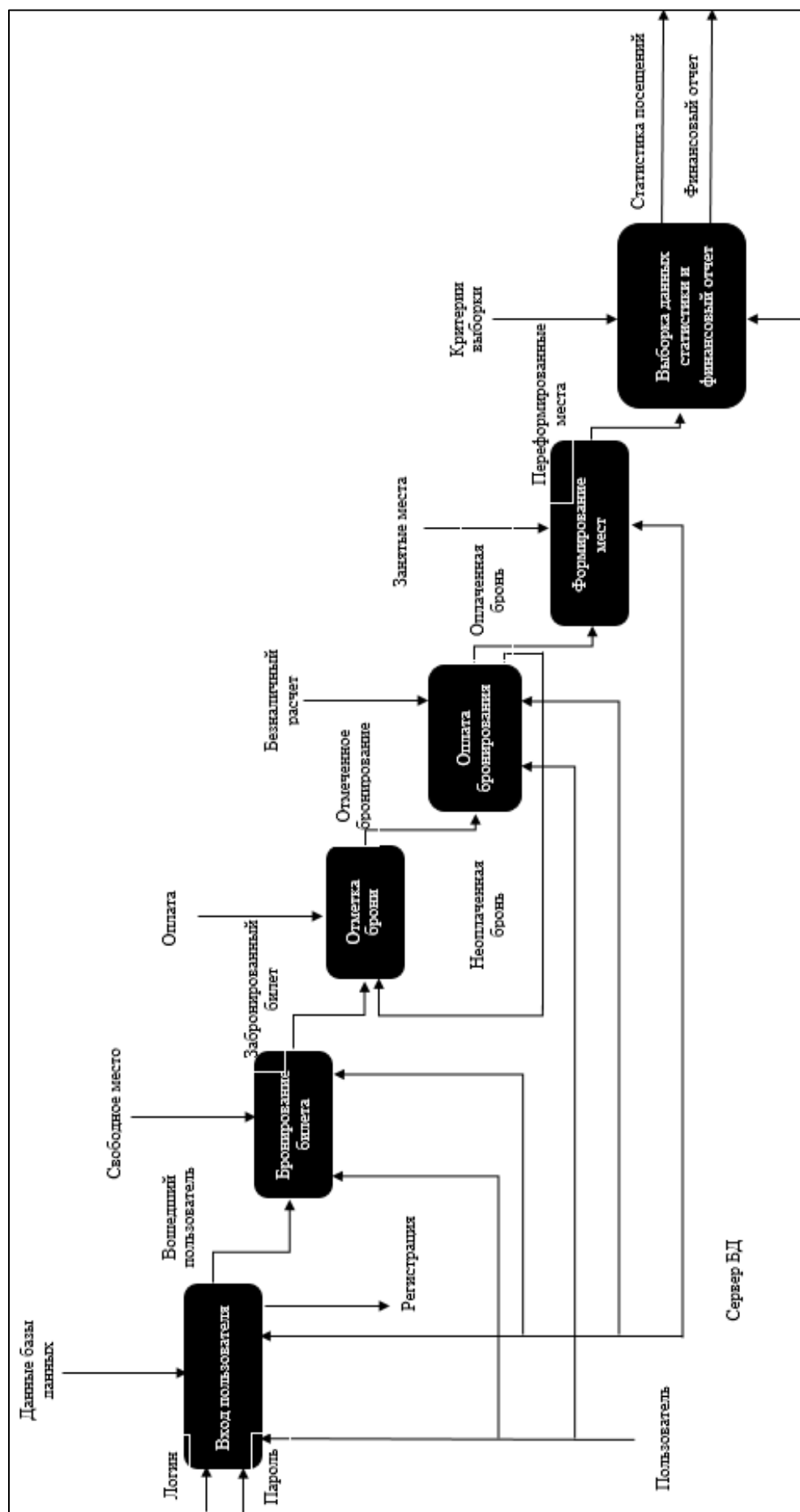


Рисунок Б.3 – Диаграмма декомпозиции IDEF0

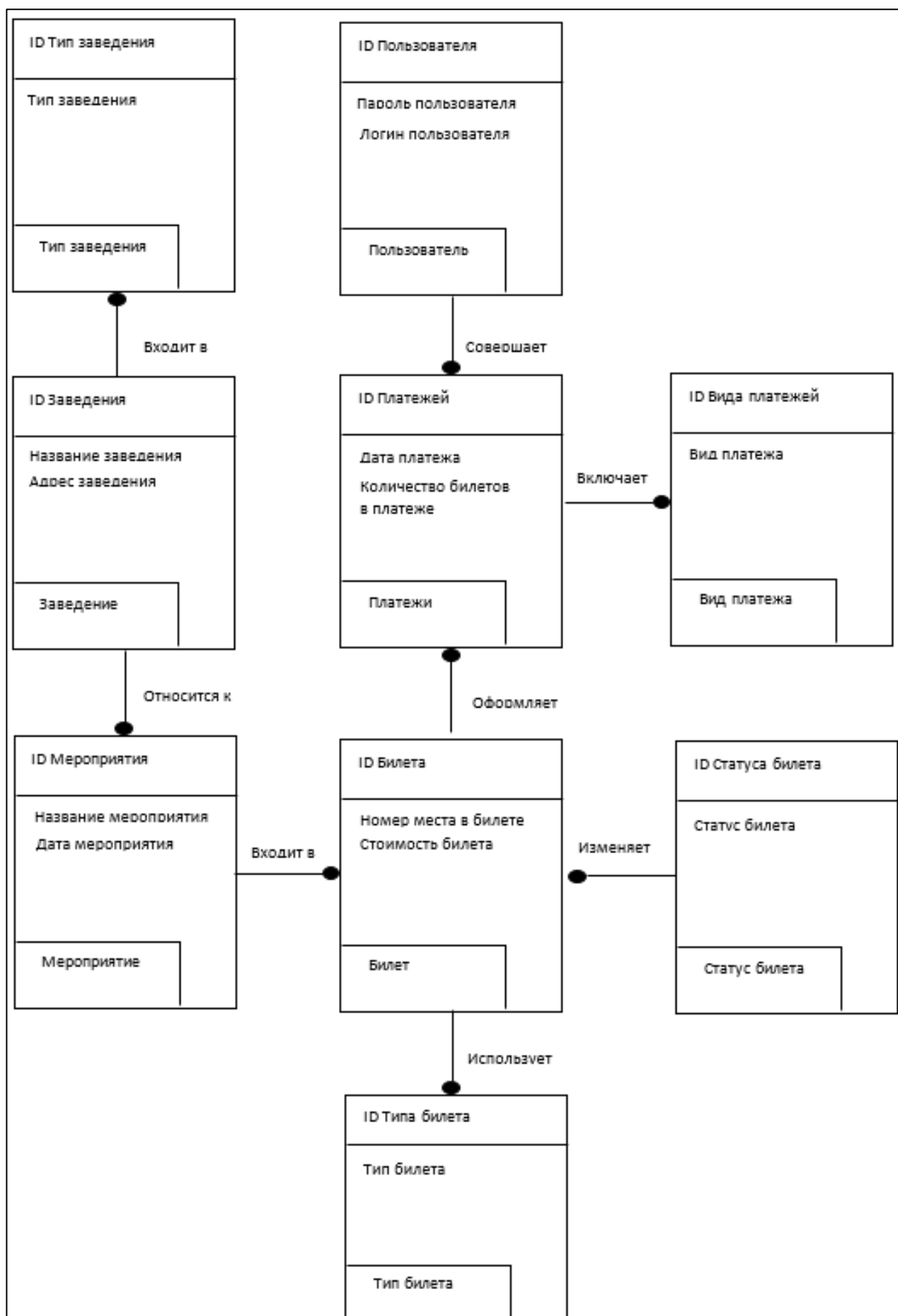


Рисунок Б.4 – Диаграмма IDEF1



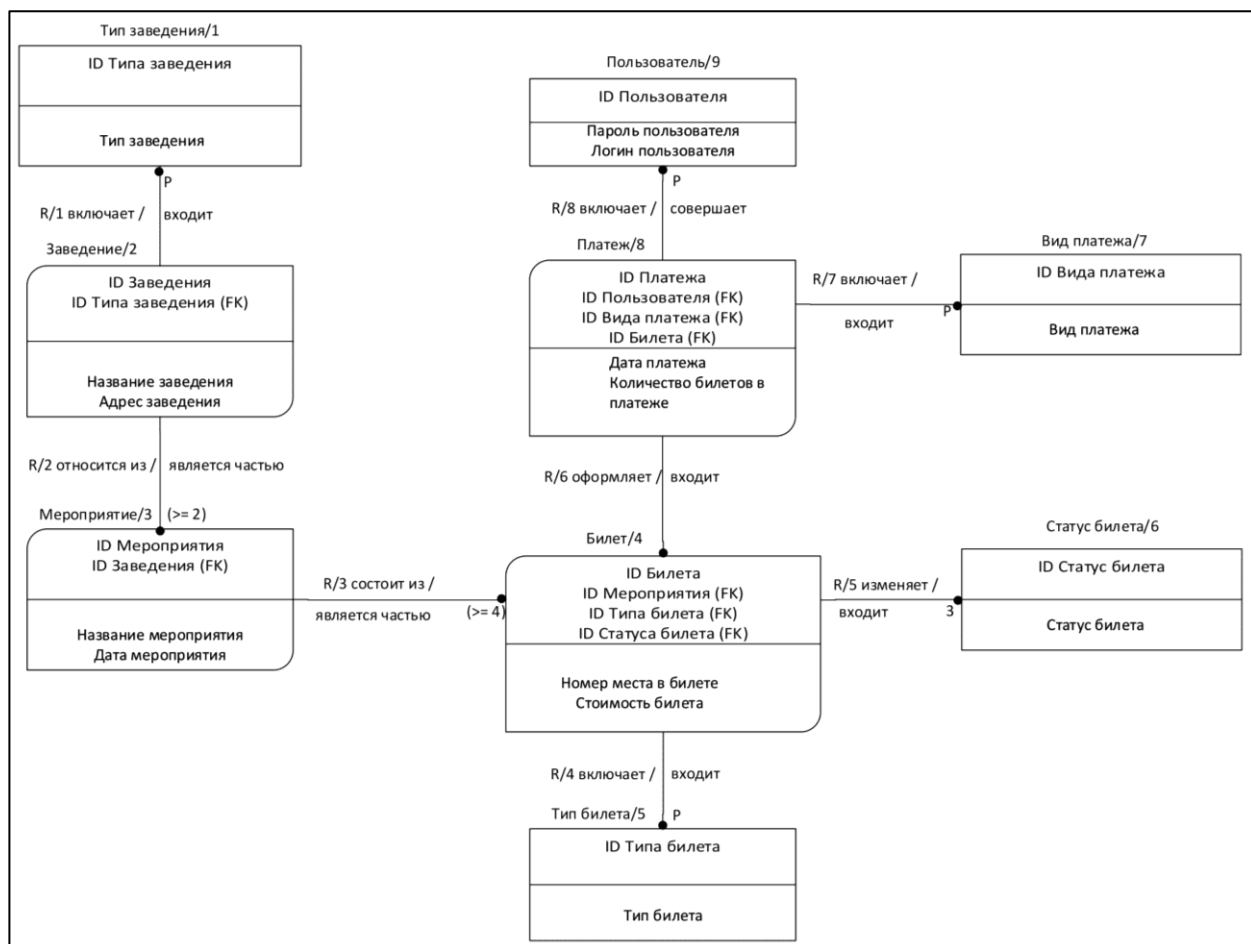


Рисунок Б.5 – Диаграмма IDEF1X

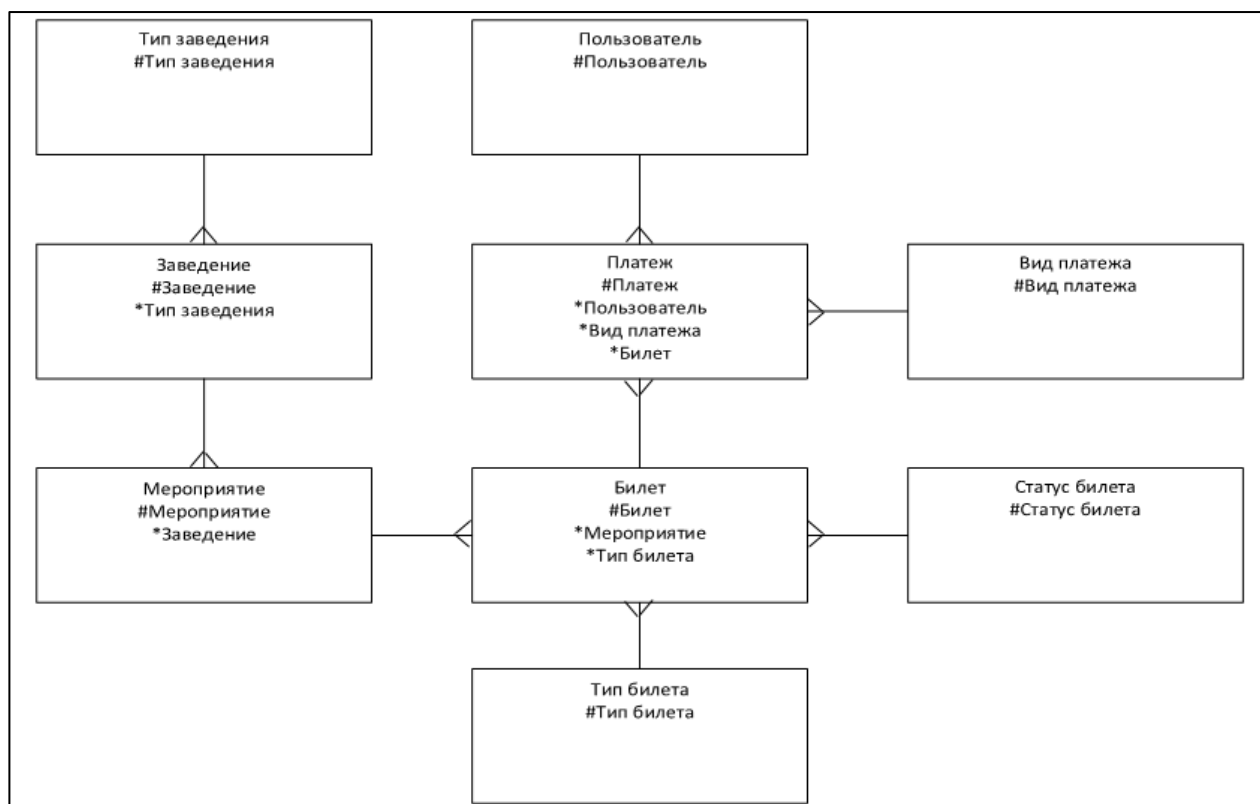


Рисунок Б.6 – ERD-диаграмма

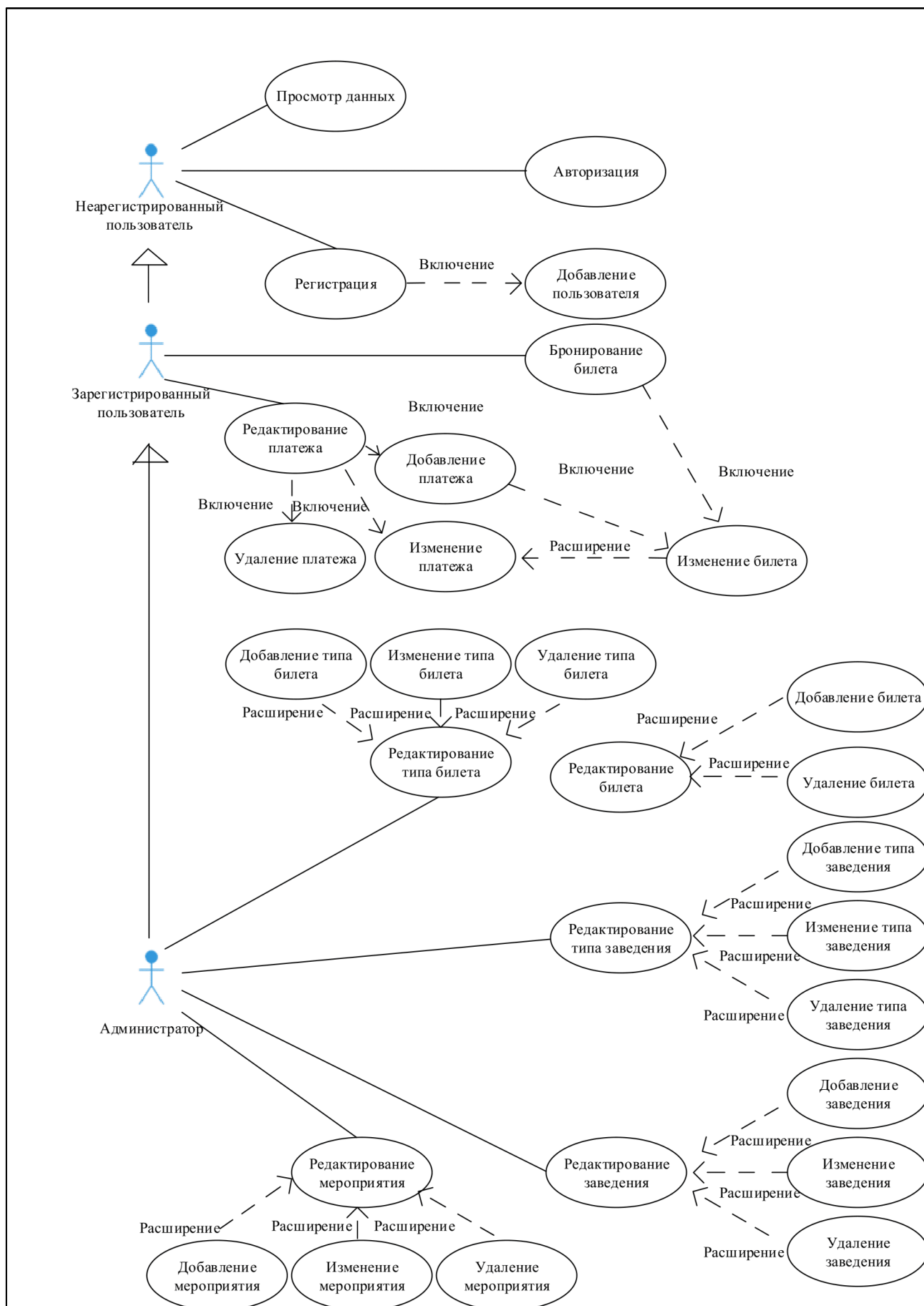


Рисунок Б.7 – Диаграмма вариантов использования

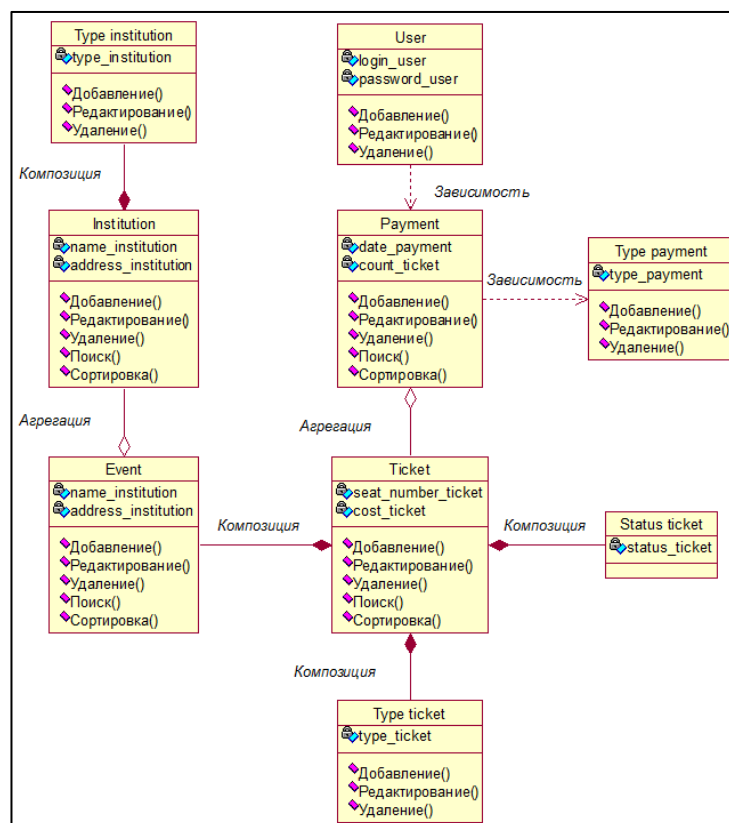


Рисунок Б.8 – Диаграмма классов

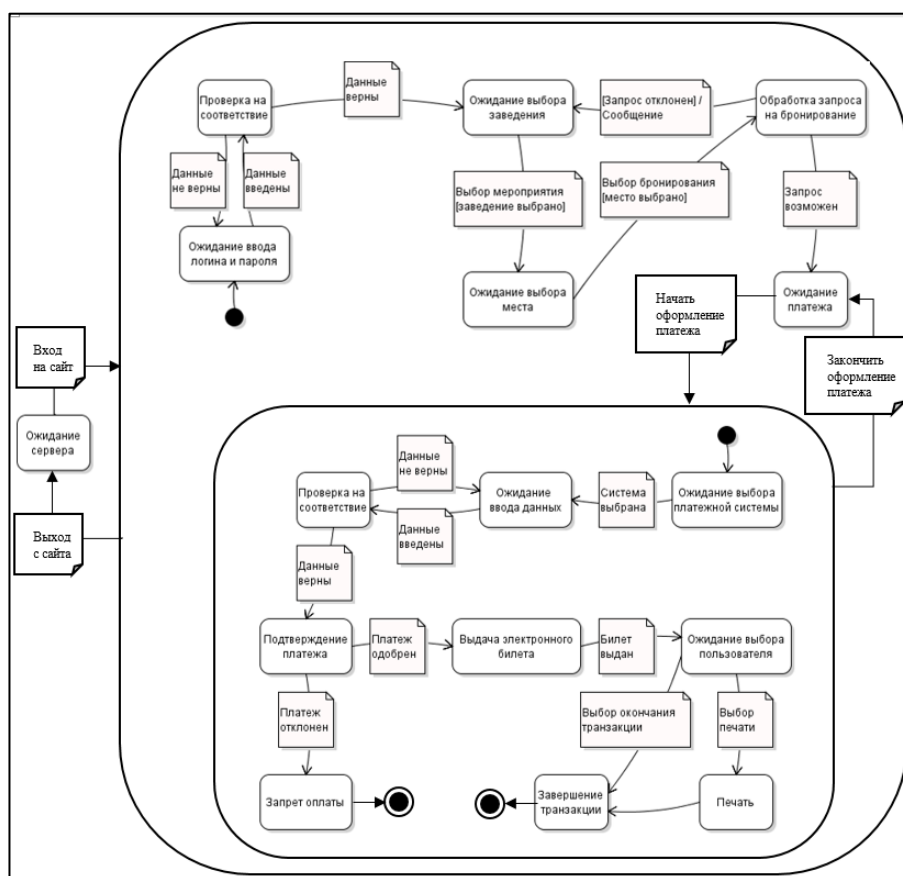


Рисунок Б.9 – Диаграмма состояний

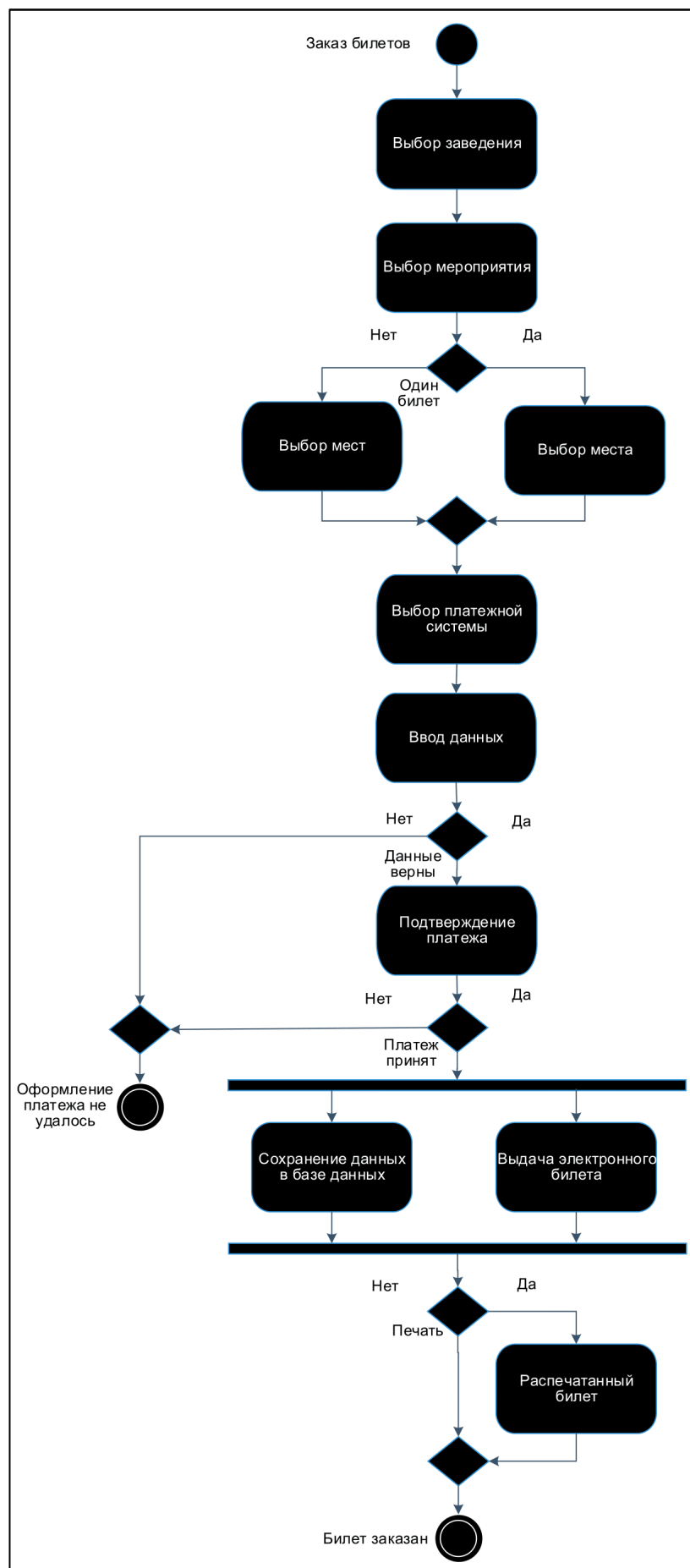


Рисунок Б.10 – Диаграмма деятельности

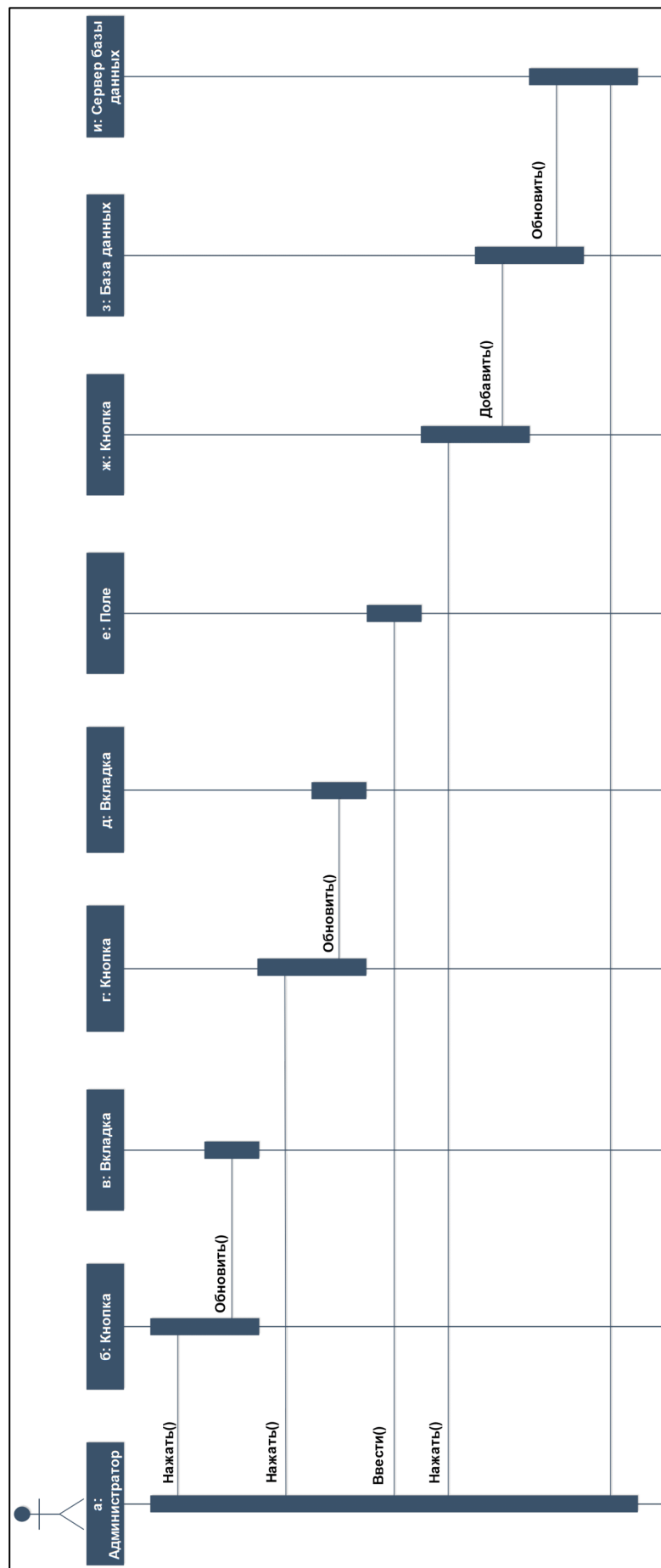


Рисунок Б.11 – Диаграмма последовательности действий



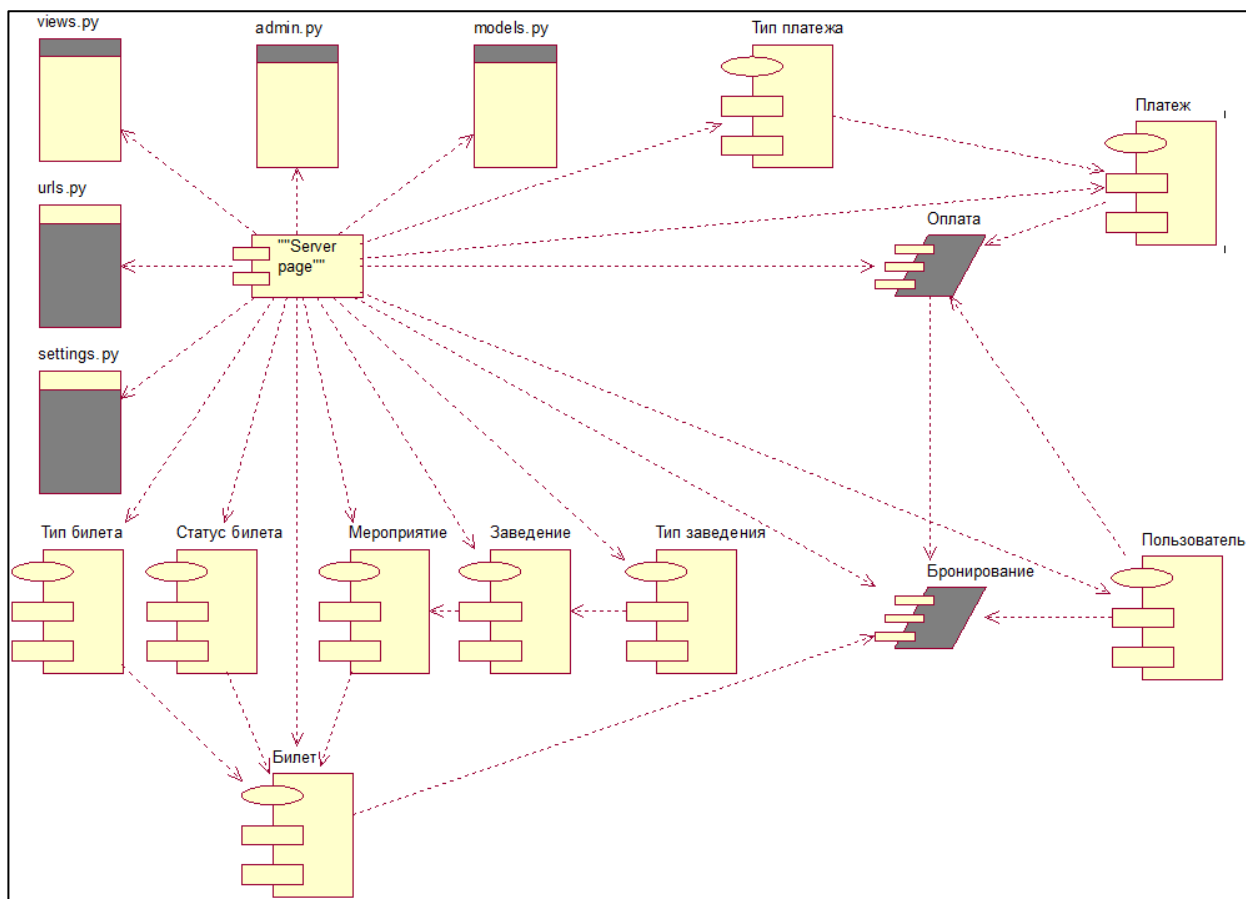


Рисунок Б.14 – Диаграмма компонент