Федеральное государственное автономное образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информационные системы»

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ П. П. Дьячук
подпись
«____» _____ 2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 — Информационные системы и технологии

Разработка серверной части для сервиса организации тематических встреч

Руководитель _____ доцент каф. ИС И. А. Легалов
подпись, дата
Выпускник _____ П. А. Петров
подпись, дата
Нормоконтролер _____ Ю. В. Шмагрис
подпись, дата

Красноярск 2019

Федеральное государственное автономное образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информационные системы»

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ П. П. Дьячук
подпись
«____» _____ 2019 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

студенту Петрову Павлу Алексеевичу

группы КИ15-13Б, направления 09.03.02 «Информационные системы и технологии».

Тема выпускной квалификационной работы «Разработка серверной части для сервиса организации тематических встреч».

Утверждена приказом по университету No7237/с от 24 мая 2019 г.

Руководитель ВКР И. А. Легалов, доцент кафедры информационных систем ИКИТ СФУ, кандидат технических наук.

Исходные данные для ВКР: задание на бакалаврскую работу.

Перечень разделов ВКР:

- введение;
- обзор предметной области;
- выводы по главе 1;
- проектирование серверной части приложения;
- выводы по главе 2;
- реализация серверной части приложения;
- выводы по главе 3;
- заключение;
- список использованных источников;
- приложения.

Перечень графического материала: презентация «Разработка серверной части для сервиса организации тематических встреч».

Руководитель ВКР _____ И. А. Легалов
подпись

Задание принял к исполнению _____ П. А. Петров
подпись

«____» _____ 2019 г.

График

выполнения выпускной квалификационной работы студентом направления 09.03.02 «Информационные системы и технологии».

График выполнения выпускной квалификационной работы приведен в таблице 1.

Таблица 1 — График выполнения выпускной квалификационной работы

Наименование этапа	Срок выполнения	Результат выполнения этапа	Примечание руководителя (отметка о выполнении этапа)
Определение темы работы	28.01.19 —10.02.19	Краткое эссе по теме ВКР	Выполнено
Сбор литературных источников	11.02.19 — 18.02.19	Список источников литературы	Выполнено
Анализ собранных литературных источников	19.02.19 — 5.03.19	Реферат о проблемно-предметной области	Выполнено
Уточнение и обоснование актуальности, цели и задач ВКР	6.03.19 — 13.03.19	Окончательная формулировка цели и задач ВКР	Выполнено
Решение первой задачи ВКР	14.03.19 — 6.04.19	Доклад и презентация по первой задаче ВКР	Выполнено
Решение второй задачи ВКР	7.04.19 — 7.05.19	Доклад и презентация по второй задаче ВКР	Выполнено
Решение третьей задачи ВКР	8.05.19 — 12.06.19	Доклад и презентация по третьей задаче ВКР	Выполнено
Подготовка доклада и презентации по теме ВКР	13.06.19 — 14.06.19	Доклад с презентацией по теме ВКР	Выполнено
Компоновка отчета по результатам решения задач ВКР	15.06.19 — 16.06.19	Отчет по результатам решения задач ВКР	Выполнено
Первичный нормоконтроль	17.06.19 — 18.06.19	Пояснительная записка, презентация к ВКР	Выполнено
Предварительная защита результатов ВКР	19.06.19 — 20.06.19	Доклад и презентация по проделанной работе	Выполнено
Вторичный нормоконтроль	21.06.19 — 24.06.19	Пояснительная записка, презентация к ВКР	Выполнено

Окончание таблицы 1

Наименование этапа	Срок выполнения	Результат выполнения этапа	Примечание руководителя (отметка о выполнении этапа)
Итоговый нормоконтроль	25.06.19 — 30.06.19	Пояснительная записка, презентация к ВКР	Выполнено
Защита ВКР	1.07.19 — 2.07.19	Доклад и презентация по результатам бакалаврской работы	Выполнено

Руководитель _____ И. А. Легалов
подпись, дата

Студент группы КИ15-13б _____ П. А. Петров
подпись, дата

РЕФЕРАТ

Выпускная квалификационная работа на тему «Разработка серверной части для сервиса организации тематических встреч» содержит 46 страниц текстового документа, 25 иллюстраций, 2 таблицы, 1 приложение, 19 использованных источников.

СЕРВЕР, ПРИЛОЖЕНИЕ, СОЦИАЛЬНЫЙ ПРОЕКТ, REST, ПРОГРАММИРОВАНИЕ, СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ, API.

Целью данной выпускной квалификационной работы является создание серверной части для сервиса организации тематических встреч.

Цель выпускной квалификационной работы достигается решением следующих задач:

— Провести обзор и сравнение существующих решений;

— Выявить требования к серверной части сервиса организации тематических встреч;

— Описать структуру и взаимосвязь компонентов серверной части сервиса организации тематических встреч;

— Реализовать серверную часть сервиса организации тематических встреч.

Результатом выпускной квалификационной работы является прототип серверной части для сервиса организации тематических встреч.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Большой популярностью пользуются разработки, направленные на организацию, улучшение либо упрощение социальных взаимодействий между людьми [1]. Существует множество проектов различной направленности — массовые социальные сети, корпоративные и личные мессенджеры, dating-приложения. Часто подобные сервисы выступают инструментом организации различных тематических встреч и мероприятий, однако ввиду широкого назначения ни один из перечисленных сервисов не включает специализированных функций для организации внесетевого досуга.

Однако существует потребность в специализированном инструменте, наличие которого позволит как оптимизировать работу организаторов, так и предоставит пользователям полную информацию о проводимых событиях. Данная работа направлена на создание инструмента организации внесетевого досуга. Система предоставит возможность пользователям как получать исчерпывающую информацию о проводимых мероприятиях, так и регистрировать собственные. В рамках данной работы будет представлен серверная часть данного сервиса, поскольку наличие универсального сервера позволит в дальнейшем организовать работу различных клиентских приложений.

Цель — разработка серверной части для сервиса организации тематических встреч.

Для достижения цели решаются следующие задачи:

– Провести обзор и сравнение существующих решений;

– Выявить требования к серверной части сервиса организации тематических встреч;

– Описать структуру и взаимосвязь компонентов сервисной части сервиса организации тематических встреч;

– Реализовать серверную часть сервиса организации тематических встреч.

1 Обзор предметной области

1.1 Обзор существующих решений

Большой популярностью пользуется разработки, направленные на организацию, улучшение либо упрощение социальных взаимодействий между людьми. Как пример можно привести многочисленные социальные сети, dating-приложения, проекты для поддержки совместной командной работы. На данный момент можно уверенно утверждать, что подобные сервисы востребованы у людей различных по роду занятий, возрасту, месту проживания, полу и т.д.

Однако в отрытых источниках не удалось обнаружить инструмента для поиска, выбора и планирования оффлайн мероприятий. Одним из предлагаемых решений подобных задач являются афиши, однако круг освещаемых ими тем недостаточно широк. Также за неимением лучшего для организации подобного досуга используются площадки социальных сетей.

Таким образом возможно выделить следующие критерии для сравнения используемых решений:

1. Информативность — возможность полного формализованного описания мероприятия по многим признакам, таким как название мероприятия, место проведения мероприятия, дата и время проведения мероприятия, тип мероприятия, контактные данные организатора мероприятия, описание мероприятия, теги;
2. Полнота — возможность для конечного пользователя получить полную информацию о всех событиях интересующего его по типу, теме, времени и месту проведения, возможность комбинированного поиска по диапазонам;
3. Доступность — возможность регистрации мероприятия пользователем;
4. Персонализация — приоритет в выдаче мероприятий, подходящих под указанные пользователем его интересы;

5. Геопозиционирование — приоритет в выдаче в порядке географической удаленности.

Для сравнительного анализа были выбраны следующие существующие решения:

1. Социальная сети (от англ. social networking sevice) — платформа, онлайн-сервис или веб-сайт, предназначенные для построения, отражения и организации социальных взаимоотношений [2]. Исходя из того, что в России крупнейшей социальной сетью является «ВКонтакте», пользователям которой доступны следующие возможности: создавать профиль с информацией о себе, управлять настройками доступа к информации на своей странице, взаимодействовать с другими пользователями приватно и публично, отслеживать через ленту новостей активность друзей и сообществ [3].

a. Представительство арт-пространства «YUSHIN BROTHERS» — публичная страница, содержащая информацию о всех проводимых в этом заведении мероприятий в виде анонсов, отчетов, фоторепортажей.

b. «кто со мной?» — группа, в которой пользователи размещают объявления, в которых инициируют набор желающих присоединиться в определенном виде досуга;

2. Афиша «kudago.com» — сервис по продвижению событий и мест, содержащий информацию о концертах, спектаклях, выставках, мастер-классах и курсах, событиях для детей, городских фестивалях. Также сервис предоставляет свободный доступ к базе с помощью API KudaGo [4].

3. Тематическое приложение «Geoscope» — сервис, предоставляющий пользователю возможность просмотреть на карте ближайших к нему зарегистрированных пользователей и отозваться на их приглашение пообщаться.

Таблица 1 — Сравнительная таблица существующих решений

	Социальные сети		Афиши	Тематические приложения
	«YUSHIN BROTHERS»	«кто со мной?»	kudago.com	«Geoscope»
Информативность	+	-	+	-
Полнота	-	-	+	-
Доступность	-	+	-	+
Персонализация	-	-	-	-
Геопозиционирование	-	-	-	+

Очевидно, что ни один из существующих сервисов не удовлетворяет заявленным критериям. Так регистрация собственных мероприятий пользователям доступна в тематических группах в социальных сетях и приложении «Geoscope», однако ни один из этих сервисов не предоставляет инструментов фильтрации и поиска мероприятия. Афиша предоставляет полную и структурированную информацию о проводимых мероприятиях, однако не дает пользователю возможности внести собственное мероприятие.

1.2 Выявление пользовательских требований

После проведенного анализа были сформулированы пользовательские требования к приложению.

1. Пользователь может зарегистрироваться и редактировать информацию о себе, которая будет использована для определения подходящих мероприятий;

2. Пользователь может просматривать список всех мероприятий, фильтровать и сортировать их по определенным параметрам;

3. Пользователь может присоединиться к интересующему его мероприятию, связаться с организатором;

4. Пользователь может зарегистрировать свое мероприятие, информативно его описать.

1.3 Выявление функциональных требований

1. Личный кабинет пользователя.

После авторизации пользователь получает доступ в личный кабинет, в котором может редактировать следующую информацию о себе: ник, город, список приоритетных тегов.

2. Лента мероприятий.

Каждое мероприятие содержит следующие поля для описания: название мероприятия, место проведения мероприятия, дата и время проведения мероприятия, тип мероприятия (спортивное мероприятие, дискуссия, просмотр, дело), контактные данные организатора мероприятия, описание мероприятия, теги. Лента мероприятий доступна всем посетителям сервиса, однако зарегистрированные пользователи могут ее фильтровать и сортировать по следующим параметрам: тип, теги, время. Также существуют автоматические механизмы для определения приоритета выдачи мероприятий зарегистрированным пользователям:

a. Приоритет в выдаче по интересам пользователя.

Имея доступ к той информации о себе, которую написал пользователь, система запускает внутренний алгоритм, позволяющий автоматически определить более интересные пользователю мероприятия и отобразить их в ленте в первую очередь;

b. Приоритет в выдаче по геопозиции пользователя.

Имея доступ к GPS, система запускает следующий алгоритм внутренний алгоритм, позволяющий автоматически определить мероприятия, которые

проводятся в определенном радиусе от пользователя, и отобразить их в ленте в первую очередь;

3. Регистрация мероприятия.

Каждый пользователь имеет возможность создания, редактирования и удаления собственного мероприятия.

Выводы по первой главе

В данной главе проведен обзор существующих решений на различных платформах, описаны критерии для анализа. Также представлена таблица с результатами сравнения, наглядно иллюстрирующая, что ни одно из существующих решений полностью не удовлетворяет заявленным критериям.

На основании анализа были сформулированы пользовательские и функциональные требования. Необходимо уточнить, что описанные в них функции являются широкоиспользуемыми, однако ни одно приложение не использует полную их совокупность.

В результате была выполнена первая задача — выявление и анализ требований к серверной части сервиса для организации тематических встреч.

2 Проектирование серверной части приложения

Проектирование — процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или ее части. Результатом проектирования является проект — целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы.

2.1 Диаграмма вариантов использования

Диаграмма вариантов использования (usecase) предназначена для наглядной демонстрации возможных сценариев взаимодействия с системой [5]. Актером (субъектом, осуществляющим взаимодействие с системой) является зарегистрированный пользователь, сценарием использования, в свою очередь, является набор действий, совершение которых в приложении доступно зарегистрированному пользователю. Подробное описание действий находится в пункте 1.3.

На рисунке 1 изображена диаграмма использования для разрабатываемого приложения.

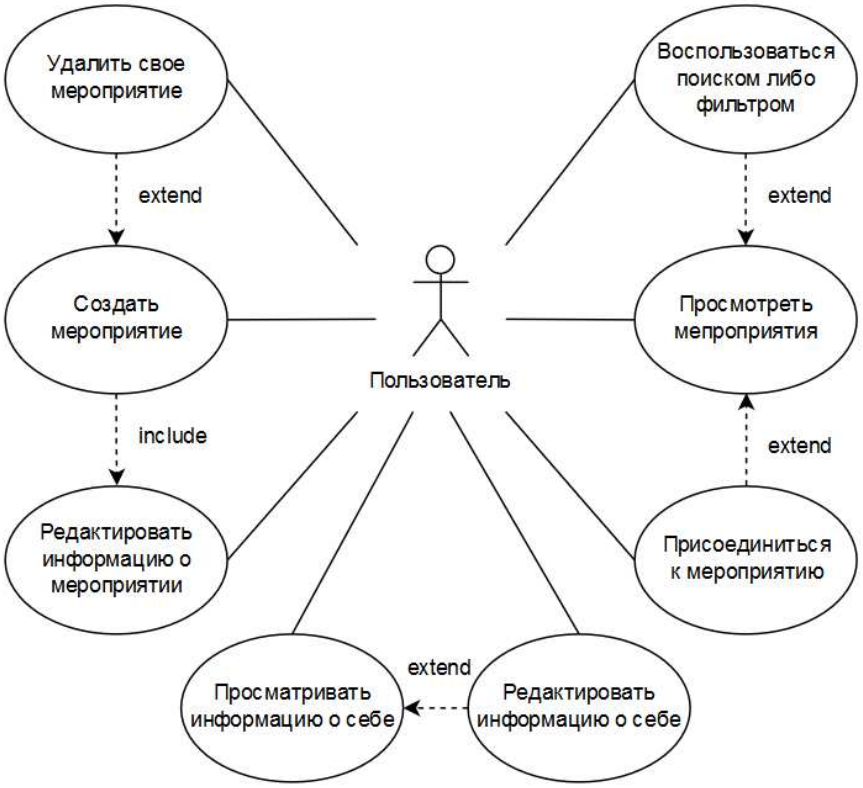


Рисунок 1 — Диаграмма вариантов использования приложения

2.2 Архитектура приложения

Из всего разнообразия архитектурных стилей для разработки веб-приложения наиболее подходящими являются клиент-серверная и многоуровневая архитектура [6]. Основными преимуществами архитектурного паттерна клиент/сервер являются:

1. Большая безопасность. Все данные хранятся на сервере, который обычно обеспечивает больший контроль безопасности, чем клиентские компьютеры.

2. Централизованный доступ к данным. Поскольку данные хранятся только на сервере, администрирование доступа к данным намного проще, чем в любых других архитектурных стилях.

3. Простота обслуживания. Роли и ответственность вычислительной системы распределены между несколькими серверами, общающимися друг с

другом по сети. Благодаря этому клиент гарантированно остается неосведомленным и не подверженным влиянию событий, происходящих с сервером (ремонт, обновление либо перемещение).

На рисунке 2 показана схема взаимодействия компонентов в клиент-серверной архитектуре.

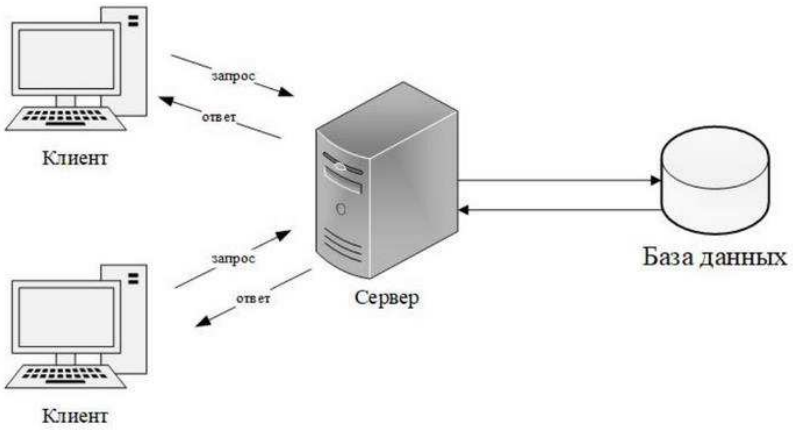


Рисунок 2 — Клиент-серверная архитектура

Использование такого типа архитектуры дает возможность уменьшить размер и сложность программ, перенести наиболее трудоемкие операции на сервер и уменьшить объем информации, передаваемой по сети. Все это улучшает производительность системы.

Тем не менее, традиционная архитектура клиент/сервер имеет множество недостатков, включая тесное связывание данных и бизнес-логики приложения на сервере, что может иметь негативное влияние на расширяемость и масштабируемость системы, и зависимость от центрального сервера, что негативно сказывается на надежности системы. Для решения этих проблем архитектурный стиль клиент/сервер был развит в более универсальный многоуровневый, в котором устранены некоторые недостатки, свойственные 2-х уровневой архитектуре клиент/сервер, и обеспечиваются дополнительные преимущества:

1. Удобство поддержки. Уровни не зависят друг от друга, что позволяет выполнять обновления или изменения, не оказывая влияния на приложение в целом.

2. Масштабируемость. Уровни организовываются на основании развертывания слоев, поэтому масштабировать приложение довольно просто.

3. Гибкость. Управление и масштабирование каждого уровня может выполняться независимо, что обеспечивает повышение гибкости.

4. Доступность. Приложения могут использовать модульную архитектуру, которая позволяет использовать в системе легко масштабируемые компоненты, что повышает доступность.

Исходя из этого, был выбран многоуровневый архитектурный стиль проектирования.

Одной из разновидностей многоуровневого стиля проектирования является шаблон MVC (Model-View-Controller), состоящий из трех слоев: модель, отвечающая за логику приложения, представление, отвечающая за вывод информации на экран, и контроллер, связующее звено между представлением и моделью [7]. Схема взаимодействия компонентов представлена на рисунке 3.

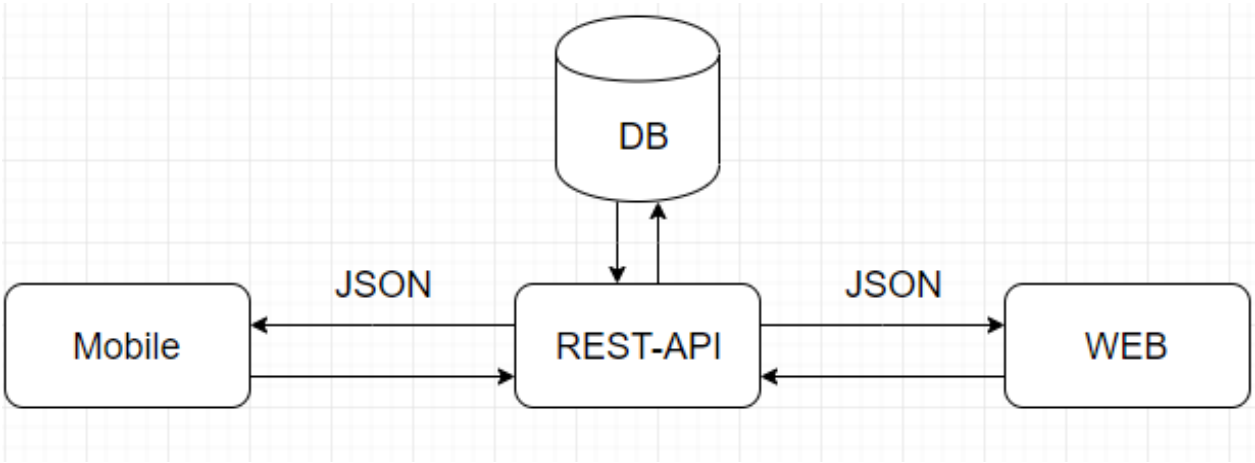


Рисунок 3 —Взаимодействие компонентов MVC

Алгоритм взаимодействия клиента и сервера:

1. Клиентское приложение формирует и отправляет HTTP-запрос.

2. Запрос доходит до веб-сервера и перенаправляется на механизм маршрутизации.

3. На основе URL-адреса, механизм маршрутизации выбирает соответствующий контроллер.

4. Контроллер обращается к базе данных, используя модель для получения данных.

5. Контроллер вызывает механизм просмотра и возвращает представление страниц.

6. Запрошенный ресурс отправляется обратно в браузер.

2.3 Выбор системы управления базой данных

Выбор системы управления баз данных представляет собой сложную многопараметрическую задачу. При попытке создать реляционную модель данных стало очевидным, что данный подход имеет не является оптимальным для отражения данных предметной области — таблицы вынуждены были содержать множество избыточных полей, как, например, параметры, которые необходимо учитывать при регистрации различных мероприятий. На основании этого было принято решение использовать нереляционную базу данных.

MongoDB — документо-ориентированная СУБД, информация в которой хранится в виде JSON-документов [8].

К ключевым преимуществам использования нереляционной базы данных *MongoDB* относится:

1. Возможность использования смешанного подхода — СУБД *MongoDB* не обязывает использовать ссылочную связность для всех объектов, не предоставляет такую возможность для разработчика;

2. Динамическая структура данных — СУБД *MongoDB* предоставляет возможность создавать документы, не зная их структуру заранее, что позволяет создавать объекты только с необходимой и достаточной информацией о них;

3. Данные в документе хранятся в порядке их внесения.

Использование нереляционной базы данных требует уточнения следующего аспекта: данные в документо-ориентированных базах данных могут быть дублированы. Хранение одинаковых данных в разных коллекциях снижает затраты времени на доступ к этим данным, так как нужно обратиться только к одному документу вместо формирования запроса по нескольким документам с помощью механизма ссылочной связи. Но также это усложняет операции записи по отношению к этим данным: при обновлении либо удалении данных изменения необходимо внести в каждый документ, который содержит эти данные. Решение о применении дублирования информации принимается в зависимости от частоты выполнения запросов на чтение и запись, а также критичности задачи обеспечения согласованности. Также необходимо отметить, что несмотря на то, что использование нереляционной базы данных не предполагает обеспечение целостности и связности данных, эти задачи выполняются на уровне приложения и ORM.

Таким образом база данных хранит коллекцию из двух документов — «Мероприятия» и «Пользователи». Каждый из этих документов хранит информацию по заданной схеме и может содержать вложенные документы как содержимое своих полей. Схема документо-ориентированной базы данных представлена на рисунке 4.

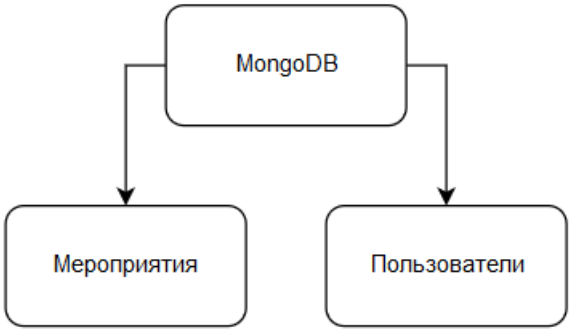


Рисунок 4 — Схема документо-ориентированной базы данных

В документе «Мероприятия» содержится информация о проводимых мероприятиях — название, время и дата проведения, место проведения, тип мероприятия, контактные данные организатора мероприятия, описание мероприятия, теги. Поскольку предполагается, что сервис работает как агрегатор, то часть записей в данном документе будет загружена из открытых источников посредством API либо другими способами.

В документе «Пользователи» содержится информация о зарегистрированных пользователях сервиса — ник, город, список приоритетных тегов, пароль.

2.4 Инструменты и среда разработки

Язык программирования — Python

Python — язык программирования общего назначения, архитектура которого предполагает использование таких механизмов как как динамическая типизация, автоматическое управление памятью, механизм обработки исключений, поддержку многопоточных вычислений, высокоуровневые структуры данных [9].

Использование языка Python имеет следующие преимущества:

1. Python является продуктом с открытым исходным кодом, использование которого допустимо в том числе в коммерческих целях;
2. Python поддерживает пакетную дистрибуцию, возможно использование в разработке только необходимых пакетов;

3. Существует большое количество разнообразных библиотек и готовых приложений, что существенно сокращает время разработки;

4. Интерпретатор CPython имеет совместимость с большинством существующих платформ;

5. Существует подробная документация, в том числе и на русском языке.

Среда разработки — PyCharm

PyCharm — это интеллектуальная Python IDE с полным набором средств для эффективной разработки на языке Python. Выпускается в двух вариантах — бесплатная версия PyCharm Community Edition и поддерживающая больший набор возможностей PyCharm Professional Edition [10].

К ключевым возможностям данного инструмента относится:

– Функциональный редактор кода с подсветкой синтаксиса, авто-форматированием и авто-отступами для поддерживаемых языков.

– Простая навигация в коде.

– Помощь при написании кода, включающая в себя автодополнение, авто-импорт, шаблоны кода, проверка на совместимость версии интерпретатора языка, и многое другое.

– Быстрый просмотр документации для любого элемента в окне редактора, просмотр внешней документации через браузер, поддержка docstring-генерация, подсветка, автодополнение и многое другое.

– Большое количество инспекций кода.

– Рефакторинг кода, который предоставляет широкие возможности по выполнению быстрых глобальных изменений в проекте.

Выводы по второй главе

Возможности и поведение системы наглядно отражены на диаграмме вариантов использования. Определены действующие лица приложения —

пользователи. Установлены типы отношений между вариантами использования.

В ходе проектирования приложения была выбрана СУБД *MongoDB* и описана модель документо-ориентированной базы данных.

Описана трехзвенная архитектура разрабатываемого приложения, а также обозначены средства его разработки — язык программирования Python, база данных *MongoDB.*

Задача проектирования серверной части сервиса для организации тематических встреч выполнена в полной мере.

3 Реализация серверной части приложения

В настоящий момент успех реализации во многом зависит от особенностей используемого инструмента. Для решения стандартных задач хорошим тоном считается использование проверенного кода, покрытого тестами. Главная задача разработчика в подобном контексте состоит в корректной компоновке элементов, создания инфраструктуры и описания алгоритма создания необходимых функций.

Representational State Transfer (REST) — это согласованный набор архитектурных принципов для создания масштабируемых и гибких сетей. Эти принципы описывают компоненты системы, их взаимодействие друг с другом, возможности масштабирования сетей и их различного конфигурирования. Система, отвечающая принципам REST, называется RESTful-системой, и должна удовлетворять следующим ограничениям:

1. Архитектура клиент/сервер. RESTful-система должна производить операции в клиент-серверной модели, даже если компонент в разные моменты времени ведёт себя то как клиент, то как сервер;

2. Отсутствие состояния. Отсутствие состояния в данном случае обозначает отсутствие необходимости для сервера и клиента отслеживать состояния друг друга. Когда клиент не взаимодействует с сервером, сервер не имеет представления о его существовании. Сервер также не ведёт учет прошлых запросов. Каждый запрос рассматривается как самостоятельный;

3. Единообразие интерфейса, гарантирующее, что между серверами и клиентами существует общий язык, который позволяет каждой части быть заменяемой или изменяемой без нарушения целостности системы. Это возможно при соблюдении следующих ограничений:

a. Идентификация ресурсов — каждый ресурс должен быть уникально обозначен постоянным идентификатором. Web использует URI для идентификации ресурсов, а HTTP — в качестве стандарта коммуникации.

Чтобы получить ресурс, хранящийся на сервере, клиент делает к URI HTTP-GET-запрос, который идентифицирует этот ресурс;

b. Управление ресурсами с помощью представлений — клиент управляет ресурсами, направляя серверу представления, обычно в виде JSON-объекта, содержащего контент, который он хотел бы добавить, удалить или изменить [11]. В REST сервер имеет полный контроль над ресурсами, и отвечает за любые изменения. В случае, если клиент хочет внести изменения в ресурсы, он отправляет серверу представление того, каким он видит итоговый ресурс, сервер принимает данный запрос как предложение, контроль ресурсов остается за ним;

c. Полные сообщения — содержат всю информацию, которая необходима для понимания его получателем. В отдельной документации или другом сообщении не должно быть дополнительной информации;

d. Гипермедиа — в RESTful-системах сервер может отправлять клиентам только гипермедиа — данные, которые содержат информацию о том, что клиенту нужно делать дальше, другими словами, какие еще запросы он может сделать;

4. Кэширование — ответы сервера должны помечаться как кэшируемые или некэшируемые. Кэширование происходит, когда клиент сохраняет ответы, полученные ранее от сервера. Когда эти данные нужны снова, кэширование может избавить от полного прохода данных по сети;

5. Система слоёв — RESTful-система может иметь большее количество компонентов, чем клиент и сервер, при этом каждый компонент ограничен способностью видеть только соседний слой и взаимодействовать только с ним;

6. Код по требованию — единственное опциональное ограничение, которое предполагает отправку сервером исполняемого кода клиенту. Для примера можно привести HTML-тег <script> — когда HTML-документ загружается, браузер автоматически выбирает на сервере JavaScript и исполняет его локально.

В проекте на языке программирования Python можно добавить и использовать специализированную библиотеку *Eve*, которая позволяет быстро разрабатывать API-приложения и включает в себя сервер, способный принимать и отвечать на HTTP-запросы клиента [12]. Также данная библиотека нативно работает с *MongoDB*, используя для этого библиотеку *PyMongo* из своего списка зависимостей. Также в списке зависимостей библиотеки *Eve* числятся пакеты:

– Events — класс, который заключает в себе ядро для подписки на события и запуска событий;

– Cerberus — легкая и расширяемая библиотека для валидации данных;

– Simplejson — простой и расширяемый JSON кодировщик, написанный на языке Python без каких-либо зависимостей, но включающий в себя дополнительное расширение, написанное на языке C, для значительного увеличения скорости работы;

– Werkzeug — комплексная библиотека веб-приложений, использующих стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером под названием Web Server Gateway Interface (WSGI) [13];

– Flask — облегченная среда веб-приложений WSGI, предназначенная для быстрого и легкого начала работы с последующей возможностью масштабирования до сложных приложений, а также предоставляющая множество расширений, облегчающих добавление новых функций. В список зависимостей фреймворка Flask входят следующие библиотеки: click, itsdangerous, Jinja2.

Для создания проекта с использованием библиотеки Eve необходимо создать главный исполняемый python-файл с произвольным именем (в данном проекте — run.py), и файл настроек в той же директории с названием settings.py. Содержимое файла run.py представлено на рисунке 5.

```
import os
from eve import Eve

# Heroku support: bind to PORT if defined, otherwise default to 5000.
if 'PORT' in os.environ:
    port = int(os.environ.get('PORT'))
    # use '0.0.0.0' to ensure your REST API is reachable from all your
    # network (and not only your computer).
    host = '0.0.0.0'
else:
    port = 5000
    host = '127.0.0.1'

app = Eve()

if __name__ == '__main__':
    app.run(host=host, port=port)
```

Рисунок 5 — Фрагмент кода файла run.py

Файл settings.py определяет конфигурацию проекта с помощью таких параметров, как: строка подключения к базе данных, перечисление разрешенных для вызова клиентом запросов к ресурсу (resourse) и экземпляру (item), описание схемы доменных объектов в формате JSON Schema и другие параметры конфигурации сервера. Также возникла необходимость определить значение ключей X_DOMAINS и X_HEADERS таким образом, чтобы разрешить совместное использование ресурсов между разными источниками. Данная технология называется cross origin resource sharing (CORS) и использование ее в данном проекте обусловлено тем, что заранее неизвестен источник тестовых запросов, которые будет отправлять разработчик клиентской части. На рисунке 6 представлен фрагмент кода файла settings.py.

Для непосредственного создания сервера в файл run.py необходимо импортировать класс Eve из библиотеки *Eve*, создать объект импортированного класса и вызвать у него метод run, передав в качестве аргументов адрес для размещения сервера. В данном проекте в качестве адреса для размещения сервера указан 0.0.0.0, что позволит серверу прослушивать все сетевые карты компьютера, на котором запущен сервер. Вторым аргументом передается номер порта для прослушивания сервером. По

умолчанию используется порт 5000, однако также можно указать произвольный свободный порт. В данном проекте номер порта был подставлен из переменной окружения с именем PORT так как это является одним из требований для использования хостинга *Heroku* [14].

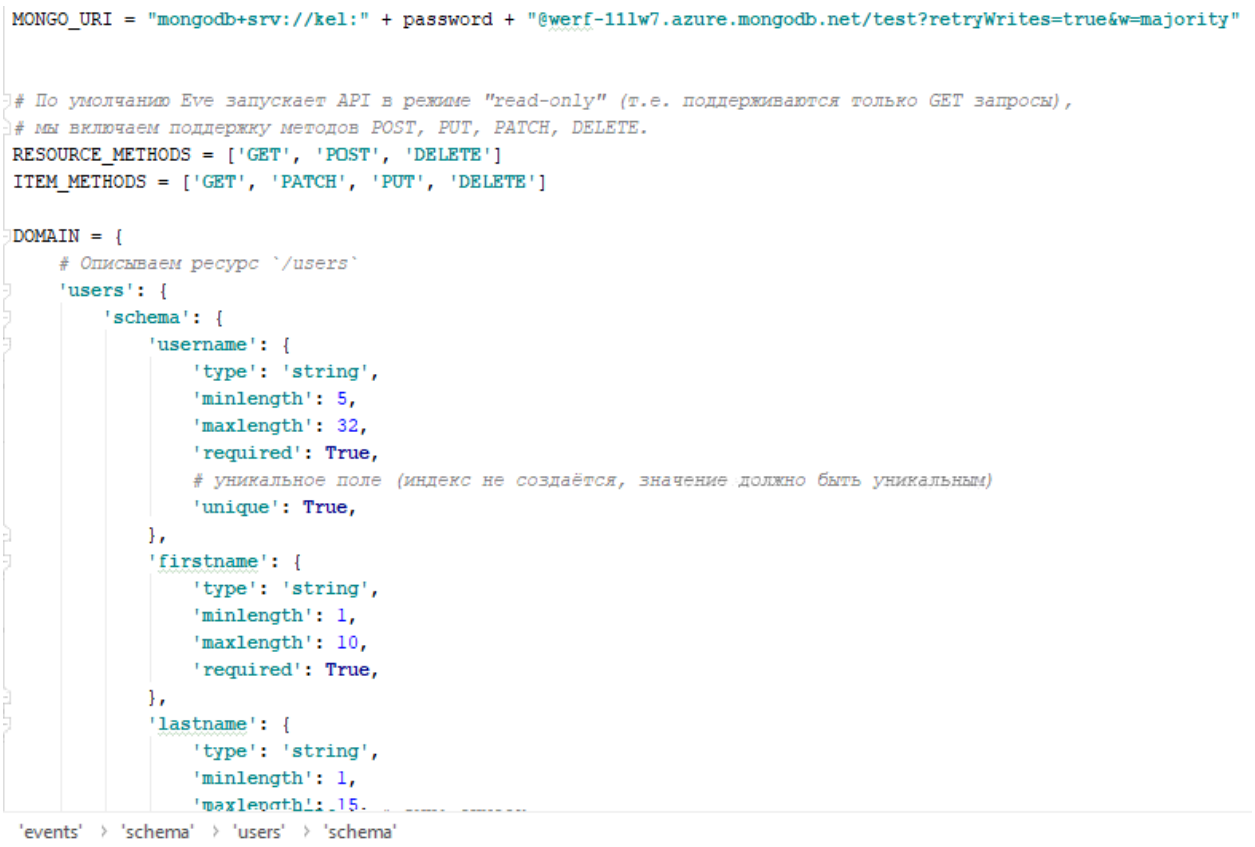


Рисунок 6 — Фрагмент кода файла settings.py

Хостинг *Heroku* позволяет бесплатно размещать приложения с ограничением по времени активности. Развертывание происходит путем отправки файлов исходного кода и конфигурации проекта с помощью системы контроля версий Git в специально предоставленный репозиторий. Далее контейнер на удаленном сервере установит необходимые для использования кода проекта зависимости, описанные выше, и запустит проект на выполнение в соответствии с инструкциями, содержащимися в обязательном файле Procfile, использующемся *Heroku* как указание каким образом и с какой точки проекта начинать выполнение. В нашем случае Procfile содержит строку "web:

python run.py": web указывает что проект является сетевым, соответственно нуждается в доступе к сетевому стеку и назначении входящего порта, эти ресурсы будут предоставлены *Heroku*. После двоеточия указывается команда, которую нужно запустить для старта проекта, в данном случае выполнение файла run.py с помощью интерпретатора Python.

Heroku предоставляет инструмент командной строки, с помощью которого производится создание, обслуживание и другие операции управления размещёнными проектами. Для размещения проекта необходимо вызвать команду «Heroku start» находясь в директории проекта с созданным локальным Git-репозиторием. Этой командой для проекта будет создан контейнер на удалённом сервере *Heroku* и адрес его репозитория для принятия кода будет установлен в качестве удалённого репозитория. После формирования коммита командой «git commit» (предполагается, что файлы проекта были на этот момент уже были добавлены для отслеживания командой «git add») проект нужно отправить командой «git push heroku master». После этого он будет передан и размещён на удалённом сервере. Процесс размещения включает в себя валидацию проекта на соответствие техническим требованиям платформы, установку необходимых пакетов, перечисленных в файле requirements.txt и запуск согласно инструкциям из Procfile. Также использование Git-репозиториев обеспечивает контроль версий разрабатываемых приложений [15]. Следить за работой и состоянием сервера можно с помощью записей в журнале событий, вызываемом при помощи команды «heroku logs», а отслеживание работы сервера в режиме реального времени возможно вызовом той же команды с указание ключа «tail». Фрагмент вывода записей из журнала событий в консоль приведен на рисунке 7.

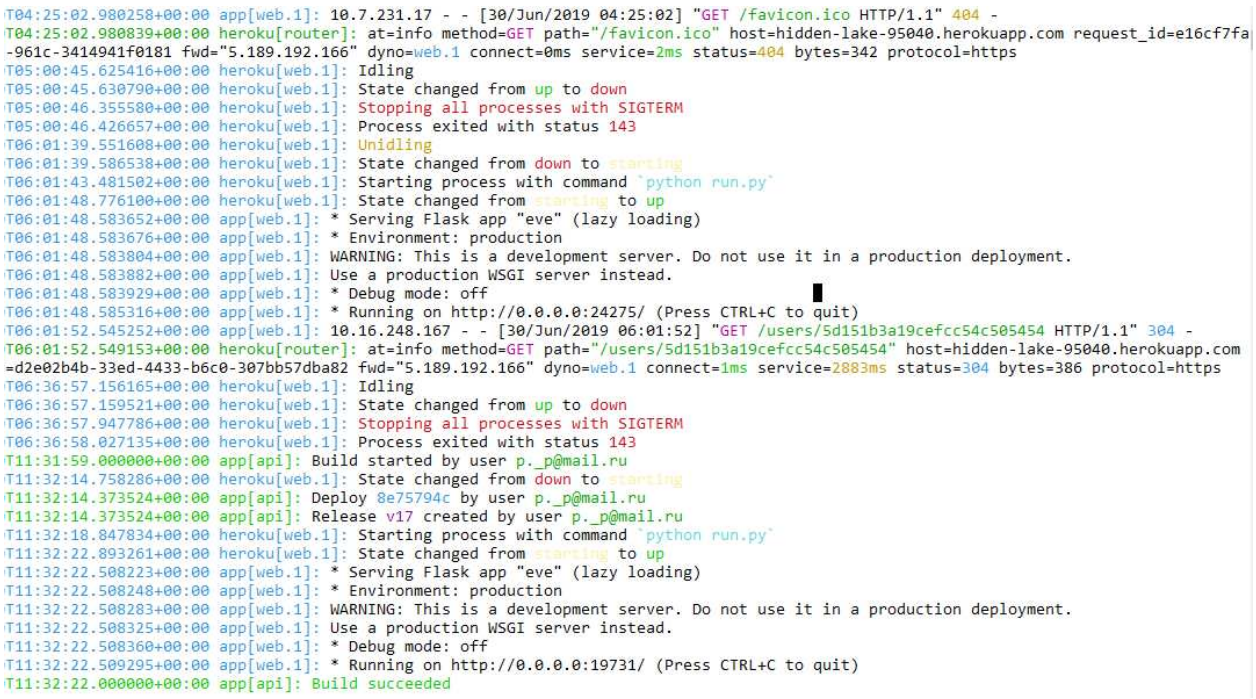


Рисунок 7 — Фрагмент вывода записей журнала событий

В качестве строки подключения к базе данных использован URI специальной формы, позволяющий приложению взаимодействие с экземпляром *MongoDB*, размещённым на облачном хостинге cloud.mongodb.com, который позволяет бесплатно размещать базу объёмом до 512 мегабайт [16]. Для создания и настройки его была выполнена регистрация на сайте cloud.mongodb.com. Для начала работы с базой данных необходимо создать пользователя базы данных. В данном проекте этот пользователь имеет полные права. Интерфейс администратора базы данных для создания пользователя показан на рисунке 8.

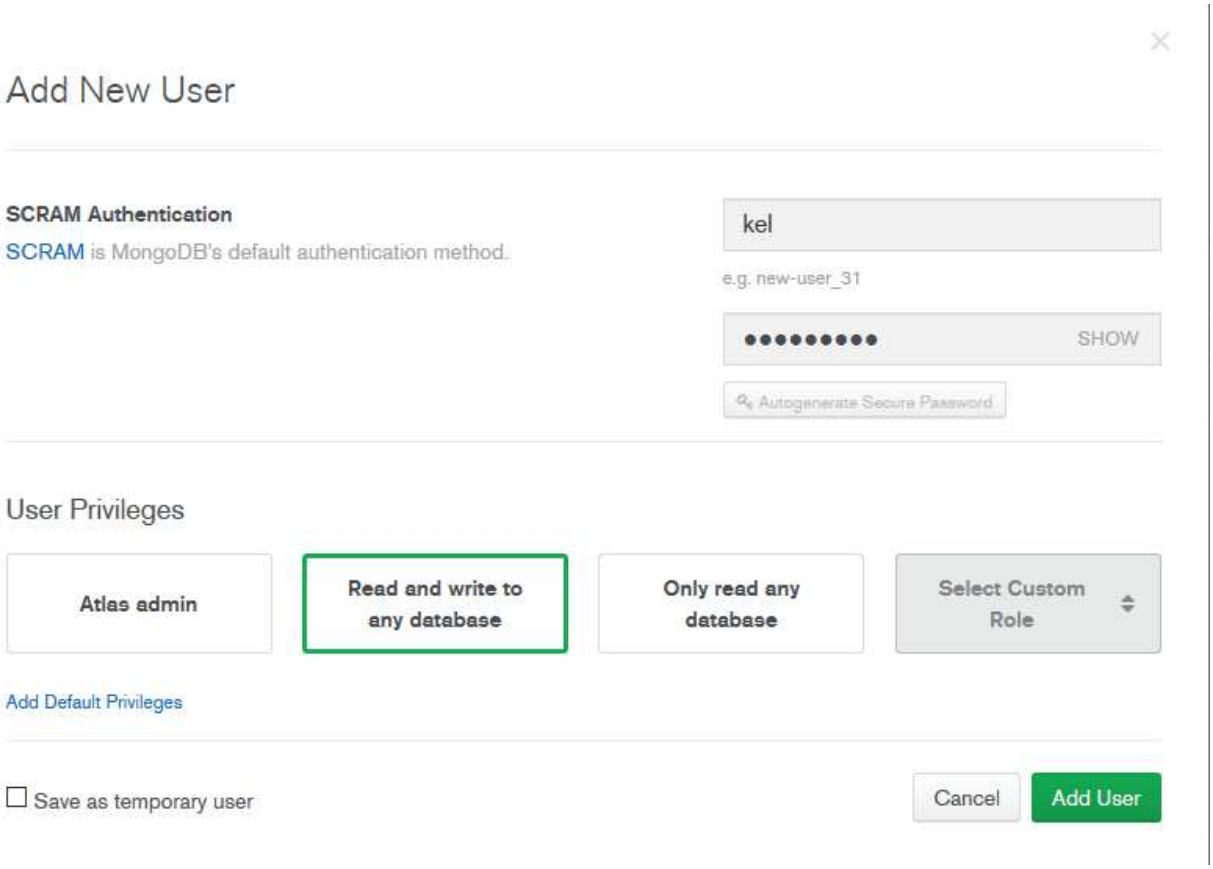


Рисунок 8 — Интерфейс добавления пользователя базы данных

Учетные данные пользователя (логин и пароль) указываются в строке подключения к базе данных. В качестве меры безопасности на сервисе необходимо задать «белый список» IP-адресов или подсетей, запросы с которых будут приняты и обработаны. Поскольку на стадии разработки неизвестен конечный адрес размещения сервера, был разрешён доступ с любого адреса при помощи записи 0.0.0.0/0 — это продемонстрировано на рисунке 9. Также безопасность будет обеспечена на уровне пользователя базы данных.

Рисунок 9 — Страница управления списком IP-адресов

MongoDB, в том числе экземпляры на cloud.mongo.com также поддерживают:

– разделение прав доступа пользователей к операциям с базой данных;

– горизонтальное масштабирование с использованием техники сегментирования (*sharding*) объектов баз данных — распределение их частей по различным узлам кластера;

– реплицирование — работа с набором реплик, то есть, содержать две или более копии данных на различных узлах.

Дополнительным преимуществом использования такой связки технологий является отсутствие необходимости преобразования данных ввиду использования единого стандарта JSON (Javascript object notation) на всех этапах работы с данными, таких как хранение в документах *MongoDB*, обработка на сервере и обмен между клиентом и сервером.

Для проверки работоспособности API необходимо эмулировать отправку и получение HTTP-запросов на адрес сервера. Для этого используем пакет *HTTPie*, являющийся простой в использовании альтернативой общепринятому консольному HTTP-клиенту *Curl* и поддерживающий работу

с данными формата JSON [17]. Ответом на запрос является HTTP-response от сервера, содержащий HTTP-код результата выполнения операции, некоторые заголовки и запрашиваемые данные.

В таблице 2 показано что API может обеспечить полный набор CRUD-операций для работы с записями. Один API можно использовать как для получения, так и для обновления. В таблице показана схема используемой библиотекой *Eve* реализации CRUD-операций в формате REST. В первом столбце содержится название действия, во втором вид отправляемого HTTP-запроса, в третьем — контекст для выполнения этого действия. Контекст в данном случае означает вид ресурса, на который должен указывать адрес, куда будет отправлен запрос.

Таблица 2 — Схема реализации CRUD-операций в библиотеке *Eve*

Action	**HTTP Verb**	**Context**
Create	POST	Collection
Create	PUT	Document
Replace	PUT	Document
Read	GET, HEAD	Collection/Document
Update	PATCH	Document
Delete	DELETE	Collection/Document

HTTP-response (ответы сервера на запрос) делятся на виды в зависимости от результат выполнения запроса [18]:

1. Информационный код состояния HTTP сервера — все коды состояний, начинающиеся с цифры 1: говорят клиенту о том, что их запрос получен и находится в обработке;

2. Успешный код состояния HTTP сервера — все коды состояний, начинающиеся цифры 2: говорят клиенту о том, что действие закончилось успешно: получено, понято и обработано;

3. Код перенаправления HTTP сервера — все коды состояния, начинающиеся цифры 3: говорят клиенту о том, что для продолжения работы ему нужно совершить какие-либо действия;

4. Коды ошибок HTTP клиента — все коды состояния, начинающиеся цифры 4: говорят клиенту о том, что ошибка происходит по его вине (неверный синтаксис, устаревший протокол и прочее), из-за чего сервер не может корректно отработать и дать ответ.

5. Коды ошибок HTTP сервера — все коды состояний, начинающиеся с цифры 5: говорят клиенту о том, что произошла ошибка на стороне сервера HTTP.

На рисунке 10 представлен результат выполнения GET-запроса на корневой адрес сервера.

```
(venv) C:\Users\User\PycharmProjects\1>http https://hidden-lake-95040.herokuapp.com/
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 99
Content-Type: application/json
Date: Thu, 27 Jun 2019 19:17:17 GMT
Server: Eve/0.9.2 Werkzeug/0.15.4 Python/3.6.8
Via: 1.1 vegur

{
    "_links": {
        "child": [
            {
                "href": "users",
                "title": "users"
            },
            {
                "href": "events",
                "title": "events"
            }
        ]
    }
}
```

Рисунок 10 — Результат выполнения GET-запроса к серверу

Благодаря поддержке принципа HATEOAS (Hypermedia as the Engine of Application State) — «гипермедиа как движок состояния приложения» — каждый ответ на GET-запрос содержит секцию _links, содержащую ссылки и заголовки доступных для запроса ресурсов в зависимости от запрошенного ресурса. Эта информация может быть использована клиентским приложением для динамического обновления пользовательского интерфейса или же для взаимодействия с API в условиях неизвестной заранее его структуры. В данном случае представлены ссылки на ресурсы users/ и events/ с аналогичными заголовками. На рисунке 11 представлен результат GET-запроса к ресурсу users.

```
(venv) C:\Users\User\PycharmProjects\l1>http https://hidden-lake-95040.herokuapp.com/users
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 166
Content-Type: application/json
Date: Thu, 27 Jun 2019 19:27:19 GMT
Server: Eve/0.9.2 Werkzeug/0.15.4 Python/3.6.8
Via: 1.1 vegur
X-Total-Count: 0

{
    "_items": [],
    "_links": {
        "parent": {
            "href": "/",
            "title": "home"
        },
        "self": {
            "href": "users",
            "title": "users"
        }
    },
    "_meta": {
        "max_results": 25,
        "page": 1,
        "total": 0
    }
}
```

Рисунок 11 — Результат выполнения GET-запроса к ресурсу users

Поле X-Total-Count отображает общее количество возвращённых записей, на момент выполнения коллекция пуста.

На рисунке 12 представлен результат выполнения запроса на добавление пользователя в коллекцию.

```
(venv) C:\Users\User\PycharmProjects\1>http https://hidden-lake-95040.herokuapp.com/users username=UserName1 firstname=Vasiliy lastname=Petrovich born="Thu, 27 Aug 1970 14:37:13 GMT"
HTTP/1.1 201 CREATED
Connection: keep-alive
Content-Length: 276
Content-Type: application/json
Date: Thu, 27 Jun 2019 19:38:34 GMT
Location: http://hidden-lake-95040.herokuapp.com/users/5d151b3a19cefcc54c505454
Server: Eve/0.9.2 Werkzeug/0.15.4 Python/3.6.8
Via: 1.1 vegur

{
    "_created": "Thu, 27 Jun 2019 19:38:33 GMT",
    "_etag": "ad1c3f8eelef3d2262d49531653467db458446f1",
    "_id": "5d151b3a19cefcc54c505454",
    "_links": {
        "self": {
            "href": "users/5d151b3a19cefcc54c505454",
            "title": "User"
        }
    },
    "_status": "OK",
    "_updated": "Thu, 27 Jun 2019 19:38:33 GMT"
}
```

Рисунок 12 — Успешное добавление пользователя

При наличии в теле запроса данных, HTTPie по умолчанию сформирует POST-запрос для отправки их на сервер. Ответ содержит строку HTTP/1.1 201 CREATED, что говорит об успешном добавлении записи в коллекцию.

На рисунке 13 представлен результат выполнения запроса на добавление в коллекцию пользователя с некорректно установленными данными: не соответствующими формату передачи или установленным на сервере ограничениям на содержание поля.

```
(venv) C:\Users\User\PycharmProjects\1>http https://hidden-lake-95040.herokuapp.com/users username=UserName1 firstname=Vasiliy lastname=Petrovich born=04.12.1970
HTTP/1.1 422 UNPROCESSABLE ENTITY
Connection: keep-alive
Content-Length: 159
Content-Type: application/json
Date: Thu, 27 Jun 2019 19:34:09 GMT
Server: Eve/0.9.2 Werkzeug/0.15.4 Python/3.6.8
Via: 1.1 vegur

{
    "_error": {
        "code": 422,
        "message": "Insertion failure: 1 document(s) contain(s) error(s)"
    },
    "_issues": {
        "born": "must be of datetime type"
    },
    "_status": "ERR"
}
```

Рисунок 13 — Отклоненное добавление пользователя

Ответ содержит строку HTTP/1.1 422 UNPROCESSABLE ENTITY, что говорит об отклонении запроса ввиду невозможности добавлении записи в коллекцию. Сервер автоматически выполнил валидацию поступающих данных и не внес некорректные записи.

На рисунке 14 представлен фрагмент результата выполнения запроса на вывод заполненной коллекции пользователей.

```
(venv) C:\Users\User\PycharmProjects\l1>http https://hidden-lake-95040.herokuapp.com/users
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 1792
Content-Type: application/json
Date: Thu, 27 Jun 2019 20:29:27 GMT
Last-Modified: Thu, 27 Jun 2019 20:28:36 GMT
Server: Eve/0.9.2 Werkzeug/0.15.4 Python/3.6.8
Via: 1.1 vegur
X-Total-Count: 4
```

Рисунок 14 — Фрагмент результата выполнения запроса на вывод заполненной коллекции пользователей

По значению заголовка X-Total-Count можно определить количество записей в данной коллекции.

Для демонстрации возможности фильтрации записей передадим запрос на получение записей о всех пользователях с именем Павел. На рисунке 15 представлен фрагмент результат выполнения этого запроса.

```
(venv) C:\Users\User\PycharmProjects\l1>http https://hidden-lake-95040.herokuapp.com/users?where=firstname=="Pavel"
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 996
Content-Type: application/json
Date: Thu, 27 Jun 2019 20:45:50 GMT
Last-Modified: Thu, 27 Jun 2019 20:34:08 GMT
Server: Eve/0.9.2 Werkzeug/0.15.4 Python/3.6.8
Via: 1.1 vegur
X-Total-Count: 2
```

Рисунок 15 — Фрагмент результат выполнения запроса на вывод всех пользователей с именем Павел

По заголовку X-Total-Count можно увидеть, что был выполнен отбор и возвращены 2 записи, также в ответ от сервера входит полное содержание этих записей, не представленное на рисунке 15. Помимо поиска(отбора) сервер поддерживает запросы для работы с записями:

– вывод в отсортированном по какому-либо полю порядке;
– перезапись имеющихся;
– частичное обновление записей;
– вывод любой информации в формате XML вместо JSON [19];
– удаление;
– постраничная выдача результатов запроса;
– запросы выполняемые при выполнении заданных условий;
– вывод информации о вложенных записях.

После дополнительной настройки сервера станет возможна аутентификация с возможностью простой интеграции аутентификации по механизму OAuth2. Возможно разграничить доступ к использованию некоторых запросов, целого API, или некоторых операций из набора CRUD. Также возможно управлять правами доступа пользователей по назначенным им ролям.

Выводы по третьей главе

В данной главе описаны действия для создания и настройки сервера для дальнейшей работы. Рассмотрен набор принципов взаимодействия сервера и клиента REST, включая принцип HATEOAS, необходимые для разработки библиотеки, фреймворки, подключаемые классы. Описан выбранный для размещения проекта хостинг и способ взаимодействия и управления им. Предложены возможности сервера и перспективы добавления нового функционала, описаны примеры взаимодействия сервера с клиентом.

ЗАКЛЮЧЕНИЕ

Обзор и сравнение существующих решений на предмет наличия функций планирования и организации внесетевых мероприятий позволил выявить пользовательские и функциональные требования, необходимые для полноценной работы сервиса. Результат проведенного сравнения наглядно представлен в таблице.

При помощи визуального моделирования были созданы такие представления системы, как диаграмма вариантов использования сервиса и схема документно-ориентированной базы данных. Первая диаграмма наглядно иллюстрирует выявленные функциональные требования, схема данных дает представление о том, как предполагается работа с данными в выбранной нереляционной базе данных.

Описана трехзвенная архитектура разрабатываемого приложения, а также обозначены средства его разработки — язык программирования Python, библиотека Eve, база данных *MongoDB*, хостинг-провайдер Heroku, а также необходимые для разработки библиотеки, фреймворки, подключаемые классы. Рассмотрен набор принципов взаимодействия сервера и клиента REST, включая принцип HATEOAS.

Описаны действия для создания и настройки сервера для дальнейшей работы. Описан выбранный для размещения проекта хостинг и способ взаимодействия и управления им, а также создание и конфигурация базы данных. Описаны возможности сервера и перспективы добавления нового функционала, описаны примеры взаимодействия сервера с клиентом, валидации данных,

Таким образом выполнены все задачи по созданию серверной части приложения организации тематических встреч, а цель — достигнута.

В качестве возможного улучшения работы можно предложить: накопление данных о мероприятиях с использование внешних источников,

таких как открытый API рассмотренного сервиса «KudaGo», возможность аутентификации различными методами с разделением доступа к ресурсам.

СПИСОК СОКРАЩЕНИЙ

API — Application programming interface;

CORS — Cross-origin resource sharing;

CRUD — Create, read, update, delete;

DB — Data base;

GPS — Global Positioning System;

HATEOAS — Hypermedia as the Engine of Application State;

HTTP — Hyper Text Transfer Protocol;

IDE — Integrated Development Environment;

JSON — JavaScript Object Notation;

MVC — Model view controller;

REST — Representational State Transfer;

URL — Uniform Resource Locator;

WSGI — Web Server Gateway Interface;

XML — eXtensible Markup Language;

ИС — Информационная система;

БД — База данных;

СУБД — Система управления баз данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт компании Smart Insights [Электронный ресурс] // https://www.smartinsights.com Компания производитель «Smart Insights». – Режим доступа: https://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/.

2. Пурдехнад Джон Открытые инновации и социальные сети // ПУСС. 2012. №7. URL: https://cyberleninka.ru/article/n/otkrytye-innovatsii-i-sotsialnye-seti (дата обращения: 4.05.2019).

3. Вебер Ксения Сергеевна, Пименова Анастасия Александровна Сравнительный анализ социальных сетей // Вестник Тамбовского университета. Серия: Естественные и технические науки. 2014. №2.

4. Социальный новостной портал KudaGo [Электронный ресурс] // https://kudago.com Компания производитель «KudaGo». – Режим доступа: https://kudago.com/pages/public-api/opisanie-api/.

5. Роберт, А. Максимчук UML для простых смертных / Роберт А. Максимчук, Эрик Дж. Нейбург. - Москва: СИНТЕГ, 2014. - 272 с.

6. Эрик Фримен, Элизабет Фримен, Кэтти Сьерра, Берт Бейтс Head First. Паттерны проектирования. Обновленное юбилейное издание Санкт-Петербург: Питер, 2018. - 656 с.

7. Информационный IT-портал [Электронный ресурс] // https://dou.ua Компания производитель «Dou». – Режим доступа: https://dou.ua/lenta/articles/php-vs-hacklang/.

8. Официальный сайт продукта MongoDB [Электронный ресурс] // https://www.mongodb.com Компания производитель «MongoDB, Inc.». – Режим доступа: https://www.mongodb.com/what-is-mongodb/.

9. Официальный сайт продукта Python [Электронный ресурс] // https://www.python.org Компания производитель «Python Software Foundation». – Режим доступа: https://www.python.org/doc/.

10. Официальный сайт продукта PyCharm [Электронный ресурс] // https://jetbrains.ru/ Компания производитель «JetBrains». – Режим доступа: https://jetbrains.ru/products/pycharm/.

11. Официальный сайт продукта JSON [Электронный ресурс] // https://json.org Компания производитель «Ecma International». – Режим доступа: http://json.org/json-ru.html.

12. Официальный сайт продукта EVE [Электронный ресурс] // https://docs.python-eve.org/ Разработчик «Nicola Iaroccі». – Режим доступа: https://docs.python-eve.org/en/stable/.

13. Официальный сайт продукта Python [Электронный ресурс] // https://www.python.org Компания производитель «Python Software Foundation». – Режим доступа: https://www.python.org/dev/peps/pep-3333/.

14. Официальный сайт продукта Heroku [Электронный ресурс] // https://www.heroku.com/ Компания производитель «Salesforce». – Режим доступа: https://help.heroku.com/.

15. Официальный сайт продукта Git [Электронный ресурс] // https://git-scm.com/ Компания производитель «Software Freedom Conservancy». – Режим доступа: https://git-scm.com/doc.

16. Информационный IT-портал [Электронный ресурс] // https://www.mongodb.com Компания производитель «MongoDB, Inc.». – Режим доступа: https://docs.atlas.mongodb.com/getting-started/.

17. Официальный сайт продукта HTTPie [Электронный ресурс] // https://httpie.org/ Разработчик «Jakub Roztočil». – Режим доступа: https://httpie.org/doc.

18. Ковалёв Сергей Сергеевич, Шишаев Максим Геннадьевич Современные методы защиты от нежелательных почтовых рассылок // Труды Кольского научного центра РАН. 2011. №7. URL: https://cyberleninka.ru/article/n/sovremennye-metody-zaschity-ot-nezhelatelnyh-pochtovyh-rassylok.

19. Миронов В. В., Шакирова Г. Р. Концепция динамических XML-документов // Вестник УГАТУ. 2006. №5. URL: https://cyberleninka.ru/article/n/kontseptsiya-dinamicheskih-xml-dokumentov.

ПРИЛОЖЕНИЕ А

Плакаты презентации

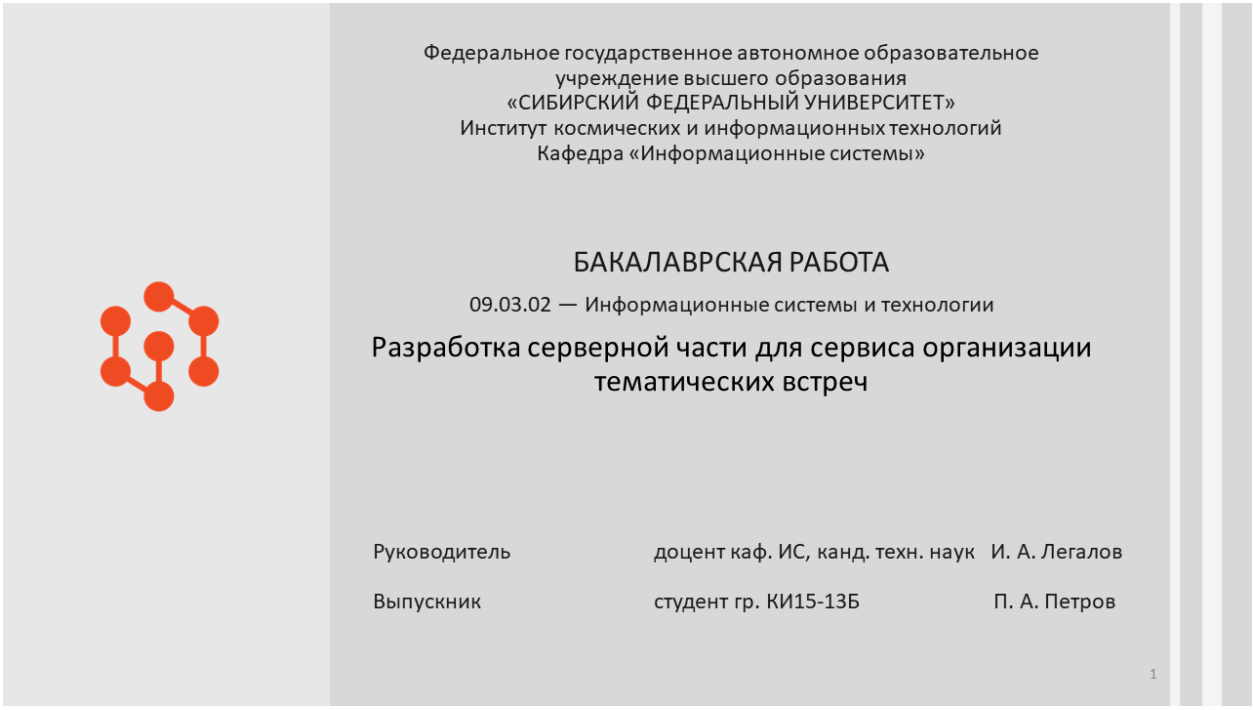


Рисунок А.1 — Плакат презентации №1

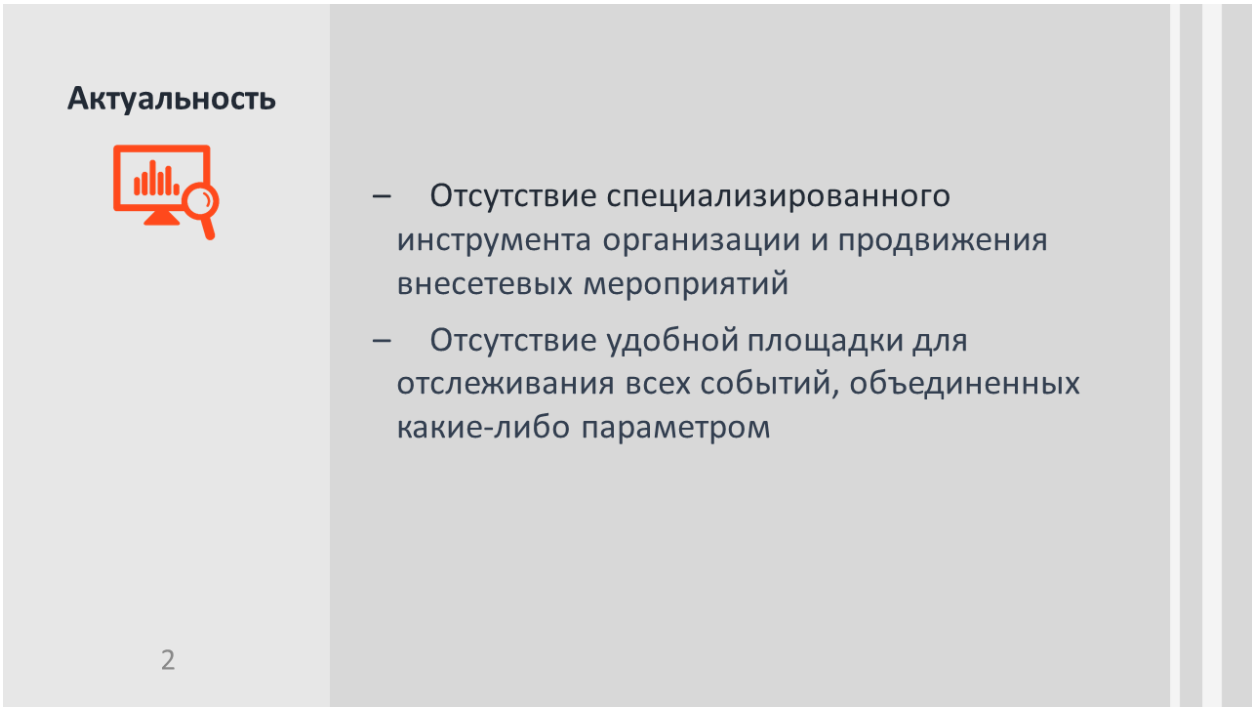


Рисунок А. 2 — Плакат презентации №2

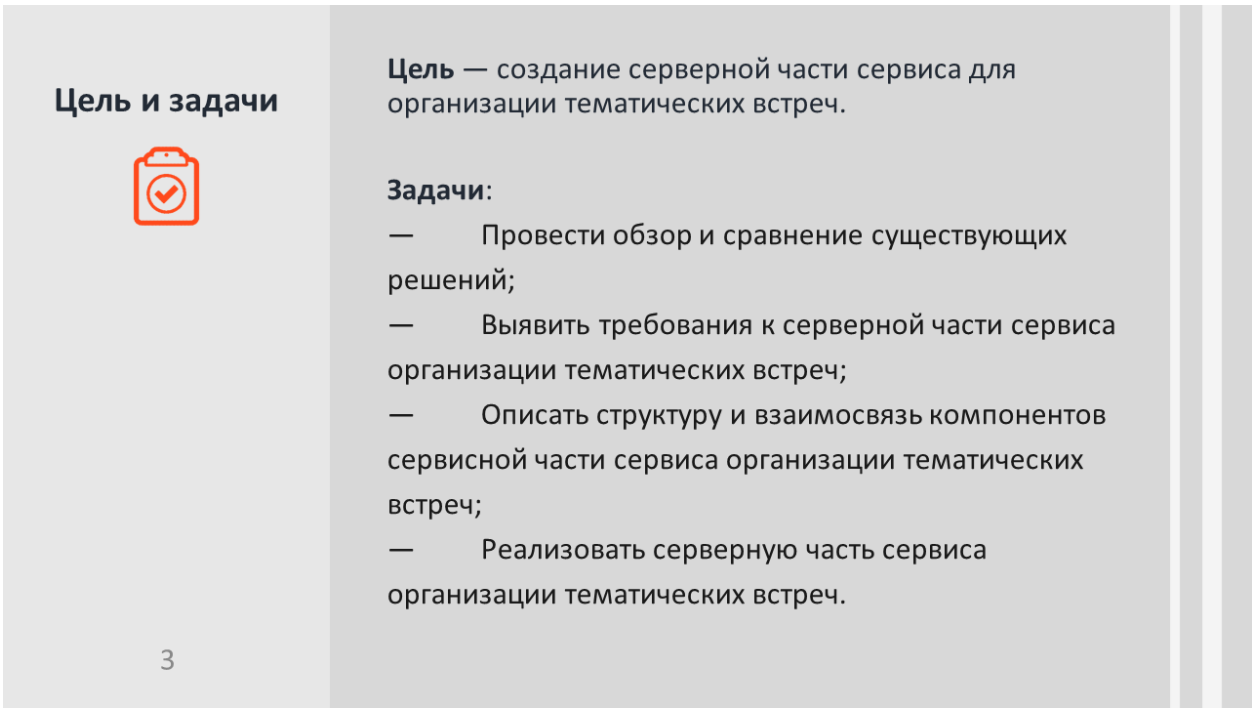


Рисунок А.3 — Плакат презентации №3

	Социальные сети		Афиши	Тематические приложения
	«YUSHIN BROTHERS»	«кто со мной?»	kudago.com	«Geoscope»
Информативность	+	-	+	-
Полнота	-	-	+	-
Доступность	-	+	-	+
Персонализация	-	-	-	-
Геопозиционирование	-	-	-	+

Рисунок А.4 — Плакат презентации №4

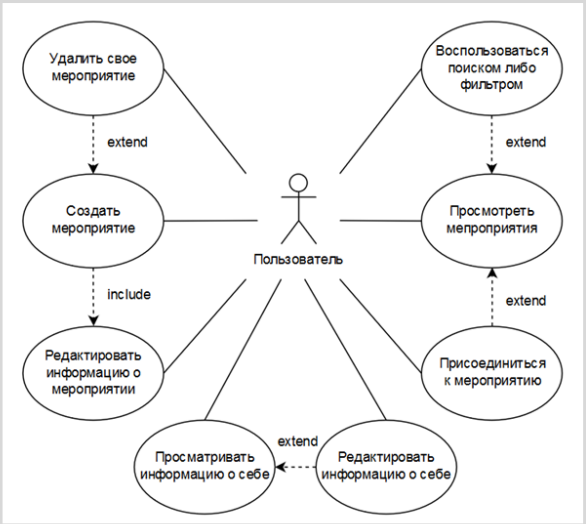
Пользовательские требования к ИС

1. Пользователь может зарегистрироваться и редактировать **информацию о себе**;
2. Пользователь может просматривать **список всех мероприятий**;
3. Пользователь может **присоединиться** к интересующему его мероприятию;
4. Пользователь может зарегистрировать **свое мероприятие**.

5

Рисунок А.5 — Плакат презентации №5

Диаграмма вариантов использования



6

Рисунок А.6 — Плакат презентации №6

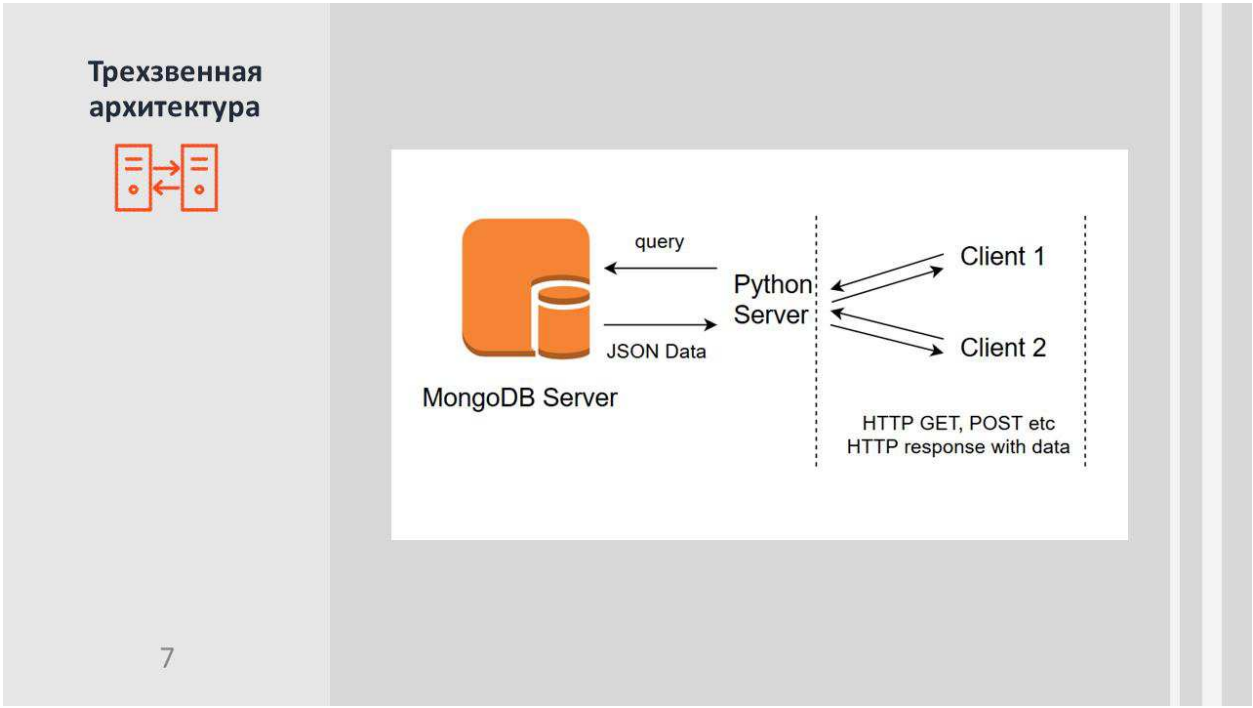


Рисунок А.7 — Плакат презентации №7

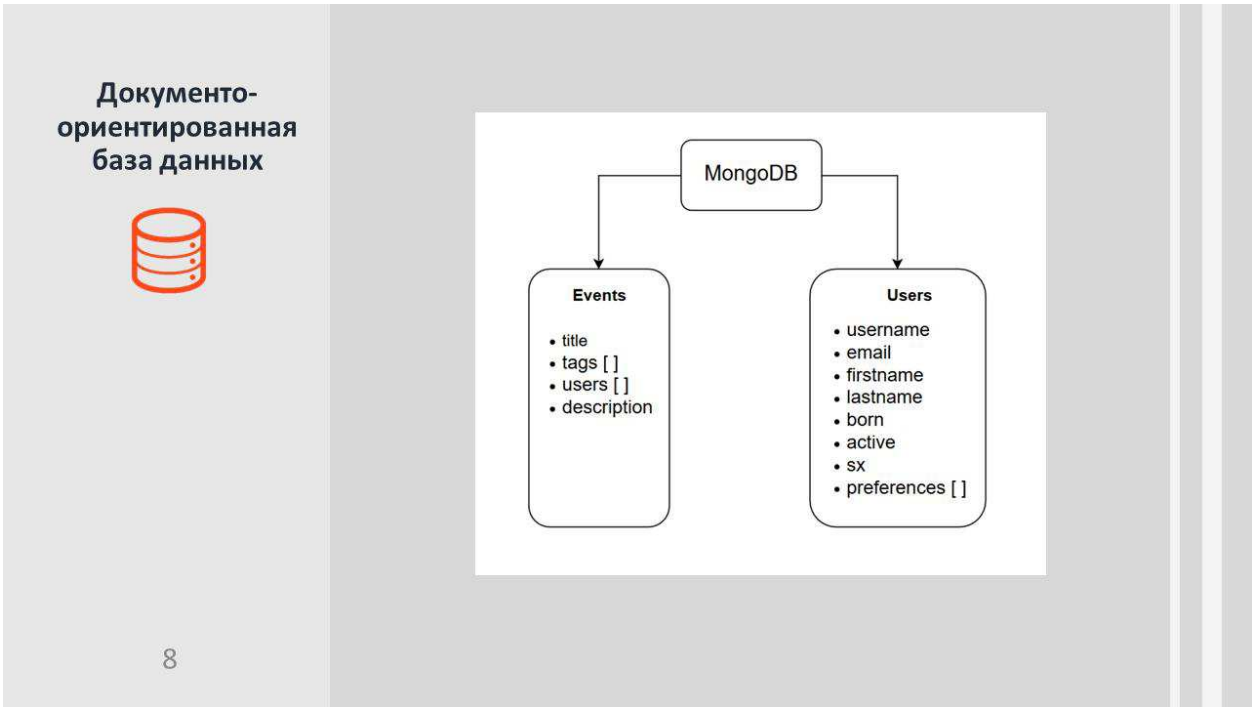


Рисунок А.8 — Плакат презентации №8

Серверная часть

Схема реализации CRUD-операций в библиотеке *Eve*

Action	HTTP Verb	Context
Create	POST	Collection
Create	PUT	Document
Replace	PUT	Document
Read	GET, POST	Collection/Document
Update	PATCH	Document
Delete	DELETE	Collection/Document

9

Рисунок А.9 — Плакат презентации №9

Заключение

- Выполнен обзор с сравнение существующих решений;
- Представлены функциональные требования к серверной части сервиса;
- Представлена диаграмма вариантов использования;
- Описана структура базы данных;
- Определены технические средства разработки серверной части сервиса;
- Представлена разработка серверной части сервиса.

10

Рисунок А.10 — Плакат презентации №10

Федеральное государственное автономное образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра «Информационные системы»

УТВЕРЖДАЮ
Заведующий кафедрой ИС
_____ П. П. Дьячук
подпись
«21» 06 2019 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 — Информационные системы и технологии

Разработка серверной части для сервиса организации тематических встреч

Руководитель _____ 21.06.19 доцент каф. ИС И. А. Легалов
подпись, дата

Выпускник _____ 21.06.19 П. А. Петров
подпись, дата

Нормоконтролер _____ 21.06.19 Ю. В. Шмагрис
подпись, дата

Красноярск 2019