МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)"

ФАКУЛЬТЕТ ИННОВАЦИЙ И ВЫСОКИХ ТЕХНОЛОГИЙ
КАФЕДРА ТЕОРЕТИЧЕСКИХ И ПРИКЛАДНЫХ ПРОБЛЕМ ИННОВАЦИЙ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(МАГИСТЕРСКАЯ РАБОТА)

Направление подготовки: 03.04.01 "Прикладные математика и физика"

НА ТЕМУ:

Единая автоматизированная информационная система поддержки и сопровождения проектов, созданных с применением стандарта BIM

Студент _____ Княжев В.А.

Научный руководитель _____ Зырин С.В.

г. Москва, 2019

Оглавление

Глава 1

Введение

1.1 Актуальность проблемы

Темпы строительства зданий и промышленных объектов в мире и сложность конструкций увеличивается с каждым годом [1]. Ранее использовавшиеся методы проектирования чертежей на бумаге отходят на второй план, и все более активно используются компьютерные технологии [2], а также становится очевидной необходимость повсеместного введения стандартов проектирования зданий.

Одним из наиболее современных стандартов проектирования является стандарт BIM (Building Information Modeling) [3]. Его концепция позволяет не только проектировать здания, но также охватить весь их жизненный цикл: от управления затратами и строительством здания до его эксплуатации.

Подобная всеобъемлемость хороша тем, что вся информация о конструкции содержится в одном проекте. Это помогает сохранять целостность данных, позволяет быстрее выявлять ошибки и уменьшать стоимость ремонта. Но также из этого вытекает необходимость координации одновременной работы большого количества людей над одним проектом: крупных команд архитекторов, иногда распределенных по всему миру, эксплуатирующих организаций и всех других людей, участвующих в обслуживании здания. Поэтому очень важно иметь возможность одновременного изменения BIM представления объекта разными людьми с минимизацией рисков потери каких-либо данных.

Малейшая ошибка в одном из элементов конструкции, не обнаруженная вовремя, может привести к серьезным последствиям, например к дополнительным затратам на проект. Поэтому важно в любой момент времени иметь доступ к электронному журналу контроля всех изменений проекта. Использование стандартных систем контроля версий, таких как *Git*, не позволяет учесть специфику предметной области. Специализированное программное обеспечение накладывает большие ограничения на окружение запуска, вплоть до ограничения на операционные системы. Также оно позволяет работать только с определенными форматами данных, которые поддерживаются только им. Для улучшения координации работы между командами специалистов разных областей требуется наличие возможности использования универсального формата данных.

1.2 Постановка задачи

Требуется разработать веб-систему на основе универсального формата данных, которая бы могла предоставить пользователям следующие возможности:

1. Управление жизненным циклом проектов.
   Создание проекта, добавление, редактирование и удаление файлов, управление правами доступа к проекту.

2. Отслеживание изменений проекта во времени.
   Отображение списка всех изменений проекта, а также возможность просмотра версии данных или внесенных в проект изменений в конкретный момент времени.

3. Одновременное внесение изменений в проекты несколькими пользователями.
   Пользователи могут работать над разными частями проекта в одно и то же время. При наличии конфликтующих изменений предоставляется возможность применения модификаций, внесенных как другими пользователями, так и текущим.

4. Подготовка окружения, запуск системы и ее масштабируемость.
   Возможность быстрой подготовки окружения и запуска сервиса для мговенного развертывания веб-платформы. В моменты пиковой нагрузки пользователей, веб-платформа не должна терять производительность.

Также система должна предоставлять как внешнее API для интегрирования с другими системами, так и веб-интерфейс для доступа обычным пользователям.

Глава 2

Основная часть

2.1 Стандарт BIM

BIM (Building Information Modeling или информационное моделирование здания) - это стандарт создания проектов строительных объектов, с учетом физических и функциональных характеристик. В основе проектов, созданных по этому стандарту, лежит информация о каждом из создаваемых объектов.

Исторически, компьютерные проекты зданий были просто цифровыми аналогами бумажных чертежей, которые использовались архитекторами, дизайнерами, инженерами и строительными бригадами. Значительные сложности могут быть вызваны тем, что разные команды должны получать доступ к информации проекта по-разному, часто в разное время на протяжении всего проекта.

Архитекторам нужны «чертежи» - планы, разрезы и фасады. Дизайнерам интерьера нужны 3D модели. Строителям и инженерам нужны документы и схемы в формате используемого в организации ПО для соответствующих задач. Если использовать стандартные подходы к планированию, необходимо затратить время специалистов на графическую конвертацию архитектурного чертежа в необходимый формат. Любые внесенные в архитектурный проект изменения должны распространяться по каждому файлу вручную, и затем проверяться на правильность.

В подобной схеме организации работы проектирование здания становится

трудоемким итеративным процессом без единого источника информации об объекте, и соответственно с большой вероятностью ошибок и несоответствий между версиями разных специалистов.

Преимущества BIM

BIM позволяет решать эти проблемы: на передний план выходит не графическое представление, а информация о каждом из объектов в будущем здании, а затем уже эти данные представляются графически.
В общем, объект BIM - это любой аспект здания, который не является частью информации о материалах строительства. Объекты BIM подразделяются на два типа: составные и слоистые.

- составные объекты – имеют фиксированные очертания и размеры. Это могут быть окна, двери, трубопроводы, воздуховоды, электромонтажные работы и т. д.

- слоистые объекты – не имеют фиксированных форм или размеров и включают в себя большинство конструктивных элементов, таких как кровля, стены и потолки.

С практической точки зрения объекты BIM далее подразделяются на общие и частные объекты. Общие объекты содержатся в библиотеках объектов, общих для большинства программ BIM, и используются на начальном этапе проектирования. Частные объекты - это объекты, которые отражают характеристики конкретных коммерческих продуктов, которые будут использоваться в строительстве. Как правило, они создаются для каждого проекта индивидуально.
Входные данные о конструкции и дизайне, состоящие из объектов BIM, могут быть сделаны через графический интерфейс. Информация о разных сферах проекта хранится отдельно, но может быть сгенерирована в любом количестве форматов.
Подобная структура проектов, созданных по стандарту BIM, предоставляет возможности для работы нескольких команд, которые могут физически находиться в разных частях света, одновременно над одним проектом [4].

Автоматическая и простая конвертация из одного формата в другой позволяет сокращать трудозатраты специалистов и соответственно облегчает взаимодействие между несколькими командами.

Уровни зрелости стандарта BIM

1. **Уровень 0 BIM** – фактически означает только 2D чертеж и отсутствие коллаборации. Распространение проекта между группами осуществляется через бумажные или электронные версии.

2. **Уровень 1 BIM** – на этом уровне 3D модели используются для визуализации концепции, и 2D чертежи для разработки строительной документации. Обмен информацией осуществляется через общую среду данных, которая обычно обеспечивается заказчиком.

3. **Уровень 2 BIM** – представляет собой комплексную модель, работа над которой ведется специалистами из разных областей деятельности в различных программах. Сборка общей модели данных, ее анализ и выявление коллизий (конфликтов) должны осуществляются в специальных программных приложениях. Данный уровень предполагает добавление следующих измерений: 4D (время) и 5D (стоимость).
   Любое программное обеспечение САПР, используемое каждой из команд, должно быть способно экспортировать данные о проекте в один из распространенных форматов файлов, таких как IFC или COBie.

4. **Уровень 3 BIM** – разрабатываемый проект использует единую интегрированную модель, содержащуюся в отдельных дисциплинарных “инструментах BIM” с вложенными данными, которая также совместима с открытым форматом данных IFC. Данная модель создается и используется для управления жизненным циклом проекта, также в ней хранится информация о выполнении строительных работ и затратах.

Применение стандарта BIM

Стандарт BIM был введен в ряде стран на законодательном уровне [5]. Первыми ввели обязательное использование стандарта BIM ряд скандинавских стран: в Финляндии и Дании он обязателен с 2007 года.

Так, правительство Великобритании обязало всех участников отрасли с апреля 2016 года выполнять все финансируемые государством проекты на втором уровне зрелости [6]. Решение было принято в частности после того, как стало ясно, что использование BIM стандарта второго уровня зрелости помогло сэкономить £840M в 2013-2014 годах, в нескольких больших европейских странах, включая Францию и Германию.

В России использование стандарта BIM будет введено летом 2019 года для зданий, строящихся по государственному заказу по поручению президента [7].

2.2 Формат данных

Существует множество способов хранения и управления данными в рабочем процессе BIM. В результате пользователям предлгается большое количество различных форматов файлов. Обычно пользователи работают со специализированным программным обеспечением в зависимости от их области деятельности, которое обычно использует проприетарный специфический формат данных, об устройстве которых знают только разработчики. Нам же надо выбрать универсальный формат данных, чтобы и архитекторы, и инженеры, и другие пользователи могли работать с ним.

Форматы данных BIM

Рассмотрим форматы данных, которые реализуют стандарт BIM. Можно выделить две основные категории форматов данных:

- проприетарные
  Файлы, которые могут быть прочитаны только определенным программны обеспечением.
  Примеры форматов:
  - RVT
    Собственный формат Autodesk для файлов Revit, который можно открыть только в специализированной программе Revit.
  - NWD
    Собственный формат Autodesk для файлов Navisworks, который можно открыть только в Navisworks Freedom или Navisworks Manage.
  - DWG
    Собственный формат Autodesk для файлов AutoCAD. Данный формат является наиболее универсальным для просмотра и создания проектов. Файлы DWG доступны для редактирования в любой программе на основе САПР.
- открытые
  Открытые форматы файлов не зависят от производителя. Они могут

быть прочитаны и отредактированы любым программным обеспечением. Обычно данные форматы данных сопровождаются открытым исходным кодом и возможностью развития мировым обществом.
Примеры форматов:

- IFC

  Industry Foundation Classes (IFC)- наиболее распространенный формат данных с открытой спецификацией, реализующий стандарт BIM. Ряд программ, включая Revit и Navisworks, могут открывать файлы IFC и работать с ними.

- COBie

  Открытый формат данных, который не хранит в себе графические / геометрические данные, но позволяет передавать большие наборы разных данных, созданных во время проектирования и строительства конечному пользователю в удобочитаемом виде.

Использование проприетарных форматов вендоров программного обеспечения может препятствовать взаимодействию членов команды, если они пользуются различными типами данных. Нам требуется найти такое решение, которое позволит поддержать взаимосвязи различных BIM моделей и формата обмена данными. Остановимся на открытых форматах данных BIM, которые подразумевают универсальный подход к созданию проекта, строительству и эксплуатации объектов, базирующийся на открытых стандартах и процессах.
Основная разница между двумя выше указанными открытыми форматами данных модели BIM состоит в том, что COBie помогает профессионалам обмениваться различными данными разных форматов, сохраняя их в удобочитаемой форме, тогда как IFC помогает различным программам понимать и обмениваться данными в едином формате с учетом иерархии и взаимосвязей между компонентами архитектурно-строительных моделей. Предпочтение было отдано формату данных IFC, преимущества которого описаны в следующем параграфе.

Формат данных IFC

Подытоживая, можно выделить следующие основные преимущества IFC формата:

1. Единый язык модели для различных областей использования.
2. Универсален, позволяет хранить различные данные в одной модели.
3. Открытый формат данных.
4. Разрабатывается независимым сообществом.
5. Наличие иерархии и взаимосвязей между компонентами архитектурно-строительной модели.
6. Возможность конвертации в/из других форматов данных.

Пример данных:

```
ISO-10303-21;
  HEADER;
    FILE_DESCRIPTION( (' ViewDefinition [CoordinationView]'),'2;1');
    FILE_NAME('TEST_Building.ifc',
      '2019-02-11T11:35:07',
     ('Valeriy Knyazhev','valeriy.knyazhev@yandex.ru'),
     ('Revolut'),
      'ECCO Toolkit Version V 3.2.1',
      'IfcExplorer Version 2.2a (Build 11)',
      'Some key');
    FILE_SCHEMA( ('IFC2X3') );
  ENDSEC;
  DATA;
    #1 =(); /* actual content of the IFC exchange structure */
  ENDSEC;
END-ISO-10303-21;
```

Рис. 2.1: Формат данных IFC

1. ISO-10303-21 – спецификация языка описания BIM моделей.
2. HEADER – блок описания файла.
   (a) FILE_DESCRIPTION – описание опций создания контента.
   (b) FILE_NAME – информация о создании файла, организации, а также об используемом программном обеспечении.
   (c) FILE_SCHEMA – версия сехмы данных IFC.
3. DATA – блок описания BIM представления объекта.

2.3 Пользовательские истории

Пользовательские истории (user story) – способ описания требований к разрабатываемому продукту, которые сформулированы на понятном пользователю языке. Каждая пользовательская история должна быть ограничена в размере и сложности ее реализации.
Пользовательские истории – быстрый способ документирования основных требований клиента, их целью является оперативное регирование на изменения требований реального мира.
Текст каждой пользовательской истории должен пояснять роль пользователя и его действия в системе. Для начала требуется определить список основных пользователей нашей системы.

2.3.1 Набор персонажей

Для получения конечного списка персонажей используется иерархическая кластеризация. Требуется составить список протоперсонажей (персонажей-гипотез), создать список шкал умений, поведенческих и мотивационных переменных. Далее, с помощью кластеризации персонажей по ранее составленным критериям (в данной работе проводится кластеризация с помощью дендограмм), анализируется первоначальный список персонажей и сокращается до самых значимых.

Персонажи-гипотезы

1. Архитектор
2. Главный архитектор
3. Дизайнер
4. Студент технического направления
5. Инвестор
6. Строитель

Умения, поведенческие и мотивационные переменные

Выделены следующие характеристики, в той или иной мере описывающие наших персонажей-прототипов.

1. Уровень технического образования
   1 - законченная средняя школа, 5 - PhD

2. Знание иностранных языков
   1 - beginner, 5 - proficiency

3. Опыт работы с персональными компьютерами
   1 - обычный пользователь, 5 - администратор ПК

4. Цели использования нашей системы
   1 - протестировать систему, 5 - коммерческое использование

5. Частота использования нашей системы
   1 - единичное использование, 5 - постоянное взаимодействие с системой

6. Масштабы создаваемых проектов
   1 - домашние маленькие проекты, 5 - проекты крупных зданий

7. Умение проектировать архитектурные модели
   1 - нет опыта, 5 - опыт более 5 лет и профессиональное образование

8. Платежеспособность
   1 - отсутствие постоянного дохода, 5 - наличие крупных бизнесов

Матрица сходства персонажей

В данной матрице в первом столбце перечислены все персонажи-прототипы, в первой строке перечислены номера указанных выше описательных характеристик.

Таблица 2.1: Матрица сочетаний персонажей и характеристик

Персонаж	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Архитектор	4	2	3	4	4	4	4	3
Гл. архитектор	5	5	4	5	5	5	5	4
Дизайнер	3	2	3	3	3	2	3	3
Студент	3	2	3	2	2	2	3	1
Инвестор	4	5	4	5	2	5	1	5
Строитель	2	1	2	3	3	4	2	2

Построение дендограммы

Под дендрограммой понимается дерево, которое построено на основе матрицы мер близости. Она позволяет отобразить взаимные связи между объектами из первоначально заданного множества объектов. Для построения дендрограммы требуется составить матрицу сходства, которая определяет уровень сходства между парами кластеров.
При построении дендограмм могут использоваться следующие способы кластеризации данных:

- *Агломеративные методы*
  Когда новые кластеры создаются путем объединения более мелких кластеров, дерево строится от листьев к корню.

- *Дивизивные или дивизионные методы*
  Когда более крупные кластеры делятся на более мелкие, дерево строится от корня к листьям.

В данной работе используется агломеративный метод, а именно метод одиночной связи (ближайшего соседа). Расстояние между двумя различными

кластерами берется равным минимальному расстоянию между двумя элементами из них

$$dist(\mathcal{A}, \mathcal{B}) = min\{d(a, b) : a \in \mathcal{A}, b \in \mathcal{B}\} \tag{2.1}$$

где $d(a, b)$ *– расстояние между элементами* a *и* b*, принадлежащими кластерам* $\mathcal{A}$ *и* $\mathcal{B}$

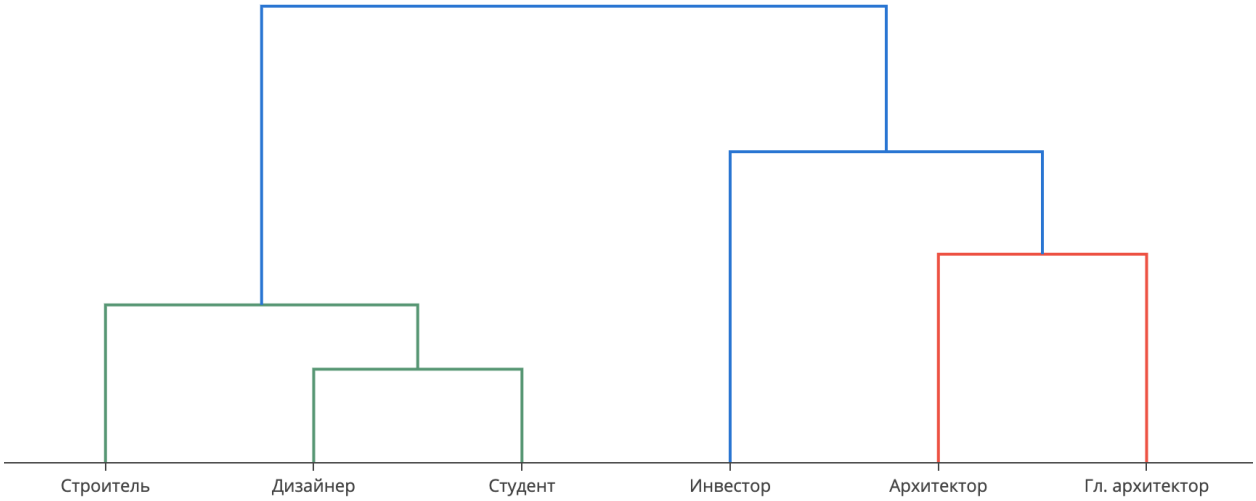

Рис. 2.2: Дендограмма.

Итоговый набор персонажей

Конечный набор персонажей, которые будут анализироваться в данной работе, представлен ниже:

- Главный архитектор
- Архитектор
- Куратор проекта
- Строитель

Куратором проекта в данном случае является любое лицо, которое желает следить за процессом разработки архитектурных проектов, например, инвестор.

2.3.2 Действия пользователей

Описанные в разделе ранее пользователи имеют следующие цели при работе с нашей системой, а также планируют выполнять определенные действия с некоторыми ограничениями.

Главный архитектор

Личные цели:

1. Создание крупных архитектурных проектов зданий.
2. Контроль над процессом разработки проекта.

Действия:

1. Начать новый архитектурный проект.
2. Итеративная разработка отдельных частей архитектурной модели.
3. Анализ выполненной командой работы.
4. Управление командами проектирования в различных проектах.

Требования:

1. Масштабируемость системы для поддержки работы большого количества людей.
2. Возможность получить данные проекта в любой выбранный промежуток времени.
3. Создание проектов в едином формате данных (IFC4 или IFC2x3).
4. Доступ к системе из любой точки мира.
5. Возможность предоставить права доступа к выбранному проекту как только на чтение, так и на редактирование.

Архитектор

Личные цели:

1. Проектирование сложных архитектурных строений.
2. Обучение на основе истории изменений проекта.

Действия

1. Итеративная разработка отдельных частей архитектурной модели.
2. Анализ выполненной другими членами команды работы над проектом.

Требования

1. Возможность одновременной работы с другими членами команды.
2. Возможность детализированного просмотра изменений проекта.
3. Разработка проектов в едином формате данных.
4. Отказоустойчивость системы.

Куратор

Личные цели:

1. Контроль над инвестиционными архитектурными проектами.

Действия

1. Просмотр истории этапов проектирования и строительства архитектурного проекта.
2. Получение детализированного описания изменений в выбранной итерации разработки проекта.

Требования

1. Понятный веб-интерфейс системы.

2. Доступ к системе из любой точки мира.

3. Возможность использования API системы для стороннего проекта.

4. Целостность хранимой информации.

Строитель

Личные цели:

1. Планирование строительства на основе функциональных и физических данных архитектурного проекта.

Действия

1. Детальное изучение цифрового контента выбранной версии проекта с целью извлечения важных для строительства характеристик.

Требования

1. Простой веб-интерфейс системы.

2. Получение данных о модели архитектурного объекта в выбранной итерации разработки.

3. Сравнительно недорогая лицензия на использование продукта.

2.3.3 Карта пользовательских историй

Итоговый набор пользовательских историй выглядит следующим образом.

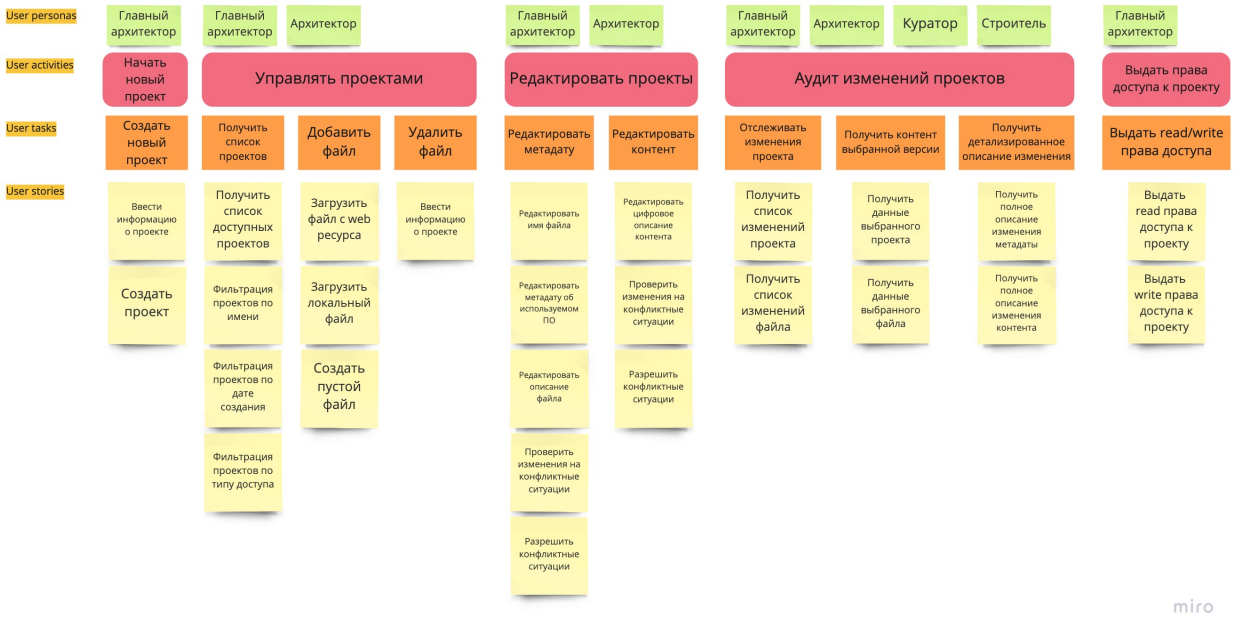

Рис. 2.3: Пользовательские истории.

2.4 Бизнес-требования

Бизнес-требования (business requirements) – информация, в совокупности описывающая потребность, которая инициирует один или больше проектов с целью предоставить решение и получить требуемый конечный результат. В основу бизнес-требований ложатся бизнес-возможности, бизнес-цели, критерии успеха и положение о концепции.

Бизнес-требования определяют концепцию решения и границы проекта, в котором оно будет реализовываться.

Концепция и границы – два базовых элемента бизнес-требований.

Концепция продукта (product vision) должна кратко описывать конечный продукт, который в свое время должен достигать заданных бизнес-целей.

Границы проекта (project scope) показывают, какая часть конечной концепции продукта будет реализована в текущей итерации.

В данной работе границы проекта совпадают с концепцией решения.

Документ о концепции и границах (vision and scope document) – единый документ, который включает в себя все бизнес-требования.

Далее будут представлены основные пункты этого документа, составленные на основе описанных ранее пользовательских историй.

2.4.1 Исходные данные

Требуется разработать веб-платформу для совместной работы над архитектурными проектами с ведением электронного журнала контроля изменений и возможностью рецензирования и интегрирования в проект модификаций, с разрешением возможных конфликтных ситуаций.

2.4.2 Бизнес-цели

Таблица 2.2: Нефинансовые цели

№	Цель
Н1	Разработать веб-платформу для управления жизненным циклом архитектурных проектов.
Н2	Реализовать возможность одновременного редактирования проектов и разрешения конфликтов в случае их наличия.
Н3	Реализовать хранение журнала контроля всех изменений проектов и возможность его просмотра.

2.4.3 Критерии успеха

- Веб-платформа позволяет управлять жизненным циклом архитектурного проекта.
- Веб-платформа предоставляет возможность просмотра электронного журнала контроля изменений проекта.
- Веб-платформа позволяет разрешать конфликты, возникающие при одновременном редактировании, с минимизацией рисков потери данных.

2.4.4 Положение о концепции проекта

Для пользователей, которым требуется управлять жизненным циклом архитектурных проектов и иметь возможность отслеживать изменения во времени, данная работа является веб-платформой, которая будет выступать в качестве единой системы по хранению и изменению архитектурных проектов с минимизацией рисков потери данных с возможностью просмотра электронного журнала контроля изменений.

2.5 Ограничения системы

2.5.1 Основные функции

1. Просмотр списка доступных пользователю проектов.
2. Создание проекта.
3. Управление правами доступа к проекту.
4. Добавление файлов в проект.
5. Удаление файлов из проекта.
6. Просмотр метадаты и контента файла.
7. Редактирование метадаты проекта и файлов.
8. Редактирование контента файла.
9. Разрешение конфликтных ситуаций при редактировании файлов проекта.
10. Просмотр журнала контроля изменений проекта.
11. Просмотр контента проекта в определенный промежуток времени.
12. Просмотр детализированного описания изменения, внесенного в проект в определенный момент времени.

2.5.2 Ограничения и исключения

- Размер каждого файла должен не превышать 150 Мб (ограничение IFC формата).
- В данной работе не предполагается возможность создания связанных между собой файлов с BIM представлениями объектов.

2.6 Функции системы

1. Просмотр списка проектов

   Пользователь может получить список доступных ему проектов. В каждом элементе списка предоставляется краткая информация о проекте. Также имеется возможность фильтрации данных по различным описывающим проект характеристикам.

Таблица 2.3: Просмотр списка проектов

Функциональные требования:	
ПСПФ1	Система должна предоставить список всех проектов, которые доступны пользователю.
ПСПФ2	Система должна предоставить возможность фильтрации результатов поиска по следующим параметрам: имя проекта, дата создания, тип прав доступа.
ПСПФ3	Записи проектов должны содержать следующую информацию: имя, описание проекта, даты создания и последнего изменения, имя владельца, а также краткую информацию о файлах.
Нефункциональные требования:	
ПСПН1	Пользователю отображаются только те проекты, владельцем которых он является, или к которым он имеет права доступа на чтение или редактирование .

2. Создание проекта

   Пользователь может создать новый проект с возможностью указания его названия и полного описания.

Таблица 2.4: Создание проекта

Функциональные требования:	
СПФ1	При создании проекта система должна предоставить пользователю идентификатор, по которому он теперь сможет работать с только что созданным проектом.
СПФ2	При создании проекта система предоставляет пользователю возможность ввести имя и описание нового проекта.
Нефункциональные требования:	
СПН1	Название проекта должно содержать не менее 1 и не более 50 символов.
СПН1	Описание проекта должно быть не длиннее 500 символов.

3. Управление правами доступа к проекту

   Пользователь может предоставить права доступа к проекту либо на чтение, либо на редактирование другим пользователям системы.

   Права на чтение подразумевают только просмотр всех данных проекта и его изменений.

   Права на редактирование включают в себя права на чтение, а также возможность управления жизненным циклом проекта.

Таблица 2.5: Управление правами доступа

Функциональные требования:	
ПДФ1	Система предоставляет возможность выдать права доступа к проекту другим пользователям.
Нефункциональные требования:	
ПДН1	Права пользователей подразделяются два типа: чтение, редактирование.
ПДН2	Только владелец проекта имеет возможность предоставлять права доступа к проекту другим пользователям системы.
ПДН3	По умолчанию только что созданный проект доступен для чтения и редактирования только его владельцу.

4. Добавление файлов в проект

   Пользователь может добавить новый файл с метадатой и контентом, описывающим BIM представление объекта. Загружать файл можно как по ссылке, так и с помощью прикрепления локального файла с данными. Также имеется возможность создать пустой файл.

Таблица 2.6: Добавление файлов в проект

Функциональные требования:	
ДФФ1	Система предоставляет возможность добавления файлов только в случае наличия прав доступа пользователя на редактирование проекта.
ДФФ2	При невозможности загрузить данные система должна оповестить об этом пользователя (с указанием причины).
ДФФ3	Система предоставляет возможность загрузить файл с другого веб-источника данных.
ДФФ4	Система предоставляет возможность загрузить локальный файл с персонального компьютера пользователя.
ДФФ5	Система предоставляет возможность создать полностью пустой файл.
Требования к данным:	
ДФД1	Формат загружаемых данных должен соответствовать IFC.
ДФД2	Максимальный размер загружаемых данных - 150 Мб.

5. Удаление файлов из проекта

   Пользователь может удалить любой файл из проекта.

Таблица 2.7: Удаление файлов из проекта

Функциональные требования:	
ДФФ1	Система предоставляет возможность удаления файлов
ДФФ2	Удаление файлов возможно только в случае наличия прав доступа у пользователя на редактирование проекта.

6. Просмотр метадаты и контента файла

   Пользователь может посмотреть метадату и контент, описывающий BIM представление объекта, файла.

Таблица 2.8: Просмотр метадаты и контента файла

Функциональные требования:	
ПМКФ1	Система предоставляет возможность получения данных файла только в случае наличия прав доступа пользователя на чтение проекта.
ПМКФ2	Система предоставляет возможность получить данные метадаты файла.
ПМКФ3	Система предоставляет возможность получить контент файла, который описывает BIM представление объекта.

7. Редактирование метадаты проекта и файлов

   Пользователь может изменить название и описание уже существующего проекта.

   Также он имеет возможность измененить метадату файлов проекта.

Таблица 2.9: Редактирование метадаты проекта и файлов

Функциональные требования:	
РМФ1	Система предоставляет возможность редактирования метадаты проекта или файлов только в случае наличия прав доступа пользователя на редактирование этого проекта.
РМФ2	Система предоставляет возможность редактирования названия и описания существующих проектов.
РМФ3	Система предоставляет возможность редактирования метадаты файлов в проектах.
Требования к данным:	
РМД1	Название проекта должно содержать не менее 1 и не более 50 символов.
РМД1	Описание проекта должно быть не длиннее 500 символов.

8. Редактирование контента файла

   Пользователь имеет возможность изменить контент существующих файлов в проекте.

Таблица 2.10: Редактирование контента файла

Функциональные требования:	
РКФ1	После внесения изменений в контент текущей версии файла система должна проверить корректность данного изменения и оповестить пользователя либо о невозможности выполнения, либо об успешности операции.
РКФ2	После внесения изменений в контент файла система должна обновить историю проекта.
РКФ3	Система предоставляет возможность редактирования контента файла только в случае наличия прав доступа пользователя на редактирование проекта.
РКФ4	Для возможности нахождения конфликтов и дальнейшего решения их система должна получать номер изменения, для которого изначально были получены данные.
Нефункциональные требования:	
РКН1	Вносить изменения разрешается только в существующие файлы.

9. Разрешение конфликтных ситуаций при редактировании файлов проекта

   Пользователь может получить список конфликтных изменений, а затем, решив их, обновить данные файлов.

Таблица 2.11: Разрешение конфликтных ситуаций при редактировании файла

Функциональные требования:	
РКСФ1	Система должна предоставить возможность получения получения информации о конфликтах изменения файлов проекта.
РКСФ2	Система должна предоставить возможность решения конфликтов и дальнейшего обновления данных в файлах.
РКСФ3	Система предоставляет возможность выявления и решения конфликтных ситуаций только в случае наличия прав доступа пользователя на редактирование проекта.

10. Просмотр журнала контроля изменений проекта

    Пользователь может посмотреть список всех изменений, который происходили с проектом за весь его жизненный цикл. Также имеется возможность получения списка изменений только для определенного файла этого проекта.

Таблица 2.12: Просмотр журнала контроля изменений проекта

Функциональные требования:	
ЖАИФ1	Система должна предоставить возможность получения полного списка изменений проекта.
ЖАИФ2	Система должна предоставить возможность получения полного списка изменений определенного файла проекта.
ЖАИФ3	Система предоставляет возможность просмотра журнала контроля изменений только в случае наличия прав доступа пользователя на чтение проекта.
ЖАИФ4	Записи изменений должны содержать следующую информацию: идентификатор изменения, идентификатор родительского изменения, время, автор и сообщение изменения.
Нефункциональные требования:	
ЖАИН1	Просмотр списка изменений для определенного файла проекта доступен только для существующих на момент выполнения действия файлов.

11. Просмотр контента проекта в определенный момент времени
    Пользователь может получить данные о проекте и его файлах, в том числе и контент, описывающий BIM представления объектов, для любой записи в журнале контроля изменений проекта.

Таблица 2.13: Просмотр контента проекта в определенный момент времени

Функциональные требования:	
КМВФ1	Система должна предоставить возможность получения метадаты проекта в любой момент времени.
КМВФ2	Система должна предоставить возможность получения списка всех файлов проекта в любой момент времени.
КМВФ3	Записи файлов проекта, полученные для любого момент времени его жизненного цикла, должны содержать метадату и контент, который описывает BIM представление объекта.
КМВФ4	Система предоставляет возможность просмотра контента проекта и его файлов только в случае наличия прав доступа пользователя на чтение проекта.

12. Просмотр детализированного описания изменения, внесенного в проект в определенный момент времени

    Пользователь может получить детализированное описание определенного изменения проекта.

Таблица 2.14: Просмотр детализированного описания изменения

Функциональные требования:	
ДОИФ1	Система должна предоставить возможность получения детализированного описания изменения проекта.
ДОИФ2	Детализированное описание изменения проекта должно в себе содержать информацию о том, как изменилась метадата и контент файлов в данном измененнии.
ДОИФ3	Детализированное описание контента файла должно содержать как сами изменения, так и часть обрамлющего из неизмененного контента.
ДОИФ4	Система предоставляет возможность получения детализированного описания изменения только в случае наличия прав доступа пользователя на чтение проекта.

2.7 Описание системы

В данной главе приводится подробное описание следующих особенностей системы:

- Хранение истории изменения проекта.
- Редактирование контента файлов с учетом возможных конфликтных ситуаций и их разрешение.
- Построение проекции данных на определенный момент времени.

2.7.1 Хранение истории изменения проекта

Каждое изменение данных проекта, например метадаты или контента файла сопровождается создание и сохранением новой сущности *Commit*.

```
1  class Commit {
2      @Nonnull Long id;
3      @Nullable Long parentId;
4      @Nonnull String projectId;
5      @Nonnull String author;
6      @Nonnull String message;
7      @Nonnull Time timestamp;
8      @Nonnull Data data;
9  }
10 class Data {
11     @Nonnull List<FileItem> changedFiles;
12 }
13 class FileItem {
14     @Nonnull String fileId;
15     @Nonnull MetadataChanges metadata;
16     @Nonnull DescriptionChanges description;
17     @Nonnull List<CommitItem items;
18 }
```

Algorithm 1: Сущность *Commit*.

Имея старые и новые метадату и контент файла можно вычислить, какие изменения должны быть применены к старым данным проекта, чтобы получить новые данные.

Для структур данных вроде *Metadata* и *Description* набор изменений определяется методом сравнения всех описательных характеристик. Если какая-либо характеристика была изменена, то в новом *Commit* будет отображено новое значение для этой характеристики.
Описание моделей *Metadata* и *Description* выглядит следующим образом:

```
1  class Metadata {
2      @Nonnull String name;
3      @Nonnull Time timestamp;
4      @Nonnull List<String> authors;
5      @Nonnull List<String> organizations;
6      @Nonnull String preprocessorVersion;
7      @Nonnull String originatingSystem;
8      @Nonnull String authorization;
9  }
10 class Description {
11     @Nonnull List<String> descriptions;
12     @Nonnull String implementationLevel;
13 }
```

Algorithm 2: Сущности *Metadata* и *Description*.

Для самого контента файла, содержащего BIM представление объекта, список изменений определяется на основе полнотекстового сравнения, которое описано далее [Algorihtm 4].
В результате работы алгоритма должен быть получен список изменений, которые подразделяются на три типа:

- ADDITION
- DELETION
- MODIFICATION

Эти изменения следует преобразовать в сущность *CommitItem*.

```
1 class CommitItem {
2     @Nonnull String value;
3     @Nonnull Integer position;
4     @Nonnull ChangeType type;
5 }
```

Algorithm 3: Сущность *CommitItem*.

где параметр типа ChangeType может принимать одно из следующих значений: ADDITION, DELETION.

Правила преобразования:

- DELETION → 1 DELETION
- ADDITION → 1 ADDITION
- MODIFICATION → 1 DELETION + 1 ADDITION

Для определения позиции изменения также существуют некоторые правила:

1. ADDITION position - позиция элемента, после которого вставляется новый элемент в контент.
2. DELETION position - позиция элемента, который удаляется из контента.
3. Нумерация начинается с 1
   Связано это с проблемой вставки элементов в начало контента. Таким образом, при вставке в начало будут созданы элементы с позициями, равными 0. Что в свою очередь значит вставить эти элементы после 0'го элемента контента, которого не существует.

Каждый новый *Commit* основывается на изменениях, примененных к предыдущему контенту. Предыдущий контент имеет свой идентификатор изменения, которое привело к получению этих данных. Следовательно новая сущность *Commit* должна указывать на последний *Commit* предыдущего контента. При первом изменении проекта, в базе данных не существует ни единого коммита, который бы относился к этому проекту, в данном случае создается сущность *Commit*, у которой отсутствует родитель

($parentId$), такой *Commit* будем называть корневым. Его данные будут содержать в себе информацию о первом создании/добавлении какого-либо файла в проект.

2.7.2 Редактирование контента файлов с учетом возможных конфликтных ситуаций и их разрешение

При редактировании контента файлов проекта происходит проверка соответствия версии актуальных данных контента и версии, на основе которой пользователь получил старый контент и начал процесс его редактирования.

В случае наличия *Commit*, который уже применен к версии, лежащей в основе полученных данных, проводится проверка списков изменений на наличие пересечений. Если пересечений не обнаружено, то следовательно конфликт не обнаружен и следует, объединив изменения, применить их к актуальной версии данных. Если же конфликт найден, пользователю предоставляется возможность разрешить его путем выбора того или иного изменения элементов данных с последующим редактированием контента конфликтных блоков данных.

В каждом из случаев, обычное редактирование, когда других изменений применено не было, или ситуации, которые были описаны выше, завершающей операцией является приведение контента к новой версии данных. Но в силу специфики предметной области, любым изменением можно повредить связи с основополагающими объектами в BIM представлении.

На данном этапе необходим процесс валидации корректности новых данных. А именно выстраиваются связи вплоть до главных корневых объектов (Rooted Entities) [8], например, IFCWall, IFCDoor и другие объекты, являющиеся подкатегориями объекта IfcProduct. Пользователю сообщается о возможном смысловом конфликте после внесения текущих изменений и предоставляется возможность либо применить эти изменения, либо их отменить.

Только при отсутствии возможных смысловых конфликтов, или согласии

пользователя на обновление модели в противоположном случае, система обновит контент файла и добавит новую сущность *Commit* в историю изменения проекта.

2.7.3 Построение проекции данных на определенный момент времени

Для корректности работы выше описанных сценариев и далее описанных алгоритмов требуется также иметь возможность построить проекцию данных проекта для определенного момента времени [Algorithm 8], в данном случае речь идет о конкретной сущности *Commit*.
Данный функционал используется для получения контента старых версий данных в алгоритме нахождения конфликтных ситуаций, а также для предоставления пользователям возможности посмотреть контент данных на протяжении всей истории изменений проекта.

2.8 Описание алгоритмов

В данном разделе описаны основные алгоритмы нашей системы поддержки и сопровождения проектов по стандарту BIM.

2.8.1 Вычисление списка изменений контента

При каждом редактировании контента файлов, которые содержат в себе описание BIM представления объекта, нам требуется вычислять список изменений, которые были применены к данным.
В данной работе для вычисления списка изменений используется разностный алгоритм Майера [9].

Суть алгоритма сводится к нахождению оптимального пути из начальной точки (0,0) в конечную. Оптимальным считается тот путь, который содержит минимальное число вертикальных и горизонтальных сдвигов. Горизонтальный сдвиг соответствует ADDITION, вертикальный – DELETION. При поиске оптимального пути во время нахождения в точке, из которой есть диагональный путь, переход по диагонали обязателен и не учитывается в длине общего пути.

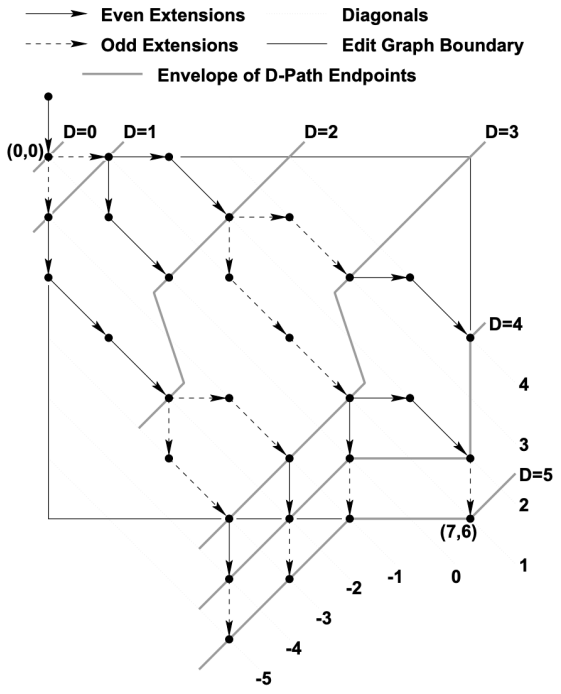

Рис. 2.4: Построение возможных путей решения.

```
   Data: sequence A and B, sizes of sequences N and M
   Result: list of changes
1  Function main(A, N, B, M):
2      matrix := new Matrix(N, M)
3      // find all common subsequences and fill diagonal elements
4      matrix.fillAllCommonSubsequences(A, B)
5      /* search of way from (0, 0) to (N, M) with minimization of
          vertical and horizontal shifts number */
6      return findMinimalWay(matrix, A, N, B, M)
```

Algorithm 4: Разностный алгоритм Майера.

```
   Data: sequence A and B, sizes of sequences N and M
   Result: longest common subsequence
1  Function LCS(A, N, B, M):
2      if (N > 0 AND M > 0) then
3          // optimal way is from (x, y) to (u, v)
4          (x, y), (u, v) = findOptimalSnake(A, B)
5          // where D is minimum number of operations to transform A
              to B
6          if (D > 1) then
7              LCS(A[1 : x], x, B[1 : y], y))
8              return A[x + 1 : u]
9              LCS(A[u + 1 : N], N - u, B[v + 1 : M], M - v)
10         else
11             if (M > N) then
12                 return A[1 : N]
13             else
14                 return B[1 : M]
15             end
16         end
17     end
```

Algorithm 5: Нахождение наибольшей общей подпоследовательности.

2.8.2 Получение детализированного описания изменения

Дано:

Старый контент файла, список изменений контента.

Ожидаемый результат:

Список блоков контента, которые содержат в себе как измененные элементы данных, так и те, которые не изменялись. Неизмененный контент, окружающий измененнные блоки, отображается для лучшего понимания пользователем, в какой части проекта были эти изменения.

Получение детализированного описания изменения всего проекта сводится к получению детализации по каждому файлу проекта.

Получение детализации изменения для одного файла

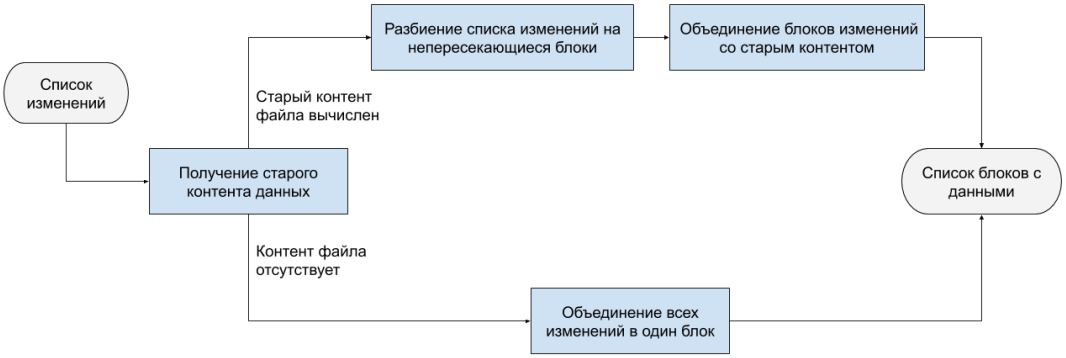

Рис. 2.5: Детализация изменения контента.

Если контент файла отсутствует, значит в текущем изменени проекта этот файл добавлен, что должно трактоваться как один блок с добавлением всего контента. Если же удалось построить контент файла предыдущей версии, значит нам требуется распределить список изменений на непересекающиеся между собой блоки, а после добавить в эти блоки недостающие сведения уже из старого контента.

Разбиение списка изменений на непересекающиеся блоки

```
   Data: list of changes
   params = { offsetSize}
   Result: disjoint blocks of changes
 1 Function main(changes, offsetSize):
 2     disjointBlocks := [ ], blockChanges := [ ]
 3     lastPosition := changes[0].position
 4     for (change : changes) do
 5         if (isNewBlock(change, lastPosition, offsetSize)) then
 6             blockChanges.add(change)
 7         else
 8             newBlock := new Block(blockChanges)
 9             blockChanges := []
10             disjointBlocks.add(newBlock)
11         end
12         lastPosition := change.position
13     end
14     return disjointBlocks
15 Function isNewBlock(change, lastPosition, offsetSize):
16     leftOperand := change.position - offsetSize
17     rightOperand := lastPosition + offsetSize
18     if (change.type == DELETION) then
19         rightOperand ++
20     end
21     if (leftOperand> rightOperand) then
22         return true
23     else
24         return false
25     end
```

Algorithm 6: Разбиение на непересекающиеся блоки изменений.

где offset_size – максимальное допустимое расстояние между элементами контента, которое позволяя определить,относятся ли два последовательных элемента к одному блоку изменений.

Объединение блоков изменений с контентом

```
   Data: list of disjoint blocks of changes, content
   params = { offsetSize}
   Result: disjoint blocks of changes with content
 1 Function main(disjointBlocks, content, offsetSize):
 2     blocks := [ ]
 3     for (block : disjointBlocks) do
 4         contentChanges = []
 5         lastIndex = block.changes[0].position
 6         enrichContent(contentChanges, content, lastIndex -
            offsetSize + 1, lastIndex))
 7         for (change : block.changes) do
 8             contentChanges.add(change)
 9             if (change.position > lastPosition + 1) then
10                 enrichContent(contentChanges, content, lastIndex +
                    1, change.position))
11             end
12             lastPosition := change.position
13         end
14         enrichContent(contentChanges, content, lastIndex,
            lastIndex+offsetSize))
15     end
16     return blocks
17 Function enrichContent(contentChanges, content, start, end):
18     for index ← start to end do
19         type = UNMODIFIED
20         newChange := new Change(type, content[index], index)
21         contentChanges.add(newChange)
22     end
```

Algorithm 7: Объединение блоков изменений с контентом.

2.8.3 Нахождение конфликтных ситуаций

Дано:

Измененный контент файла, номер версии контента,который был получен пользователем на момент редактирования.

Ожидаемый результат:

Измененный контент файла.

Если других изменений не было, то совпадает с теми данными, которые были получены от пользователя. Если же были найдены другие изменения этого контента, то либо разные изменения будут совмещены в одно, либо пользователю будет предоставлена возможность редактирования возникших конфликтов.

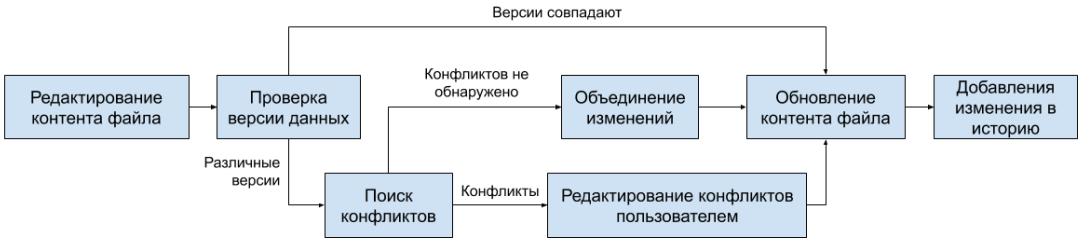

Рис. 2.6: Процесс изменения контента файлов.

Если других изменений контента не было найдено, то текущее изменение можно применять напрямую к данным.

Иначе, при наличии других, возможно конфликтующих, изменений, производится процесс поиска конфликтных ситуаций.

Если конфликтных ситуаций обнаружен не было, то достаточно просто объединить все изменения воедино и применить их к общей версии контента.

Если же при проверка были обнаружены конфликтные ситуации, то пользователю предоставляется возможность разрешить данные конфликты.

Поиск конфликтных ситуаций

Требуется построить проекцию общих данных для разных изменений (включая текущее). После вычисляется общий список изменений, которые были сделаны как в текущем процессе редактирования, так и в стороннем изменении, с которым возможны конфликты. На основе этих изменений вычисляется новый список изменений, который необходимо применить, чтобы интегрировать в проект все модификации.
Эти изменения анализируются на наличие пересечений. Если пересечений нет, то изменения двух версий объединяются в одно и применяются к общему контенту. Иначе, пользователю предоставляется список конфликтных блоков контента и варианты их разрешения.

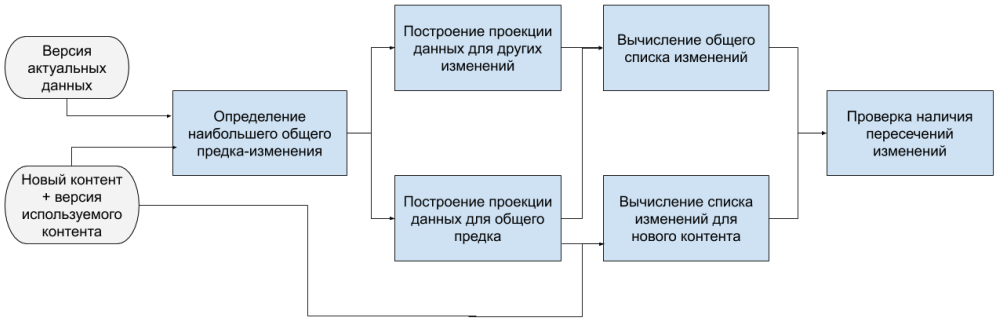

Рис. 2.7: Процесс поиска конфликтных ситуаций.

Разрешение конфликтных ситуаций

При наличии конфликтных ситуаций пользователь может принять или отклонить каждую часть как его изменений, так и сторонних. После того, как все конфликты разрешены, пользователь может редактировать контент повторно, для разрешения смысловых пересечений. Затем, новый отредактированный контент сохраняется в системе.

Объединение нескольких изменений в одно

Для получения нового списка изменений, которые должны примениться к актуальной версии контента с возможностью интегрирования в проект модификаций как текущего процесса редактирования, так и стороннего изменения, требуется объединить измнения в общий список. На основе общего списка изменений требуется вычислить контент, который будет содержать все изменения, которые происходили с контентом.

Имея актуальную версию данных и проекцию, построенную на основе общего списка изменений, можно вычислить новый список изменений, которые требуется применить к актуальной версии, чтобы все изменения были сохранены.

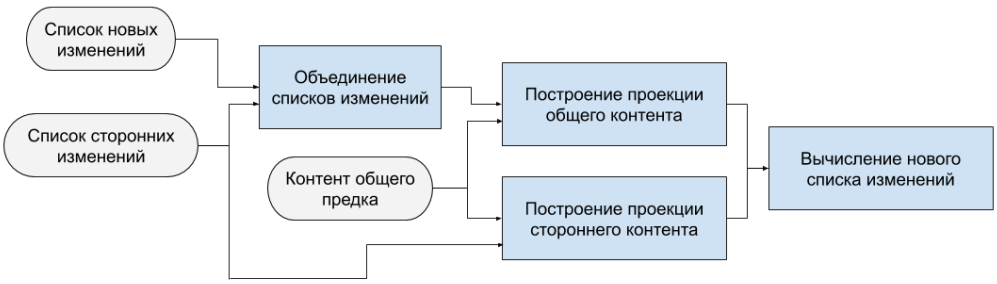

Рис. 2.8: Процесс объединения нескольких изменений в одно.

2.8.4 Построение проекции данных по идентификатору версии

Для построения проекции, требуется извлечь всю цепочку связаных *Commit* вплоть до корневого элемента. Затем, последовательно применяя эти изменения к изначально заданной пустой проекции проекта, мы будем тем самым получать проекцию данных на каждый период времени, пока не дойдем до последнего, для которого результат проекции и будет искомым.

```
   Data: commit identifier to make data projection
   Result: data projection
 1 Function main(commitId):
 2 |   projection := Projection.empty()
 3 |   commits := extractRelatedCommits(commitId)
 4 |   commits.sort(by id)
 5 |   for (commit : commits) do
 6 |   |   projection.update(commit)
 7 |   end
 8 |   return projection
 9 Function extractRelatedCommits(commitId):
10 |   allCommits := commitRepo.findAll()
11 |   result := []
12 |   lastParentId := allCommits[commidId].parentId
13 |   result.add(allCommits[commitId])
14 |   while (lastParentId ≠ null) do
15 |   |   result.add(allCommits[lastParentId])
16 |   |   lastParentId := allCommits[lastParentId].parentId
17 |   end
18 |   return result
```

Algorithm 8: Построение проекции данных по идентификатору версии.

2.9 Инфраструктура веб-платформы

Запуск даже небольшой веб-системы может занять большое количество времени, необходимого для подготовки и настройки инфраструктуры. А также возникает ряд вопросов, и все из них надо решить:

- Где взять инсталлятор и можно ли доверять этому источнику данных?
- Кросс-платформенное ли приложение?
- Есть ли зависимости, которые требуется установить для функционирования нашего приложения?
- Как запускать приложение?
- Как следить за потребляемыми ресурсами приложения и очистить их после завершения работы?
- Каким образом можно обновить версию работающего приложения?

В данной главе будут рассмотрены применяемые в работе технологии технологии, которые помогают разрешить поставленные выше вопросы.

2.9.1 Контейнеризация

Контейнерезация - метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. Для пользователя эти экземпляры (контейнеры) полностью идентичны отдельным экземплярам операционной системы с установленным по требованию программным обеспечением.
Ядро позволяет обеспечить полную изолированность каждого из контейнеров, поэтому приложения, запущенные в разных контейнерах не могут воздействовать ни друг на друга, ни на хостовую операционную систему. Под хостом в данной случае подразумевается операционная система, которая позволяет запускать контейнеры.
При аппаратной виртуализации эмулируется аппаратное окружение и может быть запущено большое количество различных программных решений,

в то время как контейнеры используют общее ядро хостовой машины. Также при контейнеризации отсутствуют дополнительные ресурсные расходы на эмуляцию достаточно ресурсоемкого виртуального оборудования для запуска программных систем, которая характерна для аппаратной виртуализации.

Изоляция контейнеров

В качестве средства изоляции контейнеров выступают процессы. При исполнении программ управляющий блок процессора считывает инструкции для выполнения из памяти, например из регистра процесса. При смене процесса происходит переключение контекста, тем самым делаетмся снимок текущего состояния регистра процесса. Именно слепок регистра процессов и является элементом изоляции.
Пространство имён (namespace) - функция ядра Linux, которая позволяет изолировать и виртуализировать глобальные системные ресурсы множества процессов [10]. Рассмотрим основные виды пространств имен на примере ядра Linux:

Таблица 2.15: Пространства имен.

Namespace	**Версия ядра**	**Год**
Файловая система (Mount)	2.4.19	2002
UTS	2.6.19	2006
ID процессов (PID)	2.6.24	2008
Сети (Network)	2.6.29	2009
Межпроцессорное взаимодействие (IPC)	2.6.30	2009
Пользовательские ID (User)	3.8	2013

Для всех типов пространств имен верно: каждый процесс связан с пространством имён и может видеть или использовать только ресурсы, связанные с этим пространством имён, и, где это применимо, потомками пространств имён.
Описание пространств имен:

1. Файловая система (Mount)
   Пространство имён файловой системы – независимое дерево файловой системы, которое ассоциированно с определенной группой процессов.

2. UTS (Unix Time Sharing)
   Изоляция двух элементов системы, которые относятся к системному вызову uname(). Позволяет каждому контейнеру иметь свои имена хоста и домена.

3. ID процессов (PID)
   Изолирует пространство ID процессов. Процессы в различных пространствах могут иметь одинаковые ID. Однако в пространстве имён хоста должны иметь уникальные идентификаторы.
   Процесс с PID равным 1 является инициализирующим процессом пространства.

4. Сети (Network)
   Изоляция систем ресурсов, связанных с сетями. Для обеспечения доступа во внешнюю сеть между физическим и виртуальным устройством из разных пространств создается мост.

5. Межпроцессное взаимодействие (IPC)
   Включает в себя семафоры, разделяемую память и очереди сообщений с уникальными идентификаторами (ID) Позволяет взаимодействующим между собой процессам обращаться к общему ресурсу.

6. Пользовательские ID (User)
   Изолируют ID пользователей и групп, корневой каталог и ключи. Пространства имён могут быть вложенными, максимальное количество уровней вложенности равно 32.

Пространства имен дают процессам, запущенным в контейнерах, иллюзию, что они имеют свои собственные ресурсы. Основная цель –предотвращение вмешательства процессов одного контейнера в работу других контейнеров, а также работу хостовой машины.

На основе выше изложенных типов пространств имен построена классификация классификаци типов изоляции:

1. Изоляция файловой системы – пространство предоставляет процессам каждого контейнера различное представление дерева файловой системы и ограничивает все события монтирования, происходящие внутри контейнера.

2. Изоляция UTS – каждый контейнер может иметь свое собственное доменное имя и имя контейнера.

3. Изоляция процессов – изоляция процессов путем переноса процессов, запущенных в контейнере, в собственное пространство имен и ограничения прав и видимости процессов, запущенных в других контейнерах и на самой хостовой машине. Пространство имён PID иерархичное, следовательно процесс может видеть только процессы в своем собственном пространстве имен, либо в его «дочернем».

4. Изоляция сетей –каждому контейнеру создается независимый сетевой стек при помощи пространства имён сети. Таким образом, каждый контейнер имеет свои собственные IP-адреса, таблицы IP-маршрутизации, сетевые устройства и т. д.

5. Изоляция IPC – процесс в пространстве имен IPC не может писать или читать IPC ресурсы, принадлежащие другому пространству имен.

6. Изоляция пользователей – пользователь остается привилегированным внутри контейнера и при этом непривилегированным внутри хостовой машины.

Механизмы контейнеризации

Существуют реализации контейнеризации, которые ориентированы на создание полноценных экземпляров операционных систем (Solaris Containers, контейнеры Virtuozzo), так и варианты, которые сфокусированы на изоляции отдельных сервисов с минимальным операционным окружением (Docker).

Таблица 2.16: Механизмы контейнеризации.

Механизм	**ОС**	**ФС**	**ВВ**	**П**	**ЦПУ**	**С**
chroot	UNIX	частично	-	-	-	-
Docker	UNIX Windows macOS	+	+	+	+	+
Solaris Containers	Solaris	+	-	+	+	+
Virtuozzo Containers	Linux Windows	+	+	+	+	+

где под ФС подразумевается изоляция файловой системы, ВВ - лимиты на ввод/вывод, П - лимиты на память, ЦПУ - квоты ЦПУ, Сеть - изоляция сети.

В данной работе предпочтение было отдано Docker в связи с его легковесностью, большим функционалом и огромным сообществом пользователей [11].

Docker-контейнеризация позволяет решить следующие вопросы:

- Поставка дистрибутива
  Установка инсталятора, лицензия, пакет.
  Docker-образ – то, что поставляется (содержит в себе все необходимое для запуска приложения).

- Эксплуатация
  Запуск, установка, конфигурирование, обновление, кросс-платформенность,

остановка, очистка ресурсов, безопасность, конфликты ресурсов.
Достигается за счет запуска программного обеспечения в изолированном окружении.
Docker-контейнер – то, что в итоге эксплуатируется (для упрощения настройки, запуска, использования приложения и очистки его ресурсов)

2.9.2 Непрерывная интеграция системы

Непрерывная интеграция (Continuous Integration) – практика разработки программного обеспечения. Она заключается в постоянном слиянии рабочих версий продукта в общую основную ветвь разработки и выполнении автоматизированной сборки проекта для того, чтобы как можно скорее можно было выявить потенциальные дефекты и решения интеграционных проблем системы.
Для применения практики необходимо выполнение следующих требования:

- Программное обеспечение должно храниться в репозитории системы контроля версий.
- Должны быть автоматизированы операции копирования программного обеспечения из репозитория, его сборки и тестирования

Преимущества применения данной практики:

1. Быстрее выявляются и исправляются проблемы интеграции.
2. Постоянный процесс тестирования новых версий продукта.
3. Наличие актуальной стабильной версии продукта.

Также непрерывная интеграция имеет следующие недостатки:

1. Затраты на поддержку работы системы непрерывной интеграции.
2. Дополнительные вычислительные ресурсы.

В данной работе использование системы непрерывной интеграции позволит автоматизировать процессы сборки, тестирования и создания нового docker-образа для обновления стенда веб-платформы.
Существует большое количество инструментов для непрерывной интеграции системы, в основном они подразделяются на две категории:

- Локальные (GitLab CI, TeamCity, Jenkins)
- Облачные (BitBucket Pipelines, Heroku CI, Travis)

С целью минимизации потребляемых ресурсов в данной работе была выбрана облачная CI система Travis.

Пайплайн сборки системы:

1. Сборка и тестирование программного обеспечения.
2. Создание и публикация нового docker-образа.
3. Деплой новой версии программного обеспечения в платформу.

2.9.3 Базы данных

В этом разделе описано сравнение основных типов хранения данных. Для эффективности работы с данными системы требуется выбрать наиболее подходящее решение. В данной работе наиболее важной характеристикой, которой должна обладать база данных, является целостность хранимых данных.

Сравнительный анализ SQL и NoSQL решений

Далее предсатвлены сравнительные характеристики различных подходов к хранению данных [12]:

1. Язык запросов:
   РСУБД используют единый SQL-стандарт. Каждая NoSQL база данных реализует свой способ работы с данными.

2. Структура данных:

   РСУБД обычно используется для однозначно определенных структур данных, которые не будут часто подвергаться изменениям. NoSQL же выделяется здесь своей возможностью хранения больших объёмов неструктурированной информации. Она не накладывает ограничений на типы хранимых данных. Более того, при необходимости в процессе работы можно добавлять новые типы данных.

3. Масштабируемость:

   Оба решения можно масштабировать вертикально (путём увеличения системных ресурсов). Однако, решения NoSQL обычно предоставляют простые способы горизонтального масштабирования (как пример, создание кластера из нескольких нод).

4. Принципы ACID:

   Большинство NoSQL решений не имеют даже частичной поддержки ACID. РСУБД же в свою очередь соответствуют требованиям ACID (Atomicity, Consistency, Isolation, Durability — атомарность, непротиворечивость, изолированность, долговечность), что позволяет обеспечить целостность хранимых данных.

Характеристика	**SQL**	**NoSQL**
Язык запросов	Единый структурированный язык	Меняется в зависимости от подхода к хранению данных
Структура данных	Однозначно определенная структура данных со связями	Документы
		Пары <key, value>
		Графы
Масштабируемость	Вертикальная	Вертикальная + Горизонтальная
Принципы ACID	Соответствуют	Частичное соответствие

Таблица 2.17: Анализ SQL и NoSQL решений

В данной работе было отдано предпочтение реляционным базам данным, так как основополагающим фактором выбора является поддержание целостности данных. В качестве решения выбран PostgreSQL, как бесплатное программное обеспечение с наиболее продвинутым функционалом и огромным сообществом, развивающим эту систему хранения данных.

Также в данной работе имеется возможность миграции данных, что позволяет при необходимости обновить структуру базы, и в случае ошибки вернуть все в прежнее состояние.

2.10 Характеристики качества

ISO-9126 – международный стандарт, который определяет оценочные характеристики качества программного обеспечения [13].
Стандарт включает в себя 4 раздела, которые описывают следующее:

1. модель качества
2. внешние метрики качества
3. внутренние метрики качества
4. метрики качества в использовании

Отличительной чертой хорошего программного обеспечения является полное или частичное соответствие этому стандарту. В данной работе рассматривается только первый раздел, описывающий модель качества ПО. Модель качества классифицирует качество ПО на 6 характеристии. В свою очередь каждая характеристика также обладает некоторым набором подхарактеристик, более детализирующих ее.

1. Функциональность
   Набор атрибутов, которые характеризуют соответствие функциональных возможностей ПО списку функциональности, которая требуется пользователю.
   Подхарактеристики:
   - Применимость – относится к уместности (спецификации) функций программного обеспечения.
   - Точность – корректность выполняемых пользователем действий в системе.
   - Совместимость – способность программного обеспечения взаимодействовать другими компонентыми или системами.
   - Соответствие – соблюдение основных отраслевых или правительственных законов и принципов.

- Безопасность – предотвращение несанкционированного доступа к функциям программного обеспечения.

Данная система реализована с соблюдением указанных ранее бизнес-требований. Никаких отраслевых ограничений на веб-платформу не накладывается. Также реализована система защиты от неразрешенного доступа к веб-платформе, которая описана в одном из следующих параграфов.
Следовательно, система полностью соответствует данной характеристике программного обеспечения.

2. Надежность
   Способность программного обеспечения поддерживать уровень качества функицонирования при определенных условиях в течение определенног опериода времени.

   - Уровень завершенности – касается частоты программных сбоев.
   - Отказоустойчивость – способность программного обеспечения выдержать внутренние и внешние отказы и ошибки.
   - Восстанавливаемость – способность вернуть вышедшую из строя систему к полноценному функционированию с восстановлением всех данных и сетевых соединений.

   В данной системе полностью реализован указанный ранее функционал с последующим его тестированием. Большая часть исключительных ситуаций, приводящих к ошибкам и сбоям системы, обрабатывается и отдается пользователю в удобочитаемом виде, тем самым не приводит к выходу системы из строя. Данные системы хранятся в выделенном хранилище данных, тем самым имеется возможность восстановления полностью функционирующей системы. Имеется возможность горизонтального масштабирования системы, что позволяет выдержать большие нагрузки пользователей и улучшит отказоустойчивость системы.
   Можно сделать вывод, что данная система полностью удовлетворяет требованиям рассматриваемой характеристики.

3. Удобство использования

   Рассматривается только в отношении существующего функционала программного обеспечения и относится к простоте использования определенных функций.

   - Понятность – определяет легкость понимания функционала системы пользователем.
   - Обучаемость – описывает прилагаемые усилия по обучению различных типов пользователей: новичков, экспертов в области и т.д.
   - Простота использования – способность программного обеспечения легко управляться конкретным пользователем в данном окружении.

   Веб-сервис реализован с соблюдением архитектуры REST. Система имеет простой веб-интерфейс, который реализован с учетом важности структурирования данных, минимизации действий не в угоду пониманию. Интерфейс выглядит достаточно аккуратным и не перегружает пользовательское внимание. Веб-интерфейс был протестирован несколькими пользователями, особых нареканий выявлено не было. Можно считать, что в некоторой мере данная характеристика соблюдается.

4. Эффективность

   Описывает соотношение между объемом используемых системных ресурсов и качеством функционирования программного обеспечения при определенных условиях.

   - Временная эффективность – характеризует время отклика системы.
   - Эффективность ресурсов – характеризует объемы используемых ресурсов: оперативная память, процессор, дисковое пространство и использование сети.

Среднее время отклика системы для проекта несущих стен здания на оборудовании t2.micro облачного провайдера AWS EC2 в 95% случаев составляло не более 300 мс. Благодаря хорошей временной эффективности пользователь всегда будет иметь представление о том, работает ли сейчас функционал системы.
Для функционирования одного экземпляра сервиса будет достаточно 512 Мб RAM, 1-core процессора с тактовой частотой выше 1.5 ГГц, дисковое пространтсво в зависимости от количества проектов и их размера, производительность сети от 10 Мбит/с.

5. Поддерживаемость
   Описывает объем работ, которые требуются для модифицирования программного обеспечения.

   - Возможность анализа – характеризует возможность идентифицировать основную проблему, которая привела к программному сбою.
   - Изменяемость – характеризует количество усилий и времени на изменение программного обеспечения.
   - Стабильность – характеризует чувствительность к изменению системы.
   - Тестируемость – характеризует усилия, необходимые для проверки корректности работы системы после внесения изменений в нее.

   Данная веб-система реализована на основе принципов SOLID и DDD (предметно-ориентированное проектирование), соответственно для изменения программного обеспечения не требуется тратить большое количество усилий. Также в системе ведется логирование всех ошибок, что позволяет достаточно быстро понять основные проблемы, которые могли привести к сбою системы. К внешнему API системы присутствует документация. Имеется тестовое покрытие основного функционала системы. Более подробно тестовое покрытие описано ниже.

6. Портируемость

   Описывает, насколько хорошо программное обеспечение может адаптироваться к изменениям окружения или его требованиям.

   - Адаптируемость – характеризует способность системы к изменению окружения или его требований.
   - Возможность установки – характеризует усилия, которые требуются для установки программного обеспечения.
   - Заменяемость – характеризует насколько легко заменить данный программный компонент на другой в данной среде.

   Данная система запускается в выделенном окружении с помощью docker-контейнеризации. Что при наличии окружения для запуска docker-образов системы нивелирует вопросы сложности адаптируемости, установки и заменяемости.

Тестовое покрытие системы

Анализ тестового покрытия системы выполнен с помощью бесплатной библиотеки *JaCoCo* [14]. *JaCoCo* предоставляет возможность анализа тестового покрытия кода в средах на основе виртуальной машины Java. Основные возможности инструмента *JaCoCo*:

- Анализ покрытия инструкций, логических разветвлений, строк кода, методов и классов, а также возможность нахождения циклических зависимостей.
- Основан на байт-коде Java и поэтому работает также без исходных файлов.
- Несколько форматов отчетов (HTML, XML, CSV).
- Наличие различных вариантов подключения анализатора тестового покрытия: Gradle, Maven, TeamCity, Jenkins, SonarQube и другие плагины.
- Совместим со всеми выпущенными версиями Java.

Element	Missed Instructions	Cov.	Missed Branches	Cov.
valeriy.knyazhev.architector		37%		n/a
valeriy.knyazhev.architector.application.commit		83%		76%
valeriy.knyazhev.architector.application.commit.command		54%		0%
valeriy.knyazhev.architector.application.commit.data		0%		n/a
valeriy.knyazhev.architector.application.commit.data.changes		85%		63%
valeriy.knyazhev.architector.application.commit.data.history		66%		n/a
valeriy.knyazhev.architector.application.project		95%		95%
valeriy.knyazhev.architector.application.project.command		97%		n/a
valeriy.knyazhev.architector.application.project.file		19%		0%
valeriy.knyazhev.architector.application.project.file.command		63%		n/a
valeriy.knyazhev.architector.application.project.file.conflict		51%		43%
valeriy.knyazhev.architector.application.project.file.conflict.command		70%		n/a
valeriy.knyazhev.architector.application.project.file.conflict.data		40%		4%
valeriy.knyazhev.architector.application.project.file.conflict.exception		100%		n/a
valeriy.knyazhev.architector.application.security		0%		0%
valeriy.knyazhev.architector.application.user		9%		0%
valeriy.knyazhev.architector.application.util		0%		0%
valeriy.knyazhev.architector.configuration		38%		n/a
valeriy.knyazhev.architector.domain.model		100%		n/a
valeriy.knyazhev.architector.domain.model.commit		90%		84%
valeriy.knyazhev.architector.domain.model.commit.projection		89%		50%
valeriy.knyazhev.architector.domain.model.project		71%		73%
valeriy.knyazhev.architector.domain.model.project.file		73%		32%
valeriy.knyazhev.architector.domain.model.user		78%		47%
valeriy.knyazhev.architector.domain.model.util		0%		0%
valeriy.knyazhev.architector.domain.model.util.serialization		48%		25%
valeriy.knyazhev.architector.port.adapter.resources		72%		n/a
valeriy.knyazhev.architector.port.adapter.resources.commit		94%		0%
valeriy.knyazhev.architector.port.adapter.resources.project		95%		83%
valeriy.knyazhev.architector.port.adapter.resources.project.file		100%		100%
valeriy.knyazhev.architector.port.adapter.resources.project.file.conflict		100%		100%
valeriy.knyazhev.architector.port.adapter.resources.project.file.conflict.model		100%		n/a
valeriy.knyazhev.architector.port.adapter.resources.project.file.model		78%		n/a
valeriy.knyazhev.architector.port.adapter.resources.project.file.request		92%		n/a
valeriy.knyazhev.architector.port.adapter.resources.project.model		83%		n/a
valeriy.knyazhev.architector.port.adapter.resources.project.request		100%		n/a
valeriy.knyazhev.architector.port.adapter.resources.user		100%		100%
valeriy.knyazhev.architector.port.adapter.util		30%		25%
Total	3,778 of 11,621	67%	274 of 618	55%

Рис. 2.9: Тестовое покрытие системы.

Сгенерированный отчет показал качество покрытия инструкций кода, равное 67%, а качество покрытия логических ветвлений, равное 55%.

Безопасность

Для предотвращения неразрешенного доступа к функционалу данной системы реализована система безопасности. В начале пользования системой каждому пользователю требуется пройти процесс авторизации. Во время авторизации для пользователя создается новая сессия, идентификатор которой передается в запросах между веб-интерфейсом и backend-сервером в cookie для валидации доступа. Каждая сессия имеет максимальное время жизни, равное 10 минутам. В случае неактивности пользователя в системе на протяжении промежутка времени, который превышает указанное ранее

максимальное время жизни сессии, доступ к системе запрещается и пользователю требуется повторно произвести процесс авторизации.

Для возможности горизонтального масштабирования системы информация о текущей сессии сохраняется в базу данных. Что позволяет разным запущенным инстансам системы получить актуальные данные о валидности пользовательской сессии работы в системе.

Также для возможности использования внешнего API системы пользователь может получить JSON Web Token, который можно использовать для аутентификации в системе.

JSON Web Token – открытый стандарт (RFC 7519) для создания токенов доступа, который основан на формате JSON. Токены создаются на стороне сервера, подписываются секретным ключом и передаются клиенту. Данный токен в дальнейшем может быть использован для подтверждения своей личности.

Глава 3

Заключение

В данной работе получены следующие результы:

1. Реализована веб-система, которая предоставляет следующие возможности:
    - управление жизненным циклом архитектурных проектов
    - отслеживание изменений проекта во времени
    - одновременное внесение изменений в проект несколькими пользователями с возможностями разрешения возникших конфликтных ситуаций
2. Веб-система предоставляет внешнее API и документацию к нему.
3. Реализован веб-интерфейс для взаимодействия с системой.
4. Написаны скрипты сборки и тестирования системы.
5. Написаны скрипты настройки окружения для запуска docker-контейнеров.
6. Написаны скрипты сборки docker-образа системы и его запуска.

Возможные направления развития данной работы:

- Возможность построения 3D модели проекта.
- Возможность создания связанных между собой файлов с BIM представлениями объектов.

Литература

1. *GlobalData.* **Global construction output growth to reach 3.4% in 2019** // Публикация на www.globaldata.com. 11 April 2019.

2. *Ar. Mustakeem Raza Khan, Prof. S.K Gupta, Ar. Rakesh Kumar.* **Role of Computer's Technology: Architectural Design** // International Journal for Research in Applied Science & Engineering Technology (IJRASET)

3. *Karen M. Kensek, Douglas E. Noble.* **Building Information Modeling: BIM in Current and Future Practice (1st ed.)** // 2014 Hoboken, New Jersey: John Wiley

4. *Arayici, Y, Coates, P, Koskela, LJ, Kagioglou, M, Usher, C.* **Technology adoption in the BIM implementation for lean architectural practice)** // Technology adoption in the BIM implementation for lean architectural practice, Automation in Construction, 2011. pp. 189-195.

5. *McAuley, B., Hore, A. and West R.* **BICP Global BIM Study - Lessons for Ireland's BIM Programme** // Construction IT Alliance (CitA), 2017

6. *UK Government.* **Level 3 Building Information Modelling - Strategic Plan** // Digital Built Britain, February 2015

7. *Президент Российской Федерации Путин В.В.* **Поручение ПР-1235** // 19.07.2018.

8. *buildingSMART International Ltd* **Industry Foundation Classes Release 4** // section "Rooted entities" http://www.buildingsmart-tech.org/ifc/IFC4/final/html/

9. *EUGENE W. MYERS* **An O(ND) Difference Algorithm and Its Variations** // Department of Computer Science, University of Arizona, Tucson

10. *Wikipedia* **Пространство имён (Linux)** // https://ru.wikipedia.org/wiki

11. *Docker, Inc.* **Docker Documentation** // https://docs.docker.com

12. *Kosovare Sahatqija ; Jaumin Ajdari ; Xhemal Zenuni ; Bujar Raufi ; Florije Ismaili* **Comparison between relational and NOSQL databases** // 2018 41st

International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)

13. *ISO/IEC* **Information technology – Software product quality – Part 1: Quality model** // INTERNATIONAL STANDARD 20.03.2000

14. *EclEmma team* **JaCoCo documentation** // Official provider's documentation https://www.jacoco.org/jacoco/trunk/doc/