

Министерство образования и науки Российской Федерации  
ФГБОУ ВО «Российский химико-технологический университет  
имени Д.И. Менделеева»

Факультет информационных технологий и управления

Кафедра информационных компьютерных технологий

## **Выпускная квалификационная работа**

**на тему:**

**«Разработка веб-ориентированных динамических  
пользовательских интерфейсов для распределенных  
систем инженерного анализа»**

**Заведующий кафедрой**

д.т.н., профессор

Э.М. Кольцова

**Руководители**

д.т.н., профессор

Э.М. Кольцова

к.ф.-м.н., доцент

А.П. Соколов

**Обучающийся**

Н.Д. Кириллов

**Москва, 2017**



## **АННОТАЦИЯ**

В данной работе представлено создание веб-ориентированной графической подсистемы формирования графических пользовательских интерфейсов для использования в рамках сложных вычислительных систем с организованным удаленным доступом для прикладных вычислительных задач микромеханики композиционных материалов.

Объем квалификационной работы составил 71 страницу, которые содержат 29 рисунков, 1 таблицу, 24 источника литературы, 6 приложений. Работа состоит из введения, трех основных глав, и выводов по проделанной работе.

Ключевые слова: GUI (графический пользовательский интерфейс), методы автоматизации формирования GUI, динамические адаптивные GUI, клиент-серверное приложение.

## **ABSTRACT**

In this paper, the creation of a web-based graphical subsystem for the formation of graphical user interfaces for use in complex computer systems with organized remote access for applied computational tasks of micromechanics of composite materials is presented.

The qualification work volume was 71 pages, which contain 29 figures, 1 table, 24 literature sources, 6 applications. The work consists of an introduction, three main chapters, and conclusions on the work done.

Keywords: GUI (graphical user interface), methods for automating the formation of GUI, dynamic adaptive GUI, client-server application.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ.....</b>	<b>6</b>
<b>ЛИТЕРАТУРНЫЙ ОБЗОР .....</b>	<b>8</b>
1.1 Понятие графического пользовательского интерфейса (GUI).....	8
1.1.1 Архитектура .....	8
1.1.2 История развития .....	9
1.1.3 Адаптивность и динамика .....	11
1.1.4 Методология и языки описания .....	13
1.1.5 Проектирование .....	15
1.2 Интерактивные веб-приложения .....	17
1.2.1 Основные программные компоненты .....	17
1.2.2 Преимущества и недостатки .....	21
1.2.3 Связь с GUI.....	23
1.2.4 Производительность.....	24
<b>ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .....</b>	<b>27</b>
2.1 Выбор языка серверной разработки .....	27
2.2 Выбор средств визуальной разработки .....	29
2.3 Использование GIT .....	34
2.3.1 Создание репозитория .....	35
2.3.2 Добавление файлов .....	35
2.3.3 Ветки .....	36
2.4 Текстовый редактор Brackets .....	37
2.5 Веб-клиент системы .....	38
2.5.1 Архитектура .....	38
2.5.2 Основные модули .....	41
2.5.3 Типы представлений .....	42
2.5.4 GBSE технология .....	43
<b>РАСЧЁТНО-ПРАКТИЧЕСКАЯ ЧАСТЬ .....</b>	<b>47</b>
3.1 Описание модуля построения древовидного отображения .....	47
3.1.1 Серверная часть .....	47
3.1.2 Визуальная часть .....	50

3.2 Описание модуля построения сгруппированного отображения .....	52
3.2.1 Серверная часть .....	52
3.2.2 Визуальная часть .....	53
3.3 Описание модуля построения графических представлений вычислительных инструментов .....	54
3.3.1 Выбор и настройка плагина для реализации .....	54
3.3.2 Серверная часть .....	55
3.3.3 Визуальная часть .....	56
3.4 Тестирование созданных GUI в рамках существующей программной системы: выявление особенностей построения, оптимизация используемых подходов .....	60
3.4.1 Обобщение кода стилового оформления .....	61
3.4.2 Создание общих шаблонов .....	62
3.4.3 Структурирование кода скриптовой части клиента .....	63
3.5 Руководство программиста .....	66
3.5.1 Назначение и условия применения программы .....	66
3.5.2 Характеристика программы .....	66
3.5.3 Обращение к программе .....	67
3.5.4 Входные и выходные данные .....	67
<b>ВЫВОДЫ.....</b>	<b>68</b>
<b>СПИСОК ЛИТЕРАТУРЫ .....</b>	<b>69</b>
<b>ПРИЛОЖЕНИЕ.....</b>	<b>72</b>
Приложение 1 Листинг программного модуля древовидного представления .....	73
Приложение 2 Листинг программного модуля сгруппированного представления .....	75
Приложение 3 Листинг программного модуля графо-ориентированной модели .....	78
Приложение 4 Листинг программных модулей оптимизации кода .....	85
Приложение 5 Текст доклада .....	90
Приложение 6 Иллюстрационные материалы к докладу .....	94

## ВВЕДЕНИЕ

**Тема работы:** «Разработка веб-ориентированных динамических пользовательских интерфейсов для распределенных систем инженерного анализа».

Тема этой дипломной работы затрагивает сразу несколько областей научной деятельности и самое важное, что эти области, на сегодняшний день, являются очень перспективными и быстроразвивающимися. Актуальность выбранной мною темы обуславливается следующим:

- На данный момент область композиционных материалов, в которой активно работает распределенные вычислительные системы инженерного анализа, является очень развивающейся;
- Такая область программирования, как веб-разработка, последнее десятилетие является наиболее востребованной во всех сферах жизни, а в будущем, по оценкам многих исследований, веб-программирование станет основой для большинства проектов;
- Данная разработка может повысить уровень исследований, что в последствии сможет повлиять на другие отрасли: строительной, авиакосмической, медицинских назначений;

**Цель работы:** создать веб-ориентированную графическую подсистему формирования графических пользовательских интерфейсов для использования в рамках CAE систем с организованным удаленным доступом для прикладных вычислительных задач микромеханики композиционных материалов.

Соответственно, выбор данной темы также обуславливается этими причинами, а также наличием знаний и опыта разработки с похожими инструментами.

**Задачи,** которые поставлены в соответствии цели, можно сформулировать следующим образом:

- Изучить GBSE технологию (базовая графо-ориентированная технология разработки программных реализаций сложных вычислительных методов);
- Разработать специализированные веб-ориентированные программные модули в рамках существующей системы, обеспечивающие автоматизацию построения GUI на основе данных в БД;

- Разработать специализированные веб-ориентированные программные модули в рамках существующей системы, обеспечивающие автоматизацию построения графических представлений вычислительных инструментов ("решателей"), применяемых для решения задач, в рамках технологии GBSE;
- Провести тестирование созданных GUI в рамках существующей программной системы: выявить особенности построения, преимущества и недостатки используемых подходов, а также сформировать aINI форматы файлов исходных данных для последующего использования при создании программного обеспечения проведения расчетов эффективных характеристик композиционных материалов;

**Объект исследований:** технологии построения адаптивных веб-ориентированных GUI.

**Предмет исследований:** распределенная вычислительная система инженерного анализа GCD, которая ведет свою деятельность по изучению композиционных материалов.

**Практическая значимость** этой работы заключается в создании универсальной адаптивной графической подсистемы, которая сможет существенно повысить возможности и качество работы пользователей. Эту подсистему можно использовать таким образом, чтобы обеспечить графическими пользовательскими меню ввода исходных данных консольные приложения, которыми и являются большинство вычислительных программ.

Суть разработки состоит в том, что для реальных прикладных задач постоянно меняются требования. Если реализовано приложение со статическим GUI, то в последствии с увеличением числа функций — это может стать серьезной проблемой. Поэтому и требуется создавать динамическое адаптивное GUI.

# ЛИТЕРАТУРНЫЙ ОБЗОР

## 1.1 Понятие графического пользовательского интерфейса (GUI)

Для лучшего понимания термина графического пользовательского интерфейса (GUI) обратимся к нескольким источникам. На различных языках в это понятие вкладывают достаточно близкий по своему значению смысл.

Под графическим пользовательским интерфейсом понимают вид такого интерфейса, в котором основные элементы представляются в виде графических изображений на дисплее пользователя [1].

Графический интерфейс сейчас используется практически везде и выполняет функцию визуализации, что для многих процессов и исследований является очень важным. О таком же применении графического интерфейса размышляют и авторы иностранных статей. И конечно же, с этим сложно не согласиться [2].

### 1.1.1 Архитектура

Архитектура GUI достаточно проста в понимании. Она обладает иерархической структурой, которая, в свою очередь, состоит из контейнеров и виджетов. Контейнер по своей сути ограничивает структуру, а также окружает области виджетов. Контейнер может быть не виден пользователю, но существовать программно. Под виджетом же понимают любой программный или структурный блок, с которым пользователь может взаимодействовать интерактивно, то есть может работать с ним в реальном времени.

К контейнерам относят:

- Frame (рамка)
- Window (окно)
- Panel (область панели)
- Tab pane (область вкладок)
- Menubar (меню)
- Popup menu (всплывающее меню)



К виджетам относят:

- Button (кнопка)
- CheckBox (флаг)
- RadioButton (радиокнопка)
- Label (Метка)
- Tree (древовидный список)

Для каждого виджета создается свой обработчик событий, который выполняется при взаимодействии с пользователем. Кроме того, некоторые виджеты могут обрабатывать сразу несколько действий [3].

### 1.1.2 История развития

С появлением компьютера и реализацией на нем каких-либо вычислительных или графических операций, необходимо было иметь графический пользовательский интерфейс. Поэтому можно сказать, что история GUI начинается с появления первых вычислительных машин.

Взаимодействие человека как пользователя с интерфейсом должно осуществляться по определенным правилам. Форма и обращение пользователя зависит от нескольких факторов, и с течением времени эти факторы остаются неизменны:

- Вычислительная мощность и объем памяти очень влияют на пользовательский интерфейс. Соответственно с развитием технологий и повышением мощности, существенно повысилось качество графического пользовательского интерфейса;
- Активная разработка новых устройств ввода/вывода также привела к существенному скачку в области пользовательских интерфейсов, и в будущем это развитие только ускорится;
- Человек, который использует на протяжении многих лет компьютер, сам создает потребность и необходимость в развитии пользовательского графического интерфейса. Потребности пользователей подтолкнули вперед развитие UI (user interface);

Очевидно, что первые ЭВМ были очень неудобны с точки зрения пользователя. Когда в 50-х годах появилась возможность цифровой обработки данных, то это подтолкнуло развитие всей области. Появились клавиатуры и мониторы, которые и сейчас являются важными элементами работы с компьютером.

В 70-х годах начали появляться первые графически-ориентированные программы. Примером может служить командная строка. С увеличением возможностей графического интерфейса возникла проблема понимания программы. С этого времени началось обучение пользователей работе с графической составляющей программ. Теперь задачей специалистов, помимо прочего, являлось ознакомление и понимание графического интерфейса.

В 80-х годах сфера информационных технологий разрастается, соответственно увеличивается в несколько раз количество новых программ, что положительно сказывается на графическом отображении. Также, активно используют такой элемент управления, как мышь. Появляются такие элементы виджетов, как меню управления, флаги и т.д.

Затем, после появления первого компьютера с графической операционной системой, это сфера начала настолько быстро развиваться, что сейчас у каждого человека есть свой компьютер или ноутбук с очень хорошей графической составляющей, которая требует очень много ресурсов памяти и мощности, если сравнивать с ранними годами развития.

В будущем область построения GUI будет еще больше развиваться, подстраиваясь под реалии и потребности общества, что должно хорошо сказаться на качестве, а главное, функциональности современного графического интерфейса.

Именно такие идеи закладывает автор Maul M., который в своей статье описывает историю развития GUI и рассматривает будущие перспективы всей области, как с точки зрения пользователя, так и с точки зрения программного обеспечения [4].

На сегодняшний день, технологии, связанные с GUI, достигли определенных успехов в дизайне и формировании интерфейса. Современный вид графического интерфейса представлен на рисунке 1.1. И несомненно, с каждым годом скорость развития и появления новых интерфейсов увеличивается.

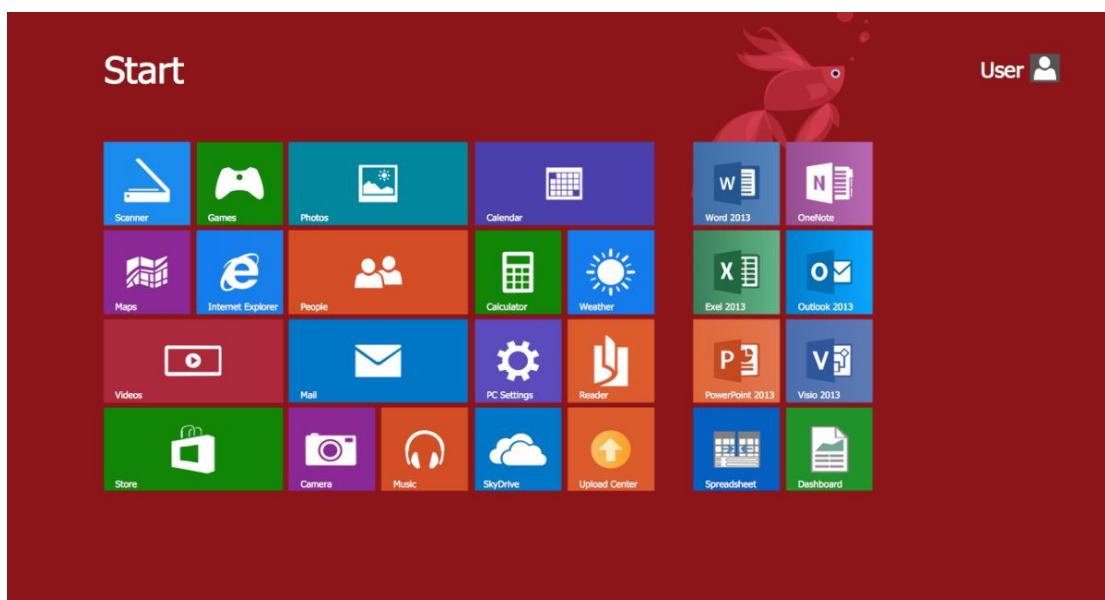


Рис. 1.1. Современный графический пользовательский интерфейс

### 1.1.3 Адаптивность и динамика

Под адаптивностью GUI понимают способность отдельных его элементов (виджетов) изменять свои свойства в зависимости от условий использования. Это понятие очень схоже с универсальностью программного обеспечения. Другими словами, адаптивность – это способность динамически изменяться в процессе работы.

Важно отметить, что не все свойства графического элемента могут изменяться, и, тем более, не все свойства влияют на графическое представление элемента на дисплее интерфейса пользователя. К основным адаптируемым свойствам относят:

- Форма и размеры элемента
- Цвет
- Шрифт
- Видимость
- Графические эффекты

Как правило, эти свойства изменяются при взаимодействии с пользователем, например, при нажатии левой кнопкой мыши по графическому элементу или компоненту.

Далее определим несколько способов адаптации графических элементов,

выделим 3 основных подхода, которые наиболее часто применяются разработчиками программного обеспечения:

- Первый подход позволяет осуществить более тонкую настройку для отдельного пользователя, то есть для двух пользователей графический интерфейс может выглядеть по-разному. Например, в зависимости от статуса, можно отображать или не отображать дополнительные сведения;
- Второй подход заключается в том, что для отдельных групп компонентов, связанных похожими свойствами, определяются значения. Например, для всех кнопок определяется цвет и шрифт, вне зависимости от пользователя;
- Также зачастую используют смешанный подход, который сочетает в себе два предыдущих, то есть используют как индивидуальный, так и групповой подход;

В современном GUI для отдельного пользователя, работающего с индивидуальным графическим интерфейсом можно определить концепцию хранения значений параметров.

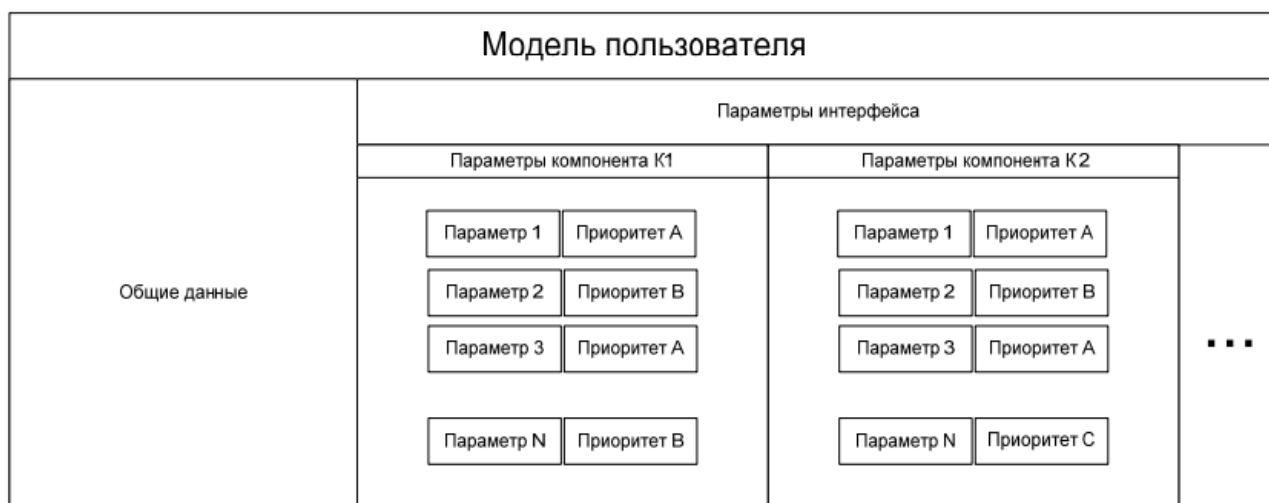


Рис. 1.2. Принцип хранения значений параметров интерфейса пользователя

На представленном рис.1.2 можно увидеть, что для каждого параметра, который задает пользователь, также задают приоритеты. Их задают для того, чтобы избежать возможных конфликтов между системой и пользователем. Например, для определенного элемента системой установлено свойство по умолчанию, а пользователь задал собственное свойство.

Очевидно, что пользовательские настройки должны иметь наивысший

приоритет в обычных случаях.

Чтобы избежать возможность возникновения конфликтов между пользователем и системой, ввели следующие функциональности:

- Добавлен механизм запроса пользователя на изменение отображения GUI при возникшем конфликте;
- Добавлена возможность расставлять приоритеты для свойств пользователей. Это уже отмечено выше;

Таким образом, мы выделили основные элементы и средства адаптивного пользовательского интерфейса и привели основные подходы автоматизации GUI, которые наиболее распространены в современном времени, что и соответствует взглядам авторов Ходакова В. Е. и Величко Ю. И. [5].

#### **1.1.4 Методология и языки описания**

При создании нового продукта, который связан с графическим интерфейсом, необходимо построить модель на абстрактном уровне. Это является первым фактором. Чем больше конкретности, тем лучше. Она является одним из важнейших естественных и полезных факторов.

Также на этом уровне выбирается язык программирования, который является 2 фактором. Очевидно, что реализация происходит на языке, который имеет четко определенные свойства: синтаксис, семантику, структуру и логику.

Третий важнейший фактор – это логика. Она определяет связь различных структурных блоков программы. От правильно-определенной логики модели зависит дальнейшая разработка.

Интеграция различных языков и моделей в формальные методы не является необычным явлением. Основная идея заключается в использовании различных особенностей и преимуществ различных по своему качеству соответствующих методов.

Иногда достаточно просто использовать различные формализмы для указания различных частей и различных свойств системы, но лучший эффект достигается, когда методы были полностью интегрированы таким образом, что есть формальные связи между ними. Это позволяет придерживаться полностью правильного развития [6].

Если говорить подробнее о языках программирования, то стоит отметить, что на сегодняшний день для реализации удобного и адаптивного графического интерфейса существует большой набор инструментов практически в каждом языке. Это может быть серверный язык, как пример Python, PHP и др. Также это языки веб-программирования, такие как HTML5, CSS3, JavaScript.

В каждом из языков описания GUI предустановлены необходимые библиотеки и плагины для лучшего взаимодействия с пользователем. Также важно отметить, что на протяжении многих лет языки меняли не только графический интерфейс пользователей, но и сами развивались и подстраивались под современные требования пользователей.

Одной из самых главных проблем является интеграция языка описания GUI в логику и структуру исследовательской системы. Эту проблему решают за счет использования нескольких языков.

Например, система работает на языке Python, через этот язык происходит передача данных с сервера. Такие языки как HTML5 и CSS3 представляют эти данные в удобном для пользователя виде [7].

Каждый из языков выполняет свою определенную задачу и вместе с другими языками образует систему представлений. Пример такого пользовательского представления продемонстрирован на иллюстрации 1.3.



Рис. 1.3. Пример интеграции GUI в мобильное устройство

### 1.1.5 Проектирование

В современном мире очень актуальной является тема проектирования GUI. Ведь с улучшением качества интерфейса пользователю будет гораздо легче работать с программой.

При знакомстве с программой, пользователь, в первую очередь, обращает внимание на интерфейсную часть приложения. Зачастую, пользователь оценивает качество программы по ее графическому интерфейсу, поэтому он должен быть эргономичен, согласно мнению автора статьи об использовании интерфейса Попова А.А. [8].

В проектировании GUI нет определенных стандартов, многие параметры зависят от специфики области применения: различные виды дизайнов, традиции и условия эксплуатации. Также для разных областей деятельности применяют свой стиль оформления, который является идентификатором для этой области. Стоит отметить, что важную роль играет так называемое прототипирование.

Под прототипированием понимают создание прототипа или макета будущего представления графического интерфейса. Это необходимо, для того, чтобы непосредственно перейти к созданию интерфейса, прототип которого был утвержден и разработан с помощью специализированных и общедоступных инструментов.

В большинстве случаев, первым этапом является разработка макетов с помощью таких инструментов.

К общедоступным инструментам можно отнести популярные разработки: Adobe Photoshop, Adobe Illustrator, CorelDraw, GIMP, Paint. У всех этих инструментов есть свои достоинства и недостатки, но все они не являются специализированными. Это означает, что с помощью них возможно создать макет или прототип, который подойдет для определенной задачи, но изначально эти инструменты были созданы для других целей.

Специализированные инструменты сейчас активно разрабатываются и внедряются в обычные и популярные системы использования. Например, в операционные системы или в стандартные браузеры персонального компьютера.

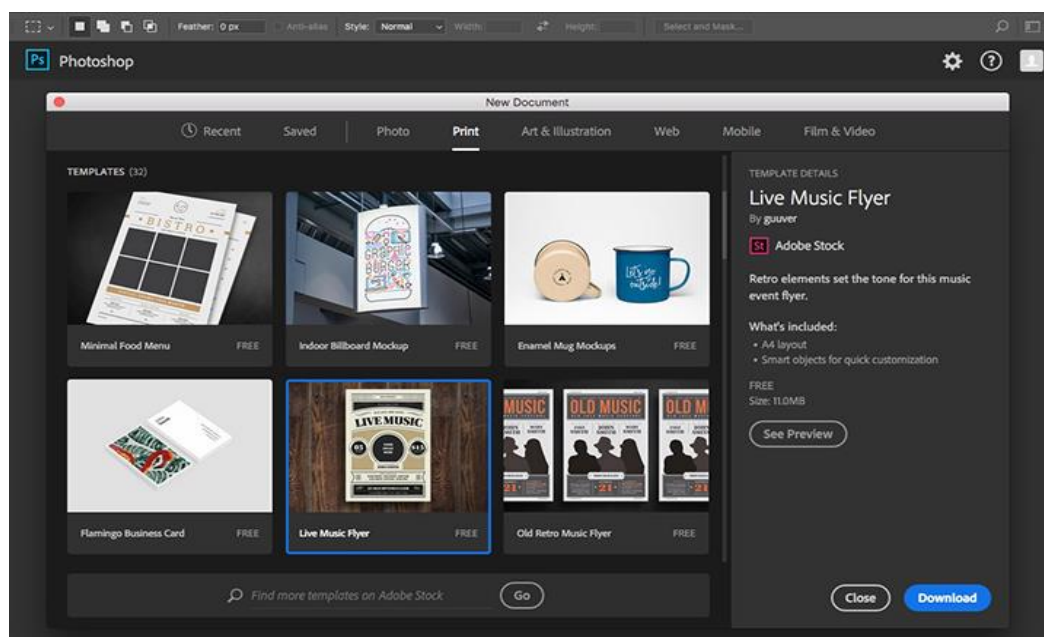


Рис. 1.4. Пример создания макета в программе Adobe Photoshop

Создание макетов или прототипов несет в себе огромную пользу как для разработчика, так и для будущего пользователя. Польза заключается в том, что разработчик может продумать все мелкие детали на стадии проектирования, и в будущем не возникнет больших проблем. На рис. 1.4 представлены макеты, которые несут в себе огромную пользу для всей области построения пользовательского интерфейса.

Также можно проверить удобство использования программой, например, на тестовом макете, с самым низким функциональным набором. Это поможет скорректировать направление по развитию программы и графического интерфейса.

Разработка макетов в отдельности от функционального подхода приводит к появлению специалистов разного класса деятельности, которые работают совместно, но каждый отвечает за свою отдельную часть. В целом это приводит к улучшению качества продукта.

По мнению автора Сливы М. В. курсы, связанные с IT разработкой и направлением следует включить изучение такой темы как прототипирование и применять во всех дисциплинах, связанных с разработкой программного обеспечения [9].



## 1.2 Интерактивные веб-приложения

Современные веб-приложения практически всегда являются интерактивными. Это объясняется тем, что взаимодействие с пользователем часто является основной задачей таких приложений. Для других целей, например, информационных, служат простые веб-ресурсы, сайты. На таких ресурсах пользователи практически никак не участвуют в работе сайта, а лишь наблюдают, что этот ресурс показывает.

В веб-приложениях пользователь является активным участником процесса, он взаимодействует с интерфейсом с помощью компонентов и выполняет определенные действия, которые необходимы пользователю. В этом и состоит главная задача интерактивности – взаимодействие с пользователем.

В настоящее время многие информационные ресурсы стараются интегрировать в свои разработки элементы интерактивных приложений, потому что будущее разработки для веб-технологий заключается именно в интерактивности. Интерактивность для пользователя повышает уровень восприятия этого ресурса и, соответственно, качество и удобство использования всего продукта [10].

Далее будет сказано о важнейших составляющих современных интерактивных приложений, также о возможности интегрирования GUI в приложение, будет отмечена значимость проектирования интерактивных приложений с точки зрения производительности ресурсов.

### 1.2.1 Основные программные компоненты

Веб-приложения в основном представляют собой клиент-серверные приложения, которые работают по сети Internet. В качестве клиента обычно выступает браузер, установленный на персональном локальном компьютере или другом устройстве пользователя.

**Браузер** (или интернет - браузер) – это программа, с помощью которой можно просматривать страницы сайтов в интернете, переходить между страницами и самими сайтами, искать информацию, скачивать файлы, просматривать видео, слушать музыку, общаться на форумах и социальных сетях и т.д.

Сейчас, с развитием интернет технологий, браузер – самая популярная программа. Для некоторых людей в браузере буквально проходит вся их жизнь. Браузер настолько развивается, что уже сейчас контролирует и отслеживает разные события во многих сферах человеческой деятельности.

На данный момент, различных браузеров достаточно много, но большинство пользователей отдают предпочтения двум – трем проверенным разработкам. Очевидно, это связано с тем, что браузер имеет доступ ко всем данным пользователя, а значит является хранилищем. И, конечно же, человек не привык раскрывать свои данные для большого количества сервисов.

Также, стоит отметить, что некоторые браузеры помимо прямого назначения занимаются сторонней деятельностью в области информационных технологий. Например, являются поисковиками или создают свои приложения, как «Яндекс.Карты» и «Яндекс.Метро».

К самым популярным можно отнести Google Chrome, Mozilla Firefox и Opera [11]. Один из них изображен на рисунке 1.5.



Рис. 1.5. Современный вид браузера

А в качестве серверной составляющей веб-приложения выступает сам сервер. **Веб-сервер** представляет собой программу, которая принимает и обрабатывает входящие HTTP запросы, а затем генерирует и отправляет HTTP ответы клиенту, которым в большинстве случаев является браузер.

Существует множество веб-серверов, на данный момент два самых распространённых сервера с открытым кодом в мире: Apache и Nginx. В совокупности на них приходится более 50% всего трафика в мире. Оба этих сервера предполагают работу под разными нагрузками, а также взаимодействие как с пользователем, так и с другими серверными приложениями.

**Apache HTTP Server** был разработан Робертом Маккулом в 1995 году. С 1999 года разрабатывается под управлением фонда развития Apache Software Foundation. В конце 20 века этот сервер был очень популярен, пока не появился его конкурент Nginx. Apache привлекает своей гибкостью и мощностью. Также на сервере возможно запускать программы на многих интерпретируемых языках.

**Nginx** сервер начал разрабатываться еще в 2002 году, а первый релиз вышел в 2004 году. Этот веб-сервер отличается своей легковесностью и возможностями масштабироваться на минимальном техническом обслуживании. Сервер отлично выдерживает высокие нагрузки и был спроектирован для эффективной отдачи статических данных, поэтому многие хостинговые компании используют именно Nginx в качестве сервера.

Иногда Apache и Nginx используют вместе. Популярной схемой использования является размещение Nginx перед Apache. По своей сути, в такой схеме Nginx будет представлять frontend, а Apache – backend. Эта конфигурация дает преимущество в скорости обработки запросов, а также к выполнению этих запросов под высокими нагрузками, то есть повышается отказоустойчивость приложения.

Таким образом, сервер является очень важной составляющей веб-приложения, который представляет собой очень мощное, гибкое и универсальное устройство, которое выполняет многие требования пользователей [12].

Важной составляющей интерактивного приложения также является **база данных**. Самые простые приложения могут обойтись без использования базы данных, но большинство современных приложений активно взаимодействуют с этим инструментом, потому что есть потребность обрабатывать и хранить информацию больших объемов.

Под базой данных понимают совместно используемый набор логически связанных данных, используемых прикладными программами и предназначенный для удовлетворения пользовательских информационных

потребностей.

Важной отличительной характеристикой базы данных является то, что она хранится и обрабатывается в электронной вычислительной системе. Эта система способна осуществлять анализ и обработку хранящейся информации, а также выполнять запросы к данным. Обычно такие системы называются системами управления базами данных (СУБД).

Сейчас практически все веб-приложения и сайты, которые являются динамическими и адаптивными, используют СУБД для работы с пользовательскими данными, так как это повышает универсальность к быстроразвивающимся технологиям, а также позволяет использовать ресурсы веб-сервера рационально.

Существует несколько видов систем управления базами данных:

- Иерархические
- Сетевые
- Реляционные
- Объектно-ориентированные
- Объектно-реляционные

Каждый вид СУБД работает со своим набором данных, которые имеют определенный тип и структуру. Соответственно различаются и лингвистические инструменты, которые используют системы управления базами данных.

Современные СУБД содержат несколько компонентов: ядро, отвечающее за управление данными, процессор языка базы данных, оптимизирующий запросы, подсистему поддержки времени исполнения, интерпретирующие программы манипуляции данными, и другие сервисы.

Все перечисленные составляющие веб-приложения играют важную роль в работе всей системы. Но важнейшее значение является то, как эти компоненты между собой взаимодействуют. На рис. 1.6 представлено графическое отображение связи этих компонентов.

В целом, это обычное явление для веб-разработок, но в некоторых случаях, такая схема может преобразовываться: могут использоваться дополнительные компоненты, а также вполне вероятно и отсутствовать.

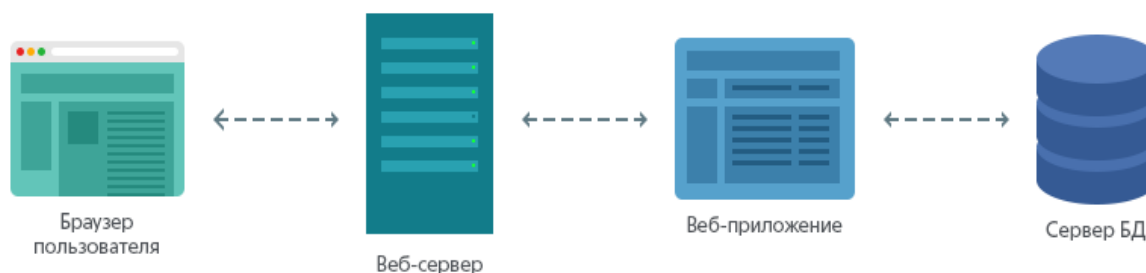


Рис. 1.6. Связь компонентов для работы веб приложения

Из этой иллюстрации очевидно, что пользователь в конечном итоге получает отображение в браузере. И, на самом деле, пользователя не интересует как происходит работа веб-приложения, какие запросы выполняются и с какой скоростью [13].

Позже будет подробно разобрана связь веб-приложения с графическим интерфейсом, а пока выделим достоинства и недостатки современных веб-приложений.

### 1.2.2 Преимущества и недостатки

С развитием технологий в области IT также развивались и веб-технологии, что привело к повышению качества современных веб-приложений. Выделим основные преимущества нынешних приложений:

- Отсутствие необходимости устанавливать приложение на компьютер или другое локальное устройство. Одно из самых значимых преимуществ, потому что отсутствие установки упрощает пользование продуктом для людей, не очень продвинутых в новых технологиях;
- Своевременное и автоматическое обновление приложения для каждого пользователя, что является одним из важных преимуществ двадцать первого века в мировой веб-среде;
- Реализация кроссплатформенности и кроссбраузерности, что означает, что приложение можно запускать на любом из существующих браузеров и под любой операционной системой, а также на любом устройстве, будь то компьютер или мобильное устройство;

- Высокая скорость обмена данными между пользователем и сервером, что отражает большой уровень интерактивности приложения, а также говорит об правильном и рациональном использовании вычислительных и обрабатываемых ресурсов сервера;
- Возможность приложения работать асинхронно, что подразумевает, что несколько пользователей в один и тот же момент времени могут получить доступ к одним и тем же данным сервера. Ранее до развития асинхронности, пользователям приходилось ожидать ответа сервера практически как в очереди;

Выделив основные плюсы и преимущества современных веб-приложений, стоит отметить, что хоть развитие происходит с очень высокой скоростью, все же есть недостатки в нынешнем функционировании приложений. Об этих недостатках и пойдет речь далее:

- Необходимость подключения к сети Internet, что приводит к тому, что качество работы приложения на стороне пользователя зависит от скорости соединения с Internet;
- Скорость обработки и передачи каких-либо пакетов зависит от их размера. Соответственно, чем больше размер требуемой передачи данных, тем дольше пользователь будет находиться в ожидании;
- Высокая вероятность взлома приложения по сети Internet. Над этой проблемой работают каждый год все более усерднее и достигают некоторого прогресса, тем не менее число таких нарушений остается примерно на таком же уровне;
- Необходимость подключения высоко требовательных сценариев в браузере, что приводит к нагрузкам на оперативную память локальной машины пользователя;

Важно сказать, что недостатки имеются, но над их решением трудится не одна компания и не один человек. Эти недостатки появились лишь с развитием технологий и приведут к появлению новых усовершенствованных продуктов, которые, несомненно, сделают нашу жизнь лучше. Ведь, как известно, лень это двигатель прогресса, а значит в стремлении улучшить свою жизнь и облегчить

ее деятельность, человек обязательно разработает новое программное обеспечение, которое будет на порядок качественнее предыдущих разработок.

С таким мнением авторы Безъязыкова Н. А. и Яковлева М. С преподносят свою статью [14].

### **1.2.3 Связь с GUI**

Как уже отмечалось ранее, веб-индустрия является одной из самых развивающихся областей информационных технологий. По этой причине разработка графического интерфейса в веб на сегодняшний день является очень востребованной.

При создании веб-приложения важно понимать для кого и для каких целей это приложение разрабатывается. В соответствии с целями и задачами также проектируется макет предполагаемого GUI и подбираются инструменты.

Так как приложение отображается в браузере, то, очевидно, что и весь интерфейс приложения разрабатывается под всевозможные виды браузеров. Также, стоит отметить, что с развитием мобильных устройств, в том числе планшетов и смартфонов, по сути, появился термин «адаптивная верстка». Этот термин напрямую связан с графическим интерфейсом и означает, что интерфейс сайта или приложения будет подстраиваться под размеры экрана пользовательского устройства, при этом сохраняя органичный и удобный для просмотра вид.

Современные веб-приложения являются очень популярными как раз, за счет того, что они разработали и вложили большие средства в графический интерфейс, который в свою очередь увеличил основные целевые показатели, будь это процент продаж или просто количество посетителей [15].

Связь современных приложений с пользовательским интерфейсом осуществляется с помощью нескольких инструментов. Если само веб-приложение можно реализовать на серверных языках программирования, таких как PHP, Python, и др., то для реализации красивого и удобного для пользователя интерфейса необходимо воспользоваться следующими основными инструментами:

- Основной язык для разработки интерфейсов – HTML5, отвечает за разметку и структуру документа или приложения;

- Язык стилей CSS3, с помощью которого настраиваются цветовые схемы приложения, а также некоторые виды анимации;
- Язык соединяющий серверную логику с дизайнерскими разработками – JavaScript. Богатый на возможности язык, который очень часто используется в разработке веб-приложений;

Здесь перечислены только самые основные инструменты [16]. На сегодняшний день инструментов для разработки пользовательских интерфейсов веб-приложения очень большое количество.

Все это свидетельствует о высоком уровне развития этой области, которая и в дальнейшем будет развиваться, создавая новые продукты и сотрудничая с разными другими отраслями производства и науки.

#### **1.2.4 Производительность**

При разработке веб-приложения очень большое внимание уделяется такому показателю как производительность. Под производительностью будем понимать скорость работы приложения, а если точнее, то скорость обработки запросов, скорость отправки данных и скорость выполнения других действий, необходимых для функционирования приложения.

Производительность веб-приложения складывается из, как минимум, двух составляющих:

- Работа веб-сервера (скорость, мощность, реализация многопоточности и асинхронности);
- Работа самого веб-приложения (программный код, структура основных модулей);

Анализ и измерение производительности осуществляют, как правило, в рамках оптимизации. Оптимизированные приложения имеют некоторые преимущества перед другими, за счет того, что пользователю всегда приятно, когда приложение работает быстро и без зависаний. И, естественно, это приводит к повышению качества продукта.

В этом обзоре мы затронем тему производительности приложения с точки зрения программного кода и основных модулей, так как это связано непосредственно с основной темой данной дипломной работы.



Как уже было сказано, производительность приложения зависит от программного кода. Например, приложение, работающее корректно, выполняет запросы к базе данных, но эти запросы выполняются очень медленно. Это может быть связано с самой структурой запроса, и, зачастую, запросы к базе данных стараются писать, как можно проще, чтобы не создавать лишнюю нагрузку на сервер.

Также, очень важно сказать, что на данный момент имеет место тенденция, при которой разработчики стараются применять как можно меньше запросов к базе данных, чтобы разгрузить сервер. Разработчик добивается это благодаря замене некоторых запросов, результат которых очевиден, на простые значения – литералы. Это один из вариантов оптимизации веб-приложения.

Еще один вариант повышения производительности, это создание отдельных функций или методов, которые встречаются в программе очень часто. На самом деле, этот прием дают еще в начале обучения программированию, но, как ни странно, не все программисты следуют этим советам. Такой подход нужно применять еще на этапе проектирования приложения и описания тестовых вариантов будущих модулей.

Если же по какой-то причине такой подход не был реализован, то самое первое, что делают для оптимизации и повышения производительности, это как раз выделяют код в отдельные модули и функции. Также, если мы говорим о веб-приложениях, то подключение таких модулей стараются делать асинхронным, чтобы иметь выигрыш по времени перед конкурентами.

Важно отметить, что создание отдельных модулей и виджетов также существенно может повысить производительность веб-приложения. И самое главное для разработчика – спроектировать структуру таким образом, чтобы эти модули работали правильно и образовывали единую систему. Другими словами, программист должен обеспечить целостность своей разработки, иначе в последствии возможно появятся трудноуловимые баги.

Также отличным решением будет использование генерации контента с помощью все тех же модулей. Генерация содержимого создает предпосылки к созданию высокопроизводительного веб-приложения, которое будет обладать высокой скоростью исполнения.

Итак, мы рассмотрели основные методы повышения производительности приложения, которые желательно продумывать еще на этапе проектирования, о

чем и говорят многие специализированные источники, в том числе и сайт об интернет приложениях [17].

Подводя итоги литературного обзора, следует отметить что данная тема в общем смысле раскрыта достаточно хорошо, и источники, которые были выбраны, содержали актуальную информацию.

Если переходить к более конкретным вещам, то важными задачами при разработке веб-ориентированного графического пользовательского интерфейса является разработка оптимизированного адаптивного продукта, который представляется как целостная система и является универсальным и удобным средством использования для множества различных целей. Именно об этих важнейших характеристиках сообщало большинство проанализированных источников.

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

### 2.1 Выбор языка серверной разработки

Говоря о веб-клиенте, на котором и для которого производится разработка, и рассматривая его с точки зрения системного анализа объекта, важно рассмотреть какой язык и какие инструменты используются для реализации веб-клиента системы.

Самым важным является веб-сервер, об основных понятиях которого было упомянуто ранее в 1 главе. В этой же главе будет произведен более конкретный разбор языка программирования и среды разработки.

В качестве языка программирования серверной логики был выбран высокоуровневый язык **Python**, на момент написания – самая последняя версия – Python 3, о чем всегда можно узнать на официальном сайте [18].

Личный выбор автора аргументируется тем, что веб-сервер уже работал через этот язык программирования, и менять язык было бы очень нерационально и неэффективно. К тому же, этот язык программирования общего назначения, ориентированный на высокую производительность и скорость работы, достаточно прост в понимании и реализации.

Python поддерживает несколько разновидностей программирования, такие как объектно-ориентированная, функциональная и структурная. Код в этом языке организуется таким образом, что он легко может быть использован в качестве модуля. А несколько модулей, в свою очередь, могут образовывать пакеты. О том какие модули используются в веб-клиенте будет описано далее.

Стоит отметить, что за долгое время существования языка Python, было разработано большое количество фреймворков для этого инструмента программирования. Фреймворком является некая разработка, которую могут использовать другие программисты по своему усмотрению, добавляя свои функции и наработки.

Обычно фреймворки используют, так как они повышают производительность и заметно упрощают код программных модулей.

Безусловно, вариант использования фреймворка несколько ограничивает программиста, но, в целом, преимуществ использования этой разработки гораздо больше, чем недостатков.

К основным и популярным фреймворкам относятся следующие:

- **Django**, пожалуй самый известный фреймворк, который представлен стандартной простой структурой, которую способен понять даже начинающий программист. С помощью этого инструмента функционирует используемый в этой дипломной работе веб-клиент;
- К другим популярным разработкам относят: Flask, TurboGears, Tornado, Web2Spy;

Так как в представленном клиенте используется первый из перечисленных выше фреймворков, то о нем и подробнее пойдет речь.

Django – свободный программный каркас для веб-приложений на языке Python, использующий шаблон проектирования «Модель-Представление-Контроллер» (MVC). Позволяет существенно упростить процесс разработки серверной части веб-приложения [19].

Для взаимодействия с сервером приложений будет использоваться специальный пакет communication, написанный на языке Python, реализующий протокол сетевого взаимодействия между веб-приложением и сервером приложений.

Стоит отметить, что Python применяется как базовый язык веб-сервера, так и язык для локального сервера. Под локальным сервером понимают программу, установленную на персональный компьютер, которая способна работать как веб-сервер в рамках одного компьютера. Она предоставляет все возможности сервера и является полезной для разработки тестовых приложений, так как есть возможность отслеживать и мгновенно исправлять ошибки при разработке.

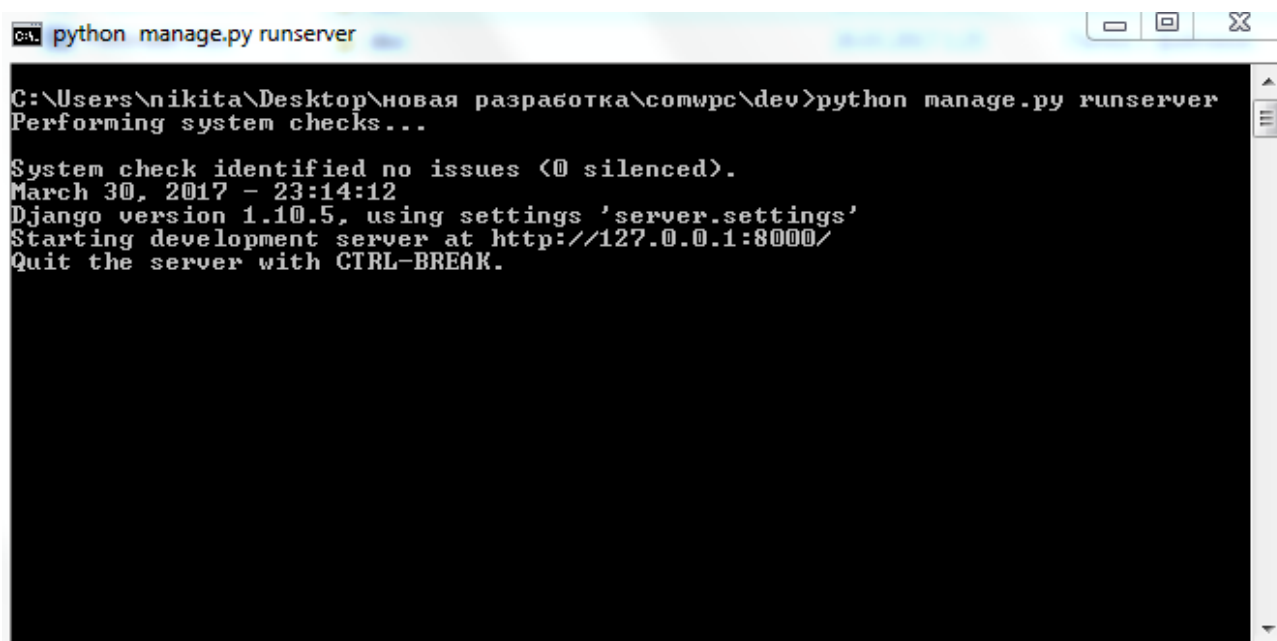
Часто программисты разрабатывают и тестируют свои программы именно на локальном сервере. Следующая команда позволяет запустить локальный сервер на Python:

`Python manage.py runserver`

Очевидно, что для правильного выполнения этой команды и для запуска локального сервера требуется скачать и установить Python. После установки необходимо зайти в ту папку, где лежит файл управления запуском manage.py, обычно этот файл отвечает за запуск.

Затем в этой директории необходимо вызвать командную строку или терминал, в зависимости от операционной системы, и прописать команду

запуска, пример которой уже приведен.



```
python manage.py runserver
C:\Users\nikita\Desktop\новая разработка\comwpc\dev>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
March 30, 2017 - 23:14:12
Django version 1.10.5, using settings 'server.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Рис. 2.1. Пример запуска локального сервера Python

После успешного запуска, который изображен на рис. 2.1, можно работать с данными, хранящимися на сервере, которые обычно копируются на локальный персональный компьютер.

Дальнейшая разработка ведется с помощью других инструментов и в других средах, о которых речь пойдет дальше. И в заключении отметим, что после изменений в локальном сервере разработчику следует поместить новые или измененные файлы проекта в директорию рабочего веб-сервера. Это можно сделать разными способами, как вариант использовать репозиторий, работающий через систему контроля версий и привязанный к серверу, что и используется в данной системе и позволяет оперативно использовать это в квалификационной работе.

## 2.2 Выбор средств визуальной разработки

В качестве инструментов программирования для визуализации и построения пользовательского графического интерфейса был выбран стандартный набор для веб-программирования, в который входит HTML5, CSS3, JavaScript, а также специальные и популярные инструменты: библиотека jQuery

и фреймворк Twitter Bootstrap3.

Выбор в пользу этих инструментов был сделан по причине того, что они уже были задействованы в разработке веб-клиента, а также потому, что эти инструменты очень просты в понимании и пригодны для реализации поставленных задач. А также по причине того, что эти инструменты с каждым годом развиваются и являются флагманами своих областей.

**HTML5** – язык разметки для структурирования и представления информации во всемирной паутине. На этом языке и держится весь интернет, все сайты и веб-приложения. Язык очень прост, но в то же время позволяет создавать различные по сложности и возможностям приложения и веб-страницы.

**CSS3** – расшифровывается как каскадные таблицы стилей. Язык, который дополняет HTML стилизацией и улучшением внешнего вида всех элементов сайта или приложения. Начиная с последней версии появилась возможность создавать анимации, а также различные сочетания эффектов.

**JavaScript** – объектно-ориентированный язык сценариев. Большое применение нашел в качестве языка, исполняемого в браузерах для реализации динамического представления и исполнения событий при взаимодействии пользователя с веб-приложением [20].

Удобство описанных инструментов заключается в том, что для их использования практически ничего не требуется. Не нужно устанавливать какое-либо дополнительное программное обеспечение. Написание кодов возможно в любом текстовом редакторе в файлах с расширениями .html, .css и .js соответственно.

Совокупность исполняемых файлов образует проект, который можно назвать веб-приложением. Связь файлов осуществляется в файле html с помощью следующих команд:

- Подключение файлов стилей:  
`<link rel="stylesheet" href="file.css">`
- Подключение файлов скриптов:  
`<script src="file.js"></script>`

Подключение файлов происходит с помощью специальных команд и указателей языка html – тегов. Стили подключаются в теге link, в котором

происходит указание, что подключаются стили с помощью атрибута `rel`, а затем в атрибуте `href` прописывается путь до файла `css`. А файлы скриптов подключаются в теге `script` с указанием пути до файла `js`.

Стоит отметить, что для быстрого действия программы, файлы стилей подключают в начале файла `html`, а файлы скриптов – в конце. Далее приведем общую структуру `html` разметки (рис. 2.2), которая используется на любом сайте или приложении.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Заголовок</title>
6      <link rel="stylesheet" href="file.css">
7  </head>
8  <body>
9      <main>
10         ОСНОВНОЙ КОНТЕНТ
11     </main>
12     <script src="file.js"></script>
13 </body>
14 </html>
```

Рис. 2.2. Структура `html` файла

Такая структура изменялась и совершенствовалась с годами, и на данный момент на рисунке выше представлен шаблон, соответствующий всем требуемым стандартам. Стандартная разметка включает несколько обязательных тегов: `html`, `head`, `body`. Они являются закрывающимися, то есть требуют открытия и закрытия и определяют весь файл `.html`, который отображается пользователю.

Помимо них существуют огромное количество тегов, но они не будут являться основополагающими, так как правильно отобразить страницу возможно и без их использования.

В общем смысле рассмотрены основные инструменты для разработки и решения поставленных задач данной выпускной работы.

Теперь подробнее разберем специальные инструменты, которые были перечислены в начале параграфа.

**JQuery** - библиотека JavaScript, которая позволяет существенно оптимизировать код, а также с легкостью обращаться к объектным элементам

документа [21]. Эту библиотеку используют многие фреймворки. В данном веб-клиенте она использована, чтобы в будущем можно было использовать эти фреймворки и другие полезные библиотеки.

Подключение jQuery происходит таким же образом, как и обыкновенный js файл. Для правильного исполнения скриптовой части, файл jQuery следует подключать ранее чем файл js.

Сам код на jQuery следует начинать со следующей конструкции, которая является одной из основополагающих:

```
$( document ).ready(function() {
```

Приведенная строчка означает, что прежде чем выполнять скриптовую часть, документ должен полностью загрузиться. Это очень важно, так как позволяет избавиться от проблем с элементами документа. Также повышается скорость загрузки сайта в браузер пользователя.

**Twitter Bootstrap** - фреймворк, который состоит из набора шаблонов, реализованных на html и css. Самый популярный фреймворк, на котором работает множество различных сайтов и веб-приложений. Требуется подключение jQuery для работы с этим инструментом.

Прежде чем подключить к нашему веб-приложению Twitter Bootstrap, необходимо скачать файлы исходников с официального сайта [22], а затем подключить файлы стилей. Как это делается, демонстрировалось ранее.

Основное преимущество Bootstrap заключается в том, что он позволяет создавать адаптивные приложения, которые могут динамически изменять свои свойства и содержимое в зависимости от устройства, которое используется. Именно это важно при создании динамического пользовательского графического интерфейса.

Bootstrap достаточно прост в освоении и в понимании основных команд. Главное, что нужно понять - вся страница делится на 12 колонок. К каждому блочному элементу применяется класс, который определяет сколько колонок этот блок будет занимать. Отметим, что можно менять ширину в зависимости от устройства. Все это выполняется следующим образом:

```
<div class="row">
```

```
  <div class="col-md-1">1/12 часть экрана</div>
```

```
  <div class="col-md-6">1/2 часть экрана </div>
```



</div>

Сначала задается тег `row`, который определяет ряд колонок. В этом теге создаются блочные элементы, которые имеют класс `col`. Классы `col-md-1` и `col-md-6` означают какое количество единичных блоков содержит данный блок соответственно. Как видно из примера - создано два блока с разными по значению ширинами.

А в общем случае, документ можно представить таким образом, как это сделано на рис 2.3.

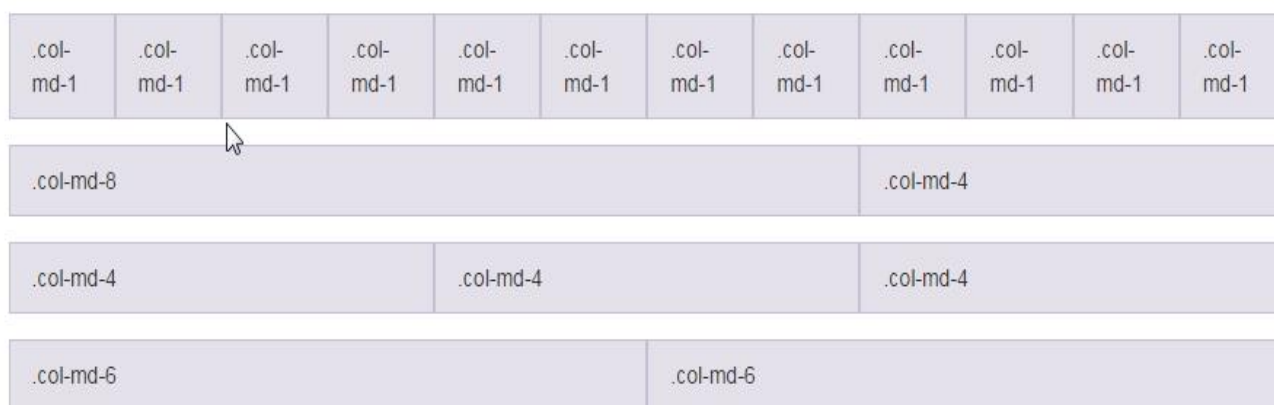


Рис. 2.3. Структура документа в Bootstrap

Помимо общих и специальных инструментов, используются библиотеки, о которых следует сказать несколько слов:

- **FontAwesome** - библиотека шрифтовых иконок, которые на сегодняшний день, очень популярны. В данной работе этот инструмент тоже пригодился для визуализации и правильного отделения блоков интерфейса;
- **Vis.js** - библиотека для построения и визуализации графиков, в частности, ориентированных графов, построение которых входит в одну из основных задач этой дипломной работы. Помимо графов, библиотека работает с двухмерными и трехмерными графиками и диаграммами, что добавляет преимущество к использованию этой библиотеки;

Таким образом, в этом параграфе рассмотрены практически все инструменты, которые используются для визуализации и построения графического интерфейса. Эти инструменты напрямую связаны с объектом исследования и представляются автору данной работы наиболее подходящими для выполнения поставленных задач.

## 2.3 Использование GIT

Как уже было сказано, разработка может осуществляться на локальном персональном компьютере, на котором запущен веб-сервер. А для связи с удаленным сервером использовать систему контроля версиями **GIT** [23].

Это очень полезный и современный инструмент, который позволяет работать удаленно над любыми проектами, как одиночными, так и командными. Отличным примером работы нескольких программистов будет случай, когда один разработчик занимается серверной логикой приложения, а другой в то же время работает над визуальной частью. Затем, с помощью команд GIT, они соединяют свои разработки в один проект.

Для операционной системы Windows используется программа GitBash, которая представляет собой консольное приложение (рис. 2.4), работающее с удаленными репозиториями по нескольким протоколам, такими как HTTP, SSH и другими.

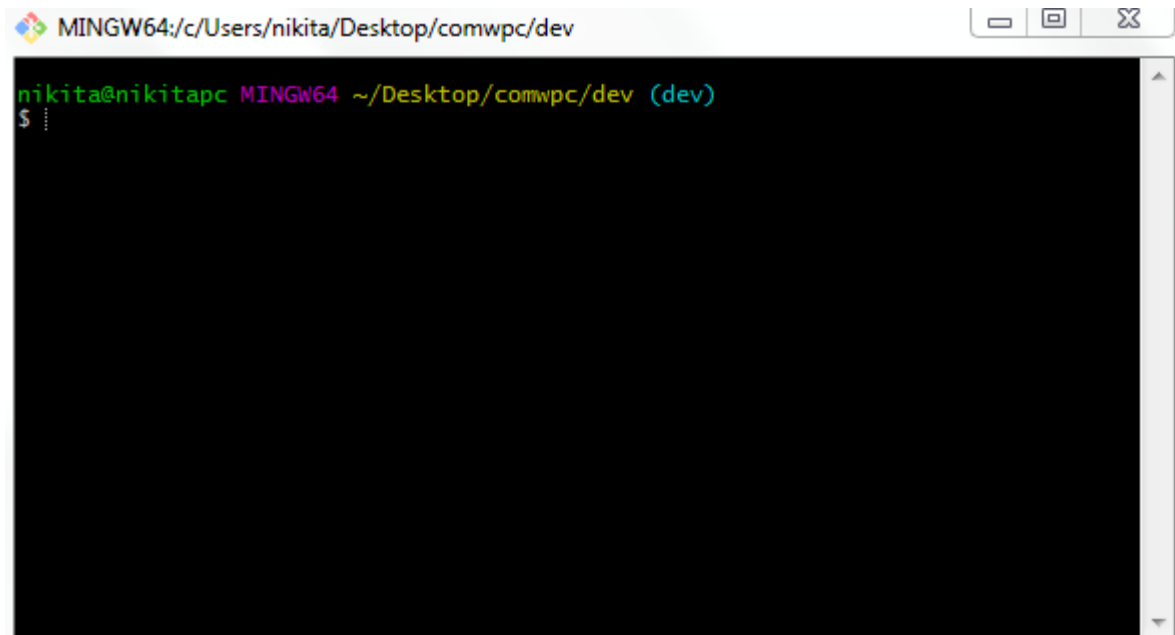


Рис. 2.4. Стандартная консоль GitBash

Такая консоль помимо того, что выполняет команды для GIT, еще и имеет настраиваемый набор команд, как в операционной системе Linux. Иногда такой консолью удобно пользоваться для решения простых задач навигации и рутинных работ в своей системе.

### 2.3.1 Создание репозитория

Прежде чем приступить к работе с GIT, необходимо создать локальный репозиторий. Это можно сделать несколькими способами. Первый способ заключается в создании папки под репозиторий, а затем выполнение следующей команды:

```
$ git init
```

Этой командой папка становится репозиторием, с которым можно выполнять дальнейшие операции.

Второй же способ - это клонирование уже созданного репозитория. Поскольку система GCD, в которой ведется разработка, существует уже давно, то и репозиторий был клонирован следующей командой:

```
$ git clone
```

Команды в GIT похожи на команды в Linux Bash, что уже отмечалось и соответственно упрощает их понимание, если имеется опыт работы в терминале.

### 2.3.2 Добавление файлов

В процессе работы программисту необходимо сохранить свои файлы – в GIT это называется сделать слепок. Сначала необходимо сделать файлы отслеживаемыми, с помощью команды:

```
$ git add file
```

Это делается для того, чтобы по случайности программист не добавил в репозиторий файлы, которые не следует добавлять по причинам безопасности. Файлы в большинстве случаев добавляются вручную, а значит программист контролирует этот процесс. Затем нужно выполнить коммит, по сути сохранить текущее состояние файлов:

```
$ git commit -m "Пример коммита"
```

Данную команду следует выполнять с флагом `-m`, который указывает на то, что после него будет указан комментарий к коммиту. Затем указывается сам комментарий. Он нужен для того, чтобы в будущем разработчик мог вернуться к любому этапу работы, прочитав комментарий, и если возникла какая-нибудь ошибка, то программист хорошо мог ориентироваться в версиях для устранения этих неполадок.

И после этого можно добавить файлы в удаленный репозиторий, с помощью следующей команды:

```
$ git push
```

При выполнении этой команды предполагается, что есть соединение с удаленным репозиторием по одному из протоколов, а также в настройках указана пара логина и пароля для получения доступа к самому репозиторию.

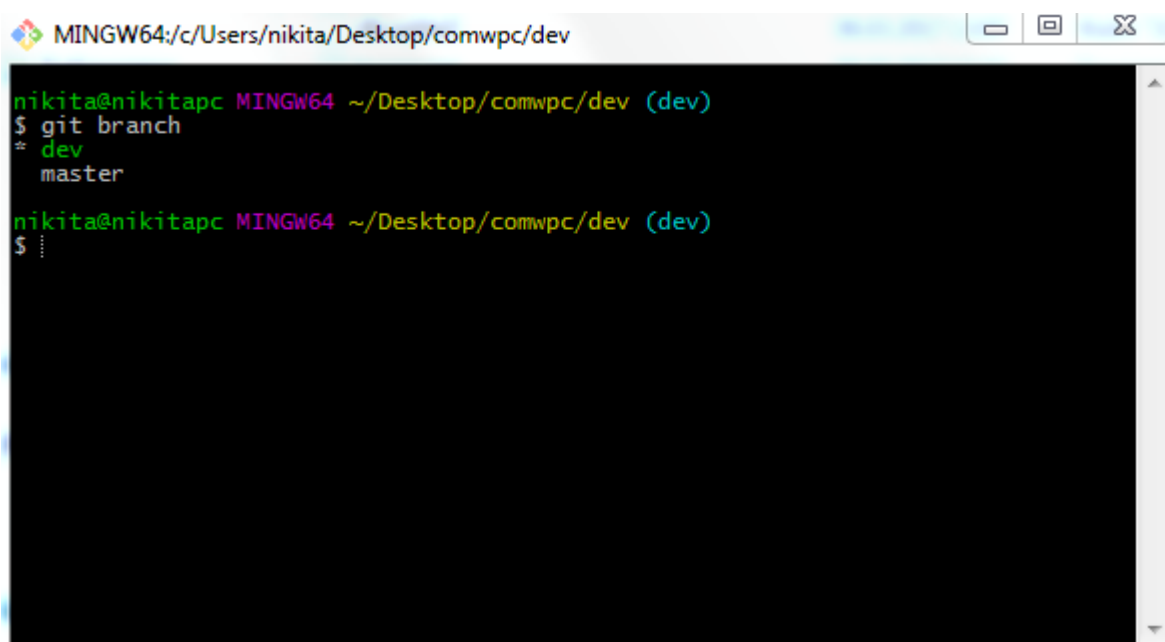
### 2.3.3 Ветки

Ветки в GIT имеют очень важное значение. Когда над одним большим проектом трудятся несколько человек, то важно разделять разработку этих программистов. С помощью ответвлений разделяются разные версии одного и того же проекта. Это нужно, например, когда одна ветка тестовая, а другая - рабочая. И, конечно, это только повышает качество разработки и позволяет без проблем осуществлять изменения в проекте.

Также использование веток очень пригодится для данной работы, так как от основной ветки разработки всего проекта следует отделить ветку разработки определенного модуля.

Чтобы посмотреть ветки в репозитории, нужно воспользоваться командой:

```
$ git branch
```

A screenshot of a terminal window titled 'MINGW64:/c/Users/nikita/Desktop/comwpc/dev'. The prompt is 'nikita@nikitapc MINGW64 ~/Desktop/comwpc/dev (dev)'. The command '\$ git branch' has been entered, and the output shows two branches: 'dev' and 'master'. The 'dev' branch is highlighted in green, indicating it is the current branch. The prompt then changes to 'nikita@nikitapc MINGW64 ~/Desktop/comwpc/dev (dev)' and the command '\$' is entered on the next line.

```
nikita@nikitapc MINGW64 ~/Desktop/comwpc/dev (dev)
$ git branch
* dev
  master

nikita@nikitapc MINGW64 ~/Desktop/comwpc/dev (dev)
$
```

Рис. 2.5. Пример команды git branch

Эта команда отображает список веток (рисунок 2.5), а также выделяют ту ветку, на которой в данный момент работает программист. Для того, чтобы переключиться между ветками нужно воспользоваться другой командой:

`$ git checkout otherbranch`

Синтаксис команды подразумевает, что нужно указать имя ветки, на которую пользователь хочет перейти. Стоит отметить, что при переходе на другую ветку, может поменяться версия некоторых файлов, то есть произойдут изменения содержимого, что вполне логично.

В этом параграфе представлена теоретическая часть основных команд GIT, которые непосредственно связаны с разработкой графического интерфейса и имеют прямое влияние на весь предмет дипломной работы в целом.

## 2.4 Текстовый редактор Brackets

Программный код реализуется в текстовом редакторе. Поскольку веб-клиент работает с файлами html, css, js, python, то выбран текстовый редактор **Brackets**, который отличается очень удобным и приветливым интерфейсом, а также поддержкой различных расширений.

Brackets создан компанией Adobe Systems под лицензией MIT License и постоянно развивается [24]: создаются новые плагины и модули для работы в этом текстовом редакторе. Пример интерфейса можно наблюдать на рис. 2.6.



Рис. 2.6. Основная панель текстового редактора

Этот редактор был создан специально для frontend разработки, существует большой набор плагинов для взаимодействия пользователя с графическим

интерфейсом веб-приложений через интерфейс редактора, и поэтому, это отличный выбор для реализации поставленных задач.

## **2.5 Веб-клиент системы**

Веб-клиент идейно построен таким образом, что представлен аналогично тонкому клиенту, который располагается на рабочем столе персонально компьютера. Вся "толстая" функциональность вынесена и выполняется на сервере приложений. Вся "тонкая" функциональность представлена в виде `action_items` (по сути, отдельных блоков веб-интерфейса, о которых будет рассказано ниже) и системы уведомлений.

### **2.5.1 Архитектура**

Архитектуру веб-клиента можно показать на нескольких уровнях взаимодействия. На первом уровне (внешнем) взаимодействует, как уже было сказано, frontend и backend части. На рис. 2.7 представлена внешний уровень взаимодействия.

По сути весь веб-клиент существует на взаимодействии frontend и backend. Часть frontend представлена языком разметки HTML5, стилевым оформлением на CSS3, шаблонами на Django-template, а также частично скриптовым языком JavaScript.

Backend представлен серверным языком Python и написанными на нем скриптами, также используется JavaScript. В качестве системы управления базами данных выбрана СУБД PostgreSQL.

Взаимодействие между двумя основами веб-клиента осуществляется посредством технологий, основанных на аjax-протоколах. Это означает, что данные приходящие из базы данных принимает backend, а отображает в удобном для пользователя виде – frontend.

Также стоит отметить, что к веб-клиенту подключена система контроля версий GIT, которая предоставляет удаленный репозиторий, что существенно облегчает введение удаленной разработки и является очень удобным для совместной работы в системе с несколькими разработчиками. Как она устроена, уже сказано в предыдущих параграфах.

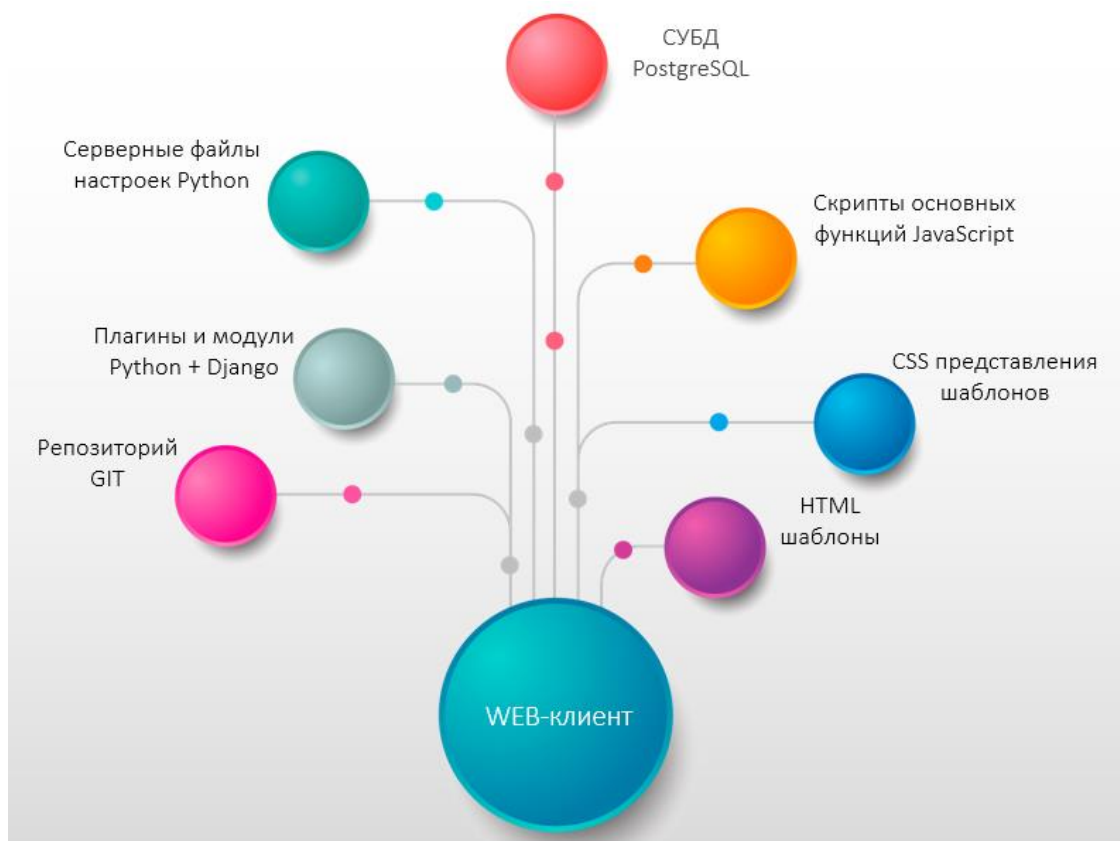


Рис. 2.7. Внешняя архитектура веб-клиента

Теперь перейдем к более глубокому описанию архитектуры веб-клиента. Внутри клиента осуществляется взаимодействие шаблонов html и плагинов python, которые именуются как `action_items`.

На данный момент реализовано чуть менее двадцати шаблонов, каждый из которых отвечает за свою задачу и направление. Что касается плагинов, то на данный момент количество их менее десяти, но тем не менее, плагины или `action_items` играют важнейшую роль в существовании и деятельности всего клиента.

Поскольку используется аякс-протоколы, то необходимости в большом количестве страниц нет. По этой причине на клиенте реализовано две страницы - страница авторизации и главная страница, которая содержит в себе:

- функциональное меню, которое является самым главным элементом, с помощью которого и выполняются все основные действия на клиенте;
- Область отображения, в которую загружается с помощью аякс-запросов данные в различных видах и типах представлений;

- Меню уведомлений, который на данный момент разрабатывается;

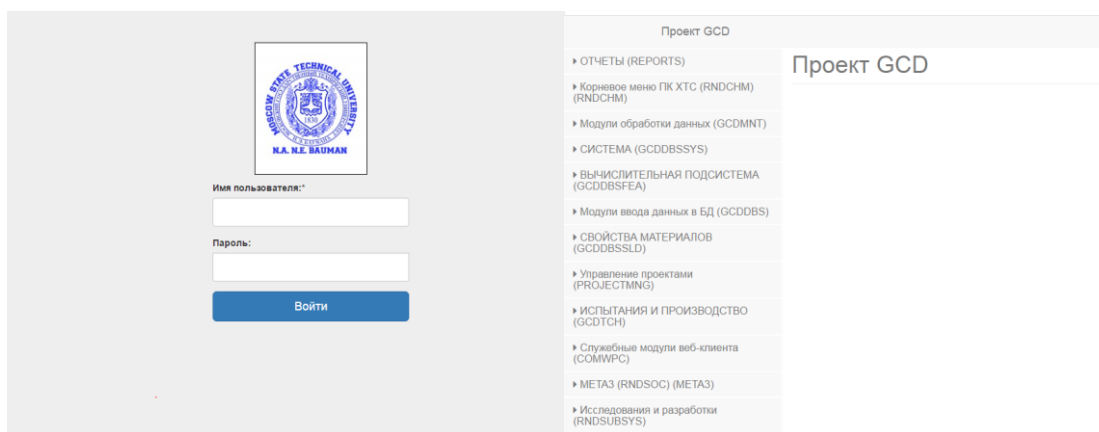


Рис. 2.8. Страница авторизации (слева), главная страница с функциональным меню (справа)

На рис. 2.8 можно увидеть, как выглядят эти страницы, чтобы иметь визуальное представление о веб-клиенте системы и лучше понимать суть дальнейшей разработки.

И в заключении этого пункта предоставим изображение внутренней структуры веб-клиента на рис 2.9, о которой только что шла речь. И можно сделать вывод о важности языка JavaScript для всего клиента, так как он участвует и в визуальной части и в серверной.

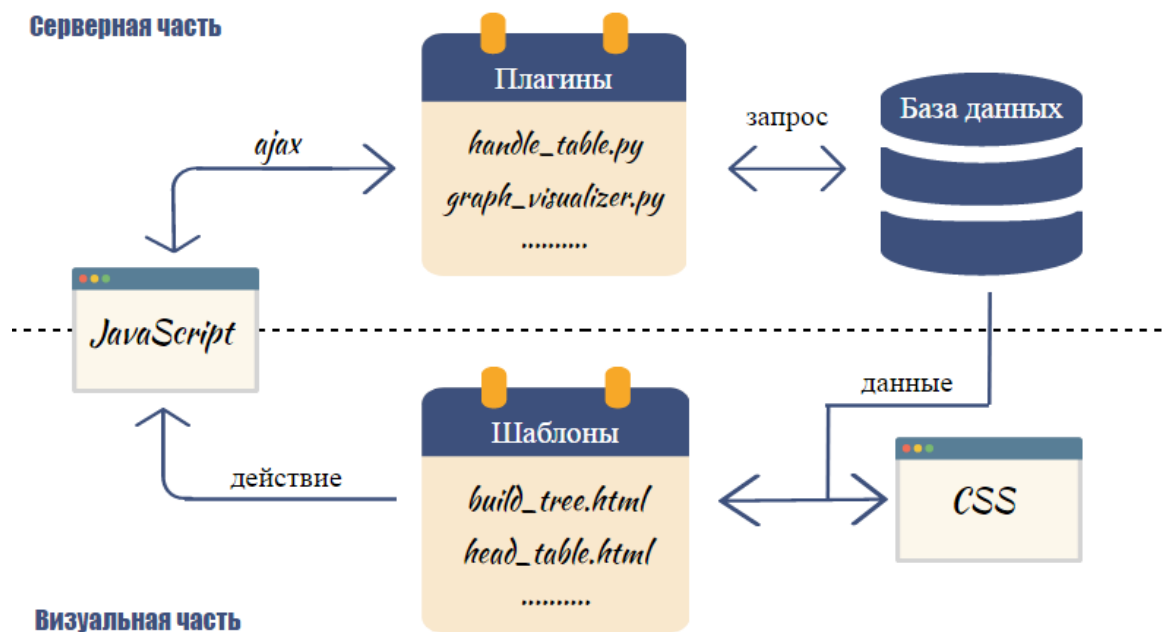


Рис. 2.9. Архитектура веб-клиента



## 2.5.2 Основные модули

После описания архитектуры перейдем к описанию основных модулей клиента – `action_items`. Как уже было сказано, это основные плагины, без которых невозможно представить работоспособность веб-клиента.

Итак, самые важные, основные и служебные `action_items`:

- `handle_table` – фиктивный плагин, который обеспечивает работу с другими плагинами, в частности класса `SPBLDB`. Необходим для обеспечения совместимости с уже существующими модулями в офлайн версии для персонального компьютера. В этом плагине строятся все типы представления данных, поэтому очевидно, что это самый используемый плагин;
- `wpc_gui_ini_builder` – построитель пользовательских графических (html-шаблонов) на основе `ini`-файлов;
- `proc_add_user_wpc` – плагин для контроля пользователей клиента, осуществляет добавление новых разработчиков в систему;
- `graph_visualizer` – осуществляет построение графов – решателей для технологии GBSE, о которой речь пойдет далее, важный плагин для построения GUI, связанного с ориентированными графами;

Важно отметить, что плагины работают в соответствии своей функциональности. Но все они используют `ajax`-запросы и `Json` формат для таких запросов.

Также очень интересным является функциональное меню системы. При нажатии на `action_item` меню, происходит `ajax`-запрос в `backend`, где формируется `div`-контейнер, которые вставляется в центральную зону. Потенциально, в `div`-контейнере может быть любой код, реализующий дальнейшее взаимодействие с `backend`, что дает возможность гибкого расширения веб-клиента.

На `backend` стороне `action_items` хранятся в виде отдельных `python`-модулей, в каждом из которых реализована функция `handle_ajax_request`, через которую и реализуется взаимодействие.

## 2.5.3 Типы представлений

Теперь подробнее рассмотрим типы представлений данных из БД, которые представляются в таблицах и других формах отображения пользовательского графического интерфейса.

Эта тема рассматривается по причине того, что две задачи для данного диплома напрямую связаны с разработкой представлений, и, соответственно, есть полная необходимость привести теоретическую часть для этой части.

На данный момент в веб-клиенте существует шесть типов представлений. Некоторые из них являются полностью уникальными, а некоторые лишь унаследовали часть свойств от других представлений. Тем не менее, эти типы представлений очень нужны и представляют большую значимость для системы.

Итак, перечислим типы представлений:

- **SIMPLE\_TAB\_VIEW** – самое распространённое представление, в котором находятся большинство функций системы, по крайней мере на сегодняшний день. Представляет собой вид обыкновенной и простой таблицы. Именно поэтому это представление самое популярное, изображение которой представлено на рис. 2.10;

### Настройки субплагинов на базе таблиц БД (sys.aidbs)

Имя таблицы: GCDDb.sys.aidbs  
Количество столбцов: 6; Количество строк: 227

#	aisid Функция РВС	schem Базовая схема данных (обязат.)	xdbtn Имя базовой таблицы (обязат.)	sqlrq SQL запрос (опц.)	dbtgu Метод представления GUI таблицы БД	dbunm Универсальный идентификатор БД
1	sys.dblog	sys	dblog			
2	RNDSOC.soc.comps	soc	comps		SIMPLE_TAB_VIEW	RNDSOC
3	sys.tlbas	sys	tlbas		GROUPED_TAB_VIEW	
4	pms.custs	pms	custs			
5	com.slprc	com	slprc		SIMPLE_TAB_VIEW	
6	tsk.mstsk	tsk	mstsk			
7	pat.ptsrc_grp_vi	pat	ptsrc_grp_vi		SIMPLE_TAB_VIEW	GCDDb

Рис. 2.10. Простое представление таблицы

- **GROUPED\_TAB\_VIEW** – сгруппированное табличное представление, которое является удобным при большом количестве записей;
- **FILTERED\_TAB\_VIEW** – отфильтрованное табличное представление, в котором, как понятно по названию, имеется возможность осуществлять

фильтрацию по атрибуту;

- **TREE\_TAB\_VIEW** – древовидное табличное представление, которое используют таблицы с ссылающимися внешними ключами на первичные ключи других строк этой же таблицы;
- **EXTERNAL\_TREE\_VIEW** – древовидное представление, построенное на базе реляционной модели двух таблиц: родительская и дочерняя с двумя внешними ключами на родительскую (предок-потомок);
- **SGROUPED\_TAB\_VIEW** – частично сгруппированное табличное представление;

В данной дипломной работе разработано отображение двух представлений: древовидного и сгруппированного, изображения и частичное описание которых, можно посмотреть в практической части, где реализованы основные модули программ.

#### **2.5.4 GBSE технология**

Поскольку в основные задачи этой дипломной работы включен пункт по изучению базовой технологии GBSE, то прежде всего, ее нужно было изучить, и, поэтому здесь будут приведены основные сведения.

**GBSE** расшифровывается как графо-ориентированная разработка программных реализаций сложных вычислительных методов (graph-based software engineering).

По своей сути это особый подход к реализации сложных вычислительных методов. Особенности этого подхода заключаются в следующем:

- понятия теории графов, без которой практически невозможно пользоваться и понимать технологию GBSE;
- технологии хранения информации в реляционных базах данных, что тоже играет очень важную роль;
- технологии применения библиотек динамической компоновки (DLL, so). применение специализированных структур данных (ассоциативных массивов, деревьев, графов);

Область применения подхода – программные реализации численных методов механики сплошной среды: методы конечных элементов, методы гомогенизации и другие.

Назначение базовой графо-ориентированной технологии:

- Системы инженерного анализа (CAE);
- Программное обеспечение, требующее определение понятия бизнес-процесс (ERP, CRM, веб-приложения с встроенной бизнес логикой);
- Оптимизационные системы;

Другие прикладные области также используют эту технологию в том или ином виде, но сейчас приведен список тех, которые представлены в системе GCD, а также в веб-клиенте.

Для понимания этого подхода необходимо ввести такое понятие, как сетевая модель. **Сетевой моделью** (GM) (графовой моделью) алгоритма будем называть ориентированный граф, при обходе которого реализуется алгоритм решения конкретной вычислительной задачи. Пример такой модели изображен на рис. 2.11.

Зеленым цветом выделен узел начала сетевой модели. Как можно увидеть, вариантов решения поставленной задачи несколько, что подчеркивает представленная модель.

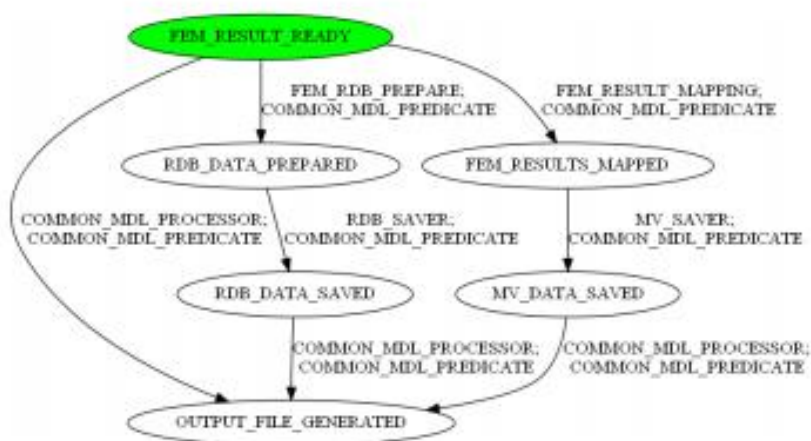


Рис. 2.11. Пример сетевой модели постпроцессора метода конечных элементов

В общем смысле, в данной задаче как раз необходимо реализовать модуль, который осуществляет построение таких представлений сетевых моделей для

всех решателей. Главное, чтобы этот модуль был достаточно универсален и строил понятные пользователю отображения с точки зрения графического интерфейса. А также важно создать такой модуль, который мог бы обрабатывать подграфы, то есть подзаключения решений, которые иногда включаются в ту или иную задачу.

Затем, важно отметить основные составляющие сетевой модели, так как с ними в дальнейшем, когда будет приводиться решение, производится различного рода действия. И поэтому важно ознакомиться с ними уже сейчас, чтобы понимать из чего состоит простая сетевая модель и как это можно использовать в будущем.

- **узел графа *Nodej*** – определяет состояние изменяющихся данных или вложенную графовую модель (подграф), если такая имеется. Обычно сложные модели включают в себя несколько маленьких моделей, которые образуют кластеры;
- **ребро графа *Edgei*** – индуцирует определение функции преобразования (перехода) одних и тех же данных из одного состояния в другое, по сути это линия, соединяющая 2 узла. Если связь имеет направление, то в этом отображении от первого узла идет стрелка ко второму узлу сетевой модели;
- **функции преобразования** – привязываются к ребрам, реализуются в виде пары экспортируемых из библиотек функций со стандартными сигнатурами (функции-обработчики и функции-предикаты могут использоваться многократно в рамках одной и той же сетевой модели);

Важно отметить, что такой подход позволяет разработчику ПО использовать одну из представленных очередей вызовов, а не только одну, которую бы он выбрал при классическом подходе. Другими словами, такой подход реализует вариативность при разработке ПО. Также существует возможность использование технологии при многопоточном выполнении алгоритма.

Технология GBSE существенно повышает производительность и качество разработки, что положительно влияет на сложные вычислительные программные модули любых систем, в том числе и систем инженерного анализа.

Очевидно, что реализация визуализации такой технологии является очень

важным и необходимым. Забегая вперед, отметим, что именно поэтому для этой задачи стоит требование создания отдельного плагина, а не просто модуля, как для остальных задач.

В заключении этой главы, стоит сказать, что рассмотрены все инструменты, которые понадобятся при реализации, также теоретически разобраны все важные моменты, технологии и методы. Очевидно, что та теоретическая часть, которая имеется у автора данной дипломной работы, позволяет приступить к практической реализации.

## РАСЧЁТНО-ПРАКТИЧЕСКАЯ ЧАСТЬ

### 3.1 Описание модуля построения древовидного отображения

Первая задача, которая была выполнена в ходе всей работы – разработка специализированных веб-ориентированных программных модулей, обеспечивающих автоматизацию построения GUI на основе данных в БД.

Это достаточно обширная и глобальная задача, которую можно разделить на несколько подзадач. Первая связана с древовидным представлением, а вторая – с сгруппированным. Об этих подзадачах и созданных модулях далее и пойдет речь.

Задача данного параграфа заключается в том, чтобы обеспечить возможность древовидного представления данных субплагинов (\*sys.aitms, sys.aitms.aclss=SBPLDB\*), построенных на основе таблицы БД.

Древовидное представление данных таблицы БД возможно в случае, когда для таблицы-источника определен внешний ключ, ссылающийся на эту же таблицу. В качестве примеров таких таблиц следует обратить внимание на таблицы: \*sys.mmenu, sys.atyps, fin.accpl\* и многие другие.

Для решения этой задачи реализованы модули, связанные с серверной частью и визуальной frontend частью.

#### 3.1.1 Серверная часть

Для реализации серверной части этого модуля и остальных модулей, представленных далее, использовался язык Python с фреймворком этого языка Django. Для того, чтобы создать графическое отображение, в этой работе сначала были получены данные, путем запроса к базе данных PostgreSQL.

Для того, чтобы представить данные в древовидном отображении, реализована проверка. При получении данных узнается, является ли эта таблица пригодной для построения в нужной древовидной структуре. Если ответ положительный – то вызывается шаблон, который написан с помощью языка HTML5 и Django, о котором речь пойдет в визуальной части.

Пример проверки и вызова шаблона tree\_table\_template.html, в котором проверяется наличие ошибки и типа модели отображения:

```

context = get_table_data(request, data)
if 'error' not in context.keys():
    if 'tree_model' in context.keys():
        return render(request, 'tree_table_template.html', context)

```

Соответственно, после проверки происходит вызов шаблона, в котором осуществляется графическое отображение.

Но, прежде чем перейти к визуальной части, следует уделить внимание тому, что данные нужно правильно представить в структурном смысле. Под этим понимается тип данных, в котором содержится информация.

В ходе разработки был выбран такой тип данных для структурного представления, как справочник, или как его еще иногда называют – контейнер тар. Справочник удобен для данных, которые можно представить в качестве пары ключ – значение.

В данной задаче справочник был сложным, то есть значения в тар могли включать еще несколько значений, а те еще несколько значений, и так, пока не закончатся их ответвления. На рис. 3.1 представлена небольшая часть такого справочника.

```

[
  { 'all_attributes': ['COMAPS', '', 'Службные модули сервера', ''], 'children': [] },
  { 'all_attributes': ['COMWPC', '', 'Службные модули веб-клиента', ''], 'children': [] },
  { 'all_attributes': ['GCDBS', '', 'Модули ввода данных в БД', ''], 'children': [] },
  { 'all_attributes': ['GCDBSFEA', '', 'ВЫЧИСЛИТЕЛЬНАЯ ПОДСИСТЕМА', ''], 'children':
    [
      { 'all_attributes': ['CMPNETMDLS', 'GCDBSFEA', 'Сетевые модели алгоритмов численных методов', ''], 'children': [] },
      { 'all_attributes': ['CMPSLVMTHD', 'GCDBSFEA', 'Вычислительные методы и решатели', ''], 'children': [] },
      { 'all_attributes': ['CMPSLVPREP', 'GCDBSFEA', 'Препроцессоры решателей вычислительной подсистемы', ''], 'children': [] },
      { 'all_attributes': ['CMPSYSDATA', 'GCDBSFEA', 'Опции и настройки вычислительной подсистемы', ''], 'children': [] },
      { 'all_attributes': ['FEMDATA', 'GCDBSFEA', 'Настройки реализации МКЭ', ''], 'children':
        [
          { 'all_attributes': ['GCDBNLC', 'FEMDATA', 'нелинейные критерии', ''], 'children': [] }
        ]
      }
    ]
  }
]

```

Рис. 3.1. Пример структуры справочника

Как видно на рисунке, справочник представлял собой для каждой строки таблицы список всех атрибутов, включая название, имя родителя, если имеется, описание самой функции, а также список детей. И, если внимательно просмотреть иллюстрацию, то прекрасно видно, что у первого, второго и третьего элемента нет ответвлений (то есть детей), а у четвертого элемента – дети есть, и даже несколько. Также один из дочерних элементов имеет еще свои дочерние элементы.



В целом, такой справочник еще хорошо подходит для применения рекурсивных функций, с чем далее и придется работать.

За счет такой структуры, возможно отобразить все эти элементы в удобном древовидном представлении, которое может сворачиваться и разворачиваться, чтобы отобразить все элементы соответственно.

Как было отмечено, для реализации такого справочника необходимо использовать функцию с рекурсией, которая проверяет наличие дочерних элементов. Пример кода описания такой функции:

```
def recur_build_children_nodes(parent_fk_values):
    children_tree_nodes = []
    if len(parent_fk_values) == 0:
        #выполняем действия по заполнению значениями
        #добавляем данные к дочерним
        children_tree_nodes.append(tree_node_data)
    return children_tree_nodes
#затем, вызываем эту функцию еще раз рекурсивно
return recur_build_children_nodes({})
```

После выполнения этой части, в шаблон передается структурированные данные, которые обрабатываются и помещаются в теги языка HTML5, затем представляются пользователю в хорошем и удобном виде.

Далее перейдем к описанию самих шаблонов. Как уже было сказано, они написаны на Django template. Для этой задачи было создано 2 модуля, которые взаимодействуют друг с другом:

- build\_tree.html
- tree\_table\_template.html

Первый шаблон отвечает за непосредственное построение древовидного списка. В нем происходит выделение из строк данных нужных значений, в частности, это первичный ключ и остальные атрибуты. Эти значения и атрибуты помещаются в html теги, в данной задаче – в теги <td>, что означает элементы таблицы.

Как только встречается элемент с наличием дочерних элементов, так сразу этот шаблон вызывается еще раз, то есть рекурсивно. И такая процедура происходит для каждого элемента, который впоследствии превращается в узел

древовидного списка. Так как имеет место рекурсия, то именно поэтому и был создан отдельный шаблон для построения представления.

Этот шаблон подключается к основному шаблону, который называется `tree_table_template.html`. Подключение происходит с помощью команды `include` примерно таким образом:

```
{% include 'build_tree.html' %}
```

А второй шаблон уже отображает, то что построилось в предыдущем шаблоне, помещает все содержимое в основную часть таблицы, а также осуществляет отображение заголовков и описания к таблице.

### 3.1.2 Визуальная часть

В визуальной части этой задачи важной проблемой являлось то, как отображать этот список, чтобы было понятно и удобно, прежде всего, пользователю, так как вся разработка направлена на улучшение пользовательского восприятия системы. А также следовало помнить об адаптивности и динамичности предоставляемых данных, что является немаловажной задачей.

Эти проблемы успешно преодолены благодаря использованию плагина JavaScript – `treetable.js`, который был успешно интегрирован в систему. Этот внешний плагин позволил настроить отображение таблицы таким образом, чтобы атрибуты выравнивались по колонкам и создавали реальную таблицу для пользователя.

Первичные ключи же при смене уровня вложенности смещались вправо на определенное значение. Этот подход получился очень оригинальным, а что самое главное – удобным.

Также использовались свои скрипты, написанные на языке JavaScript, которые позволяли контролировать элементы как плагина, так и всего шаблона построения древовидного отображения.

Помимо скриптовой части, достаточно большое количество времени уделено стилям, которые реализуются через CSS3. Подбирались стили ко многим элементам интерфейса. Важным было осуществить правильное выравнивание по ширине для всех колонок таблицы, которое осуществил плагин

jQuery. Также под пользователя был выбран шрифт, а именно его тип, размер и начертание, для лучшей пользовательской визуализации.

Все это вместе образует цельную картину такого списка и правильно его отображает – со всеми атрибутами и их значениями. Построение такого графического интерфейса оказалось очень удобным и, что самое главное, динамическим и оптимизированным, с точки зрения скорости построения. Пример отображения древовидного списка можно увидеть на рис. 3.2, где показан фрагмент одной из таблиц.

В ходе разработки были созданы следующие файлы:

- tree\_view.css – файл стилей;
- tree\_view.js – файл скриптов;
- treetable.js – файл скриптов для плагина;

В этих файлах написан достаточно тривиальный код, который часто используется в современной веб-разработке, поэтому в этой части примера кода не будет. Весь код будет вынесен в приложение к данной дипломной работе.

## Иерархия типов Action Items (sys.atyps)

atype	atdes	dscrbr
⊕ GCDTCH	ИСПЫТАНИЯ И ПРОИЗВОДСТВО	Подсистема натуральных испытаний и технологий производства КМ
⊖ SYSTEM	Ядро системы (Read Only)	
SYSLIBS	Системные библиотеки	Системные библиотеки: cls_INIParser, cls_AnyMap
⊕ HIDDEN	Скрытые функции	
LASSLV	Решатели СЛАУ	
GCDDBS	Модули ввода данных в БД	
⊕ GCDDBSSLD	СВОЙСТВА МАТЕРИАЛОВ	Верифицированные характеристики материалов
⊕ PROJECTMNG	Управление проектами	
⊕ RNDSUBSYS	Исследования и разработки	
⊖ RNDCHM	Корневое меню ПК ХТС (RNDCHM)	Корневое меню программного комплекса химико-технологических систем (ПК ХТС)
CHMTST	Химические эксперименты	Функции для ввода данных по химическим экспериментам
CHMATMDATA	Функции ввода значений атрибутов атомов	Функции ввода значений атрибутов атомов
CHMSBT	Данные о веществах	Функции работы с данными о веществах (обобщенных)
CHMCHE	Химические реакции	Функции регистрации исследуемых химических реакций.

Рис. 3.2. Пример отображения построенного древовидного списка

Данная задача оказалась достаточно трудоемкой и не самой легкой для разработчика, так как стояла первой в ходе всей работы и разработчику, несомненно было необходимо понять структуру проекта.

Но тем не менее, можно с уверенностью отметить, что выполнение этой работы действительно увеличило полезность графического интерфейса. С точки зрения пользователя, графический интерфейс стал более удобным и визуально понятным.

### **3.2 Описание модуля построения сгруппированного отображения**

Как и задача 3.1, эта связана с данными, представленными в БД. Только в данном случае необходимо данные представлять в виде, отсортированном и сгруппированном по одному или нескольким параметрам.

Есть таблицы с очень большим числом записей (результаты расчетов характеристик композиционных материалов `src.cmpat`), но большинство этих таблиц имеют сложные составные первичные ключи, т.е. такая разработка позволит существенно ускорить работу с такими данными.

#### **3.2.1 Серверная часть**

Как и в задаче для древовидного представления, в этой задаче также был доработан плагин `handle_table.py`, который вызывал шаблон `group_table_template.html`. Изначально в этот шаблон с помощью Python передавался словарь, который содержал название колонок, по которым будет производиться группировка, а также возможные значения для каждого из параметров. Общая структура выглядит таким образом:

```
filtered_pk_column_all_values = {'PK_COL_NAME_1': ['VAL1', 'VAL2', ...],  
'PK_COL_NAME_2': ['VAL1', 'VAL2', ...]}
```

Затем, когда пользователь, выберет значения, по которым будет осуществляться группировка, то на сервер поступит ajax-запрос. В случае успешного выполнения запроса и выборки данных из базы данных по заданным параметрам, этот запрос вернет в шаблон новый словарь с данными, которые нужно отобразить в табличном виде. Общая структура полученных данных:

```
filtered_rows_list = [{'COL_NAME': 'VALUE'}]
[{'COL_NAME_1': 'VALUE', 'COL_NAME_2': 'VALUE', 'COL_NAME_3': 'VALUE'},
 {'COL_NAME_1': 'VALUE', 'COL_NAME_2': 'VALUE', 'COL_NAME_3': 'VALUE'} ...]
```

То есть данные представляют собой строки со значениями для каждой отображаемой колонки.

### 3.2.2 Визуальная часть

Эта часть заключалась в создании шаблона, который отображает данные в удобном виде. Был разработан шаблон `group_table_template.html`, который помимо отображения, еще и отправлял данные для аякс-запроса.

Если затронуть подробнее разработку шаблона, то для начала было необходимо создать форму, для выбора пользователем определенных параметров группировки. Такая форма была реализована и является динамической. Пример такой формы можно увидеть на рис. 3.3.

Рис. 3.3. Пример формы для выбора параметров группировки

После выбора пользователем значений, на сервер отправляется запрос с данными и возвращаются нужные для отображения данные, для которых использовался табличный вид. Табличный вид представлен на рис. 3.4.

Рис. 3.4. Сгруппированное отображение

Помимо шаблона, необходимо было разработать стили и скрипты, которые существенно повысили уровень GUI, и в данной задаче были разработаны:

- `group_view.css` – файл стилей, который применялся к элементам формы и таблицы;
- `group_view.js` – файл скриптов, в котором осуществлялся сбор данных для ajax-запроса, а также парсинг полученных данных и заполнение таблицы;

Подводя итоги этой задачи, следует отметить, что она оказалась достаточно простой в реализации, и прежде всего, это связано с накопленным опытом, полученным в ходе разработки предыдущих задач.

### **3.3 Описание модуля построения графических представлений вычислительных инструментов**

В рамках этой задачи требовалось разработать специализированные веб-ориентированные программные модули, обеспечивающие автоматизацию построения графических представлений вычислительных инструментов («решателей»), применяемых для решения задач, в рамках технологии GBSE.

Задачу нужно реализовать с помощью написанного по стандартам данного веб-клиента плагина, который и будет осуществлять все необходимые действия, в частности, вызывать шаблон для отображения пользователю.

#### **3.3.1 Выбор и настройка плагина для реализации**

Итак, изучив основные понятия графо-ориентированной технологии в теоретической части, следовало приступить к подготовке используемых инструментов. Очевидно, что поставленная задача достаточно уникальна и не очень проста в реализации. Из этого следует, что стандартными средствами веб-разработки не обойтись, так как, на сегодняшний день, нет таких систем в веб-пространстве, которые бы реализовали это стандартными методами.

Все это натолкнуло на мысль использования плагина для построения ориентированных и неориентированных графов. Выбор плагина осуществлялся достаточно продолжительное время. Это было связано с тем, что существует порядка десяти хороших и качественных разработок, которые, как ни странно,

отличаются своей функциональностью.

Выбор для данной задачи остановился на плагине vis.js. Этот плагин позволяет осуществлять разнообразные действия с графами, также, что очень важно, есть возможность использовать язык html при создании элементов.

Еще одним преимуществом данного плагина является то, что он предусматривает работу с графикой и визуальными представлениями данных, что в будущем, несомненно, может понадобится всему веб-клиенту.

Далее приведем общую структуру кода работы для этого плагина, которая реализуется на языке JavaScript. На рис. 3.5 можно увидеть общие настройки для узлов и ребер сетевой модели.

```
var options = {
  autoResize: true,
  height: '100%',
  width: '100%',
  locale: 'en',
  locales: locales,
  clickToUse: false,
  configure: {...}, // defined in the configure module.
  edges: {...}, // defined in the edges module.
  nodes: {...}, // defined in the nodes module.
  groups: {...}, // defined in the groups module.
  layout: {...}, // defined in the layout module.
  interaction: {...}, // defined in the interaction module.
  manipulation: {...}, // defined in the manipulation module.
  physics: {...}, // defined in the physics module.
}

network.setOptions(options);
```

Рис. 3.5. Настройки параметров сетевой модели в плагине

Соответственно, эти настройки применялись в данной работе, особенно настройки, связанные со стилями узлов, по причине того, что тот формат отображения, который был выбран и который будет описан далее, требовал очень тонкой стилистической настройки.

### 3.3.2 Серверная часть

Самым важным вопросом в этой части являлось то, в каком формате предоставлять данные в шаблон, в визуальную часть. Так как выбранный плагин работает с dot форматом, то было принято решение использовать это.

В итоге был выбран формат данных – справочник, который содержал несколько справочников внутри себя. Доступ к ним осуществлялся по ключам:

- `dot_string` – хранит связь узлов и предикатов в формате `dot`;
- `predicates_description` – описывает предикаты, их названия;
- `states_description` – описывает узлы, их названия;
- `state_predicate_links` – описывает связи ключевых значений узлов со значениями предикатов;

Эти данные передавались в шаблон `graph_visualizer.html`, после того как осуществляется `ajax`-запрос к серверу. Возврат данных происходит таким образом:

```
return render(request, 'graph_visualizer.html', context)
```

Где `context` представляет собой:

```
context = {  
    'action_item': 'graph_visualizer',  
    'action_class': 'PYWEBS',  
    'solvers': solvers  
}
```

В `solvers` хранится тот решатель, который выбрал пользователь для построения. Соответственно с помощью `return render`, в шаблон передается справочник, описанный выше.

### 3.3.3 Визуальная часть

В соответствии с тем, что отображаемые графы должны были демонстрировать процесс выполнения алгоритма для определенного решателя, выбранного пользователем, была выбрана определенная модель отображения в плагине `vis.js`.

Эта модель имела название иерархическая и представляла такое отображение, при котором начальный узел алгоритма находился выше всех остальных, а узел окончания алгоритма решателя – соответственно ниже всех. Такая структура хороша подходит для того, чтобы понятно и правильно



отобразить алгоритм решателя.

Чтобы установить такую модель необходимо задать параметры в скриптовой части кода. В данной работе использованы параметры графовой модели, которые представлены в табл.3.1. В данной таблице указаны только самые основные и, соответственно, представляющие наибольшую ценность для отображения сетевой модели параметры.

Таблица 3.1

### Основные настройки узлов графовой модели

Свойство	Пояснение	Значение
enabled	В зависимости от значения включает или выключает режим отображения	true
levelSeparation	Устанавливает расстояние между уровнями для узлов модели	220
nodeSpacing	Устанавливает расстояние между узлами на одном уровне	320
edgeMinimization	Устанавливает процесс минимизации занимаемого пространства для всей модели	true
direction	Устанавливает направление для узлов графа	UD
parentCentralization	Устанавливает выравнивание положения узлов в зависимости от родительских элементов	false

Общее изображение графовой или, по-другому, сетевой модели представлено на рис. 3.6.

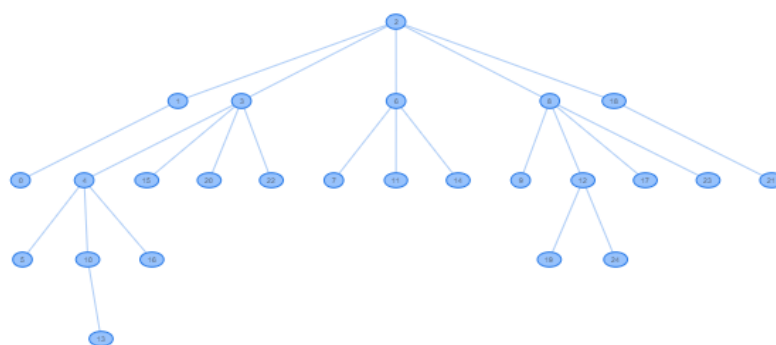


Рис. 3.6. Иерархическая структура сетевой модели плагина

После выбора иерархической модели, следующим моментом, который требовал раздумий, оказался выбор правильного отображения узлов графовой модели. В данных моделях связь между узлами происходила только после обработки предиката. Предикат по своей сути представляет условный оператор, выполнение которого означает переход через предикат к следующему узлу графа.

Основная проблема заключалась в том, что существуют узлы с одним предикатом, а также с несколькими предикатами. И было необходимо разработать модель отображения для различных вариантов.

Такая модель для отображения разработана. Основная суть заключается в том, что узел с одиночным предикатом отображается слитно все с тем же предикатом, а узел с множественными предикатами – представляет собой связь узла с этими предикатами, похожую на связь между узлами. Также было принято отделить узел и предикат цветами для лучшего восприятия пользователем. На рис. 3.7 представлены разработанные виды узлов.



Рис. 3.7. Узлы с одиночным предикатом (слева) и с несколькими предикатами (справа)

Подготовив данные в серверной части и установив параметры для плагина, можно было приступить к реализации. Прежде всего был создан шаблон для отображения. В этом шаблоне изначально отображался элемент формы выпадающего списка, в котором пользователь может выбрать решатель для отображения. После выбора пользователя происходило отображение графовой модели.

В данной задаче большинство кода написано на JavaScript, так как основные действия пользователя и настройки параметров обрабатывались на этом языке. Для реализации использовалась технология аjax-запросов, которая

осуществляла запросы к серверу. Полученные данные после запроса представлялись пользователю в удобном виде.

Также важную роль в реализации играет обработчик действия, реализованный с помощью функции jQuery «on.click», который выполняет предназначенный блок кода только при определенном действии пользователя, в данном случае при нажатии на какой-либо элемент.

Результатом созданных модулей является графическое представление, с которым можно ознакомиться на рис. 3.8 и 3.9, где представлены основные модели решателей.

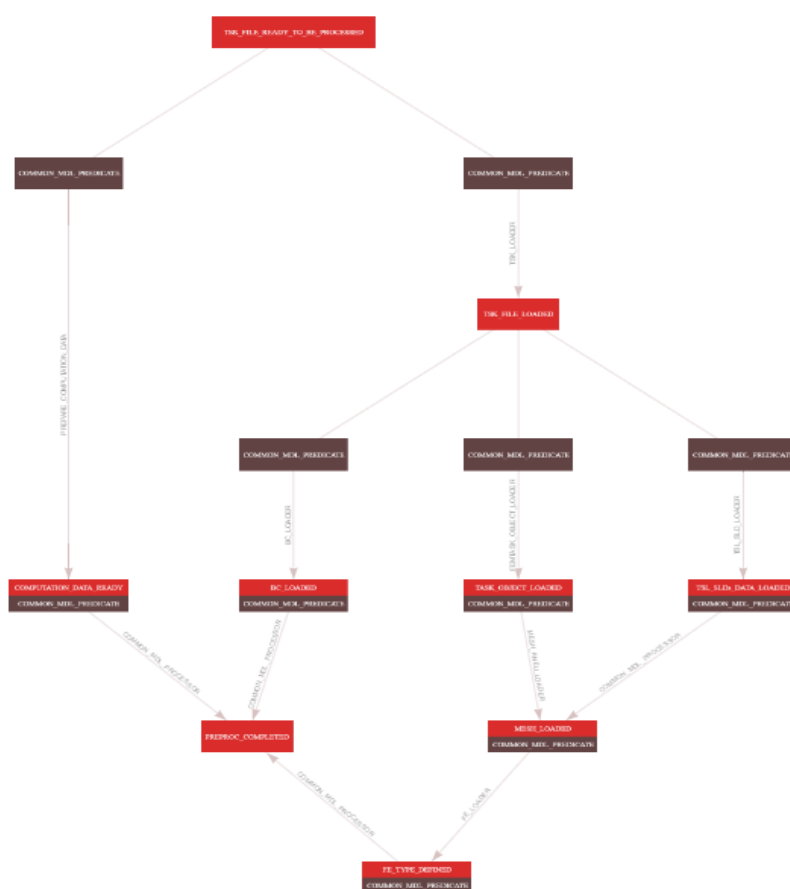


Рис. 3.8. Отображение сложной модели с разветвлением при выполнении алгоритма

Как уже было сказано, некоторые модели могут содержать в себе и другие, более простые модели и, также содержать параллельные решения определенных вычислительных задач.

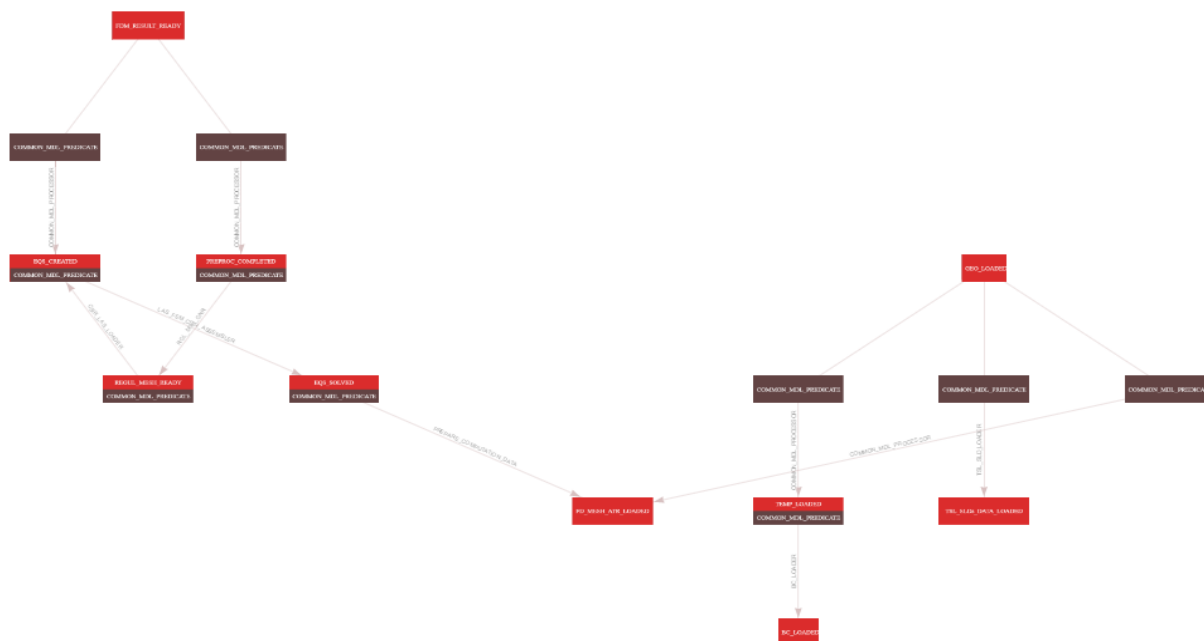


Рис. 3.9. Отображение сложной модели с совокупность решений

Итак, самая трудоемкая и оригинальная задача была решена. И можно сказать, что решена так, как она и ставилась. То есть были разработаны модули, которые связаны с плагином.

### 3.4 Тестирование созданных GUI в рамках существующей программной системы: выявление особенностей построения, оптимизация используемых подходов

После выполнения задачи с построением древовидного и группированного представления данных из существующих баз данных, возникла задача оптимизации исходных ресурсов, в частности обобщения кода стилей (.css) и скриптов (.js).

Для выполнения этой задачи потребовалось создание и сортировка по своей основной функциональности всех модулей, написанных на представленных выше языках программирования, которые непосредственно относились к табличным представлениям.

Эта задача возникла благодаря тому, что, то представление, которое было разработано в ходе выполнения этой дипломной работы, было схоже с обычной таблицей и, очевидно, что некоторые элементы возможно и нужно объединить.

### 3.4.1 Обобщение кода стилового оформления

Для начала выполнения общей задачи, следовало понять, что написание единого общего кода – практически невозможная задача. Поэтому для обобщения мною были выбраны файлы стилей.

Для этого обобщения, самым важным являлось, то, что разные стили, которые применялись к нескольким представлениям, в том числе и древовидному, нужно, как можно лучше структурировать в отдельный файл. А те стили, которые по своей функциональности не могли быть обобщенными, необходимо было выделить в другие файлы, чтобы разработчики этого веб-клиента могли понять, что делает тот или иной файл.

Решение этой задачи получилось благодаря использованию классов в html шаблонах. Как известно стили применяются к классам, и поэтому, для одинаковых элементов графического представления оказалось возможным применить один и тот же класс. А уже к классу применялись нужные стили.

Далее приведем список классов, которые являются общими для нескольких представлений:

- *background-pk* – класс, отвечающий за выделение первичного ключа цветом, что визуально оказывает положительное влияние на пользователя;
- *head-pk* – класс, который отвечает за инициализацию первичного ключа в заголовке представлений, то есть можно обратиться к первичному ключу с помощью этого класса;
- *head-fk* – класс, который отвечает за инициализацию внешнего ключа в заголовке таблицы. Назначение такое же, что и для первичного ключа;
- *head-attr* – отвечает за инициализацию атрибутов представлений в заголовках;
- *head-col-descr* – инициализация описания для ключей и атрибутов в заголовках;

Также были выделены классы отдельно как для обычного представления данных, так и для древовидного.

Помимо создания различных по значению классов, стили были выделены в отдельные файлы, что является очень удобным и структурным подходом. Изначально все стили были описаны в файле `common_main.css`. Теперь создано

несколько файлов, обладающие своей смысловой нагрузкой.

Созданы следующие файлы:

- *table.css* – файл, в котором описаны стили для всех элементов таблиц, которые, в свою очередь, являются общими для разных представлений, а также затронуты стили заголовков всех таблиц и других элементов, в частности: модального окна, пример которого изображен на рис. 3.10 и кнопок действия;
- *tree\_view.css* – файл для стилового описания древовидного представления, включающий описание тех классов элементов, которые являются уникальными для данного представления, а также описание изменения некоторых стилей, необходимых для такого отображения;

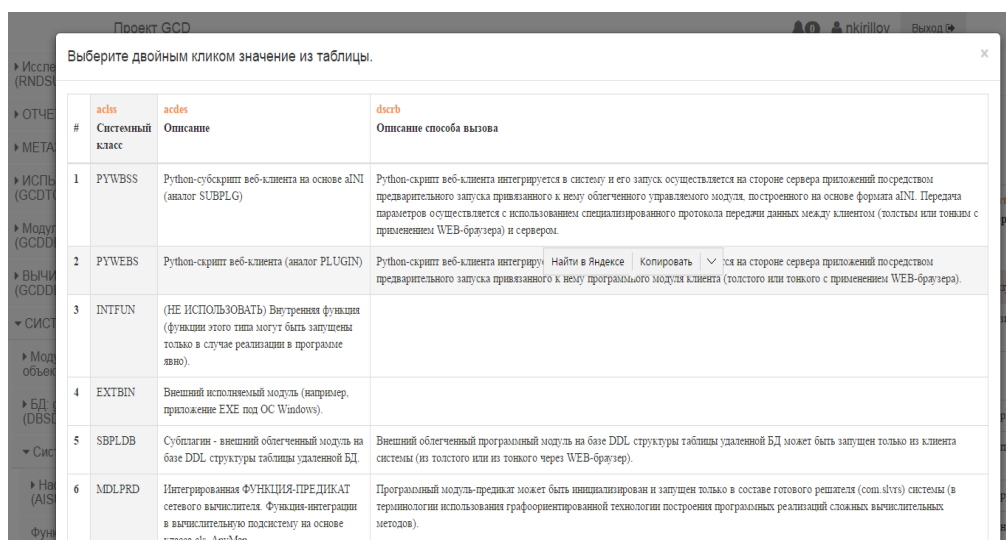


Рис. 3.10. Пример отображения таблицы, вызванной по внешнему ключу, в модальном окне

### 3.4.2 Создание общих шаблонов

Для оптимизации кода и улучшения дальнейшей разработки, было принято решение создать общий шаблон, который отобразил бы заголовки к таблицам древовидного и обычного представлений.

Единственным ограничением являлся тот факт, что при древовидном представлении данных столбцы с внешним ключом не отображались, по причине избыточности данных.

Эта проблема была успешно решена с помощью использования условных операторов Django в шаблонах и последующим отображением заголовков

таблиц.

Итак, был разработан шаблон `head_table.html`, в котором осуществлялось построение заголовочной части любой таблицы. Как уже было написано, использовался следующий оператор для обработки разных представлений:

```
{% if column in fk_columns_list% }  
    {% if not tree_model % }
```

В этих двух строках кода происходит проверка: сначала проверяется тот факт, что относится ли колонка к внешнему ключу. А затем, если в результате возвращается `true`, то происходит выполнение 2 строчки, где проверяется как раз тип представления: древовидный или обычный.

Также, важно отметить, что были разработаны классы для элементов этой части, которые описаны в пункте 3.3.1, а также применены стили, которые осуществляют визуальное отделение структурных частей ячейки заголовков таблицы данных, что, несомненно повышает уровень пользовательского графического интерфейса.

Таким образом, разработан шаблон, который строит часть таблицы с заголовками. Пример такого построения можно наблюдать на рис. 3.11, для таблицы `sys.aitms`.

#	<b>aisid</b> Идентификатор функции (Action Item)	<b>atype</b> Предметный тип функции	<b>aclss</b> Системный класс функции	<b>aides</b> Описание модуля	<b>cxid</b> Комплекс	<b>slnid</b> Решение	<b>prjid</b> Проект
---	---	---	---	---------------------------------	-------------------------	-------------------------	------------------------

Рис. 3.11. Пример работы шаблона `head_table.html`

На этом рисунке представлена часть, которую отображает разработанный шаблон. Также можно заметить, что с помощью цвета осуществляется отделение наименования колонки таблицы от основного описания этого наименования.

### 3.4.3 Структурирование кода скриптовой части клиента

В заключении последует описание самой главной и очень трудоемкой части этой задачи. Как и для файлов стилей, на веб-клиенте основная скриптовая часть была практически представлена в одном файле под названием `common_main.js`, что опять же является не лучшим подходом. По этой причине

были разработаны дополнительно несколько файлов:

- *table.js* – общий файл скриптов, в котором описаны все обработчики событий для табличных данных, также файл содержит описания данных в json формате, которые посредством ajax запросов отправляются на сервер веб-клиента;
- *tree\_view.js* – файл, в котором описаны структурные изменения в DOM (Document Object Model) древовидного представления, которое затрагивает основные элементы построения, а также частичное описание работы плагина *treetable.js*, который использовался в параграфе 3.1;

Помимо обычного выделения функция в отдельные файлы проделана работа по восстановлению функциональности веб-клиента. Как оказалось, после создания нового отображения заголовков с наименованием и описанием перестали работать основные функции, связанные с таблицами в обычном представлении. К таким основным функциям можно отнести удаление и добавление строки, изменение значения в ячейки таблицы.

На изменении остановимся чуть подробнее. Изменение значения ячейки таблицы осуществляется по двойному нажатию на эту ячейку (событие *double click*), после этого пользователь может изменить, что он захочет. Этот процесс может происходить в двух вариантах.

Первый вариант состоит в том, что пользователь кликнул на ячейку столбца внешних ключей. В таком случае отправляется запрос на получение таблицы, на которую этот ключ ссылается. Открывается модальное окно, которое уже было представлено на рис. 3.10, и в нем пользователь может выбрать любое значение.

Второй вариант, это когда пользователь дважды кликнет на ячейку из столбца с атрибутами. Очевидно, что в этом случае, нет никакой ссылки на другую таблицу, и тогда пользователю предлагается ввести значение самому в форму с *input-box*.

Обоим этим вариантам необходимо знать какую колонку с ячейкой выбрал пользователь для изменения, а также все наименования заголовков таблицы для отправки запроса на сервер. Но после изменения заголовков, эти функции стали работать некорректно.

Это связано с тем, что данные которые отправлялись посредством ajax,



теперь структурировались в неправильном виде. Теперь, помимо наименования, в данные заносились и описания этих значений.

Для решения этих проблем, в первую очередь, пришлось осуществлять выделение и отделение в файлах шаблонов html наименований колонок и описания друг от друга с помощью тегов <span> с соответствующими классами.

Затем в файлах скриптовой части были переписаны функции подготовки данных в json формате для отправки на сервер через ajax запрос. Основные моменты заключались в правильном представлении этих данных, в которых не должно присутствовать описание колонок.

После изменения этой функции была проделана достаточно трудоемкая работа по переделыванию функций-обработчиков событий, привязанных к ячейкам таблиц с данными. Были затронуты функции для обоих вариантов изменения значения, а также изменены классы, которые добавлялись к данным в случае успешного или неуспешного запроса на изменение.

В случае успешного изменения строка, в которой это изменение произошло подсвечивалась зеленым цветом в течении 2 секунд, с момента изменения. Пример приведен на рис. 3.12:

#	aisid	atype	aclss	aides	cpxid	slnid	prjid	shrtx
	Идентификатор функции (Action Item)	Предметный тип функции	Системный класс функции	Описание модуля	Комплекс	Решение	Проект	Короткий тек
1	ACKLEY_FUNCITON	COMAPS	EXTBIN	Вычисление функции Эклика	cmc	fes	tst	
2	ADD_USER_WPC	META3	MDLPRD	Добавление пользователей	com	wpc	asd	
3	ANEU_TO_BINARY	COMAPS	PLUGIN	Выгрузка сетки формата ANEU в бинарный файлах	edu	adh		

Рис. 3.12. Пример успешного изменения данных в таблице с обычным представлением

В ходе работы были использованы различные функции как JavaScript, так и jQuery, в частности, функции работы со строками и объектами как созданными, так и элементами документа.

В целом, вся задача была направлена на выявление возможностей улучшения тех подходов, которые использовались на веб-клиенте, и можно утверждать, что данная разработка повысила уровень организации функций и модулей системы, что всегда является первостепенной задачей для любой вычислительной системы.

### **3.5 Руководство программиста**

Программные модули для построения и визуализации пользовательских графических интерфейсов для различных вычислительных методов и модулей системы инженерного анализа.

#### **3.5.1 Назначение и условия применения программы**

Созданные модули предназначены для визуализации данных пользователю. В зависимости от модуля, данные представляются в различных видах: структурном древовидном представлении, табличном сгруппированном, в виде графовой модели.

Условия применения состоят в наличии данных для отображения, которые передаются из серверной части, а те, в свою очередь, из базы данных. Другими словами, нужно обладать правами доступа. Также необходимо подключение к интернету.

Требования к периферийным устройствам очень непритязательные, так как все вычисления и другие действия происходят на удаленном сервере. Процессор Intel Core 2 Duo CPU T7700 2.40 GHz и выше, оперативная память: от 2 Гб.

В качестве программного обеспечения требуется операционная система Windows или Linux. Также необходим любой браузер, версия, которого разработана позже 2013 года.

#### **3.5.2 Характеристика программы**

Как и большинство модулей системы, разработанные модули работают посредством технологии ajax-протокола. При нажатии пользователем пункта в главном меню, подгружается соответствующий модуль, который взаимодействует с пользователем.

Модуль для представлений отображает данные в нужном виде, а также при действии пользователя, изменяет эти данные как в базе данных, так и для пользователя в этот же момент.

Модуль технологии GBSE в самом начале отображает элемент формы

селектор, в котором пользователь выбирает какой решатель отобразить. При нажатии на кнопку «применить», осуществляется построение и отображение графовой модели.

### 3.5.3 Обращение к программе

Обращение к модулям происходит в веб-клиенте, для этого нужно в браузере зайти на соответствующий домен. И, если имеется права доступа к системе, то при нажатии на пункт главного меню произойдет обращение к модулю.

Программное обращение к самим модулям представлений осуществляется так из других модулей:

```
{% include 'build_tree.html' % }
```

В файлах на языке Python, в качестве параметра функции:

```
return render(request, 'grouped_table_template.html', context)
```

### 3.5.4 Входные и выходные данные

В качестве входных данных выступают данные из базы данных веб-клиента.

Для древовидного представления выходные данные являются справочником, в котором описаны атрибуты и дочерние элементы всех родительских элементов.

Для группового представления выходными являются данные, полученные в результате взаимодействия с пользователем, который установил параметры ключей, по которым происходит группирование.

Для графовой модели выходными являются данные, которые представляют справочник, хранящий описание узлов, предикатов и связей между узлами и множественными предикатами.

## ВЫВОДЫ

- Изучена технология GBSE, знания о которой понадобились для построения GUI на основе графовой модели.
- Соответственно, разработаны веб-ориентированные программные модули, обеспечивающие автоматизацию построения графических представлений вычислительных инструментов ("решателей").
- Разработаны специализированные веб-ориентированные программные модули, обеспечивающие автоматизацию построения GUI на основе данных в БД, для различных представлений.
- Проведено тестирование созданных GUI в рамках существующей программной системы: выявлены особенности построения, преимущества и недостатки используемых подходов.
- Оптимизирована структура файлов стилей и скриптов веб-клиента с помощью обобщения кода и написания новых функций, в соответствии с результатами тестирования созданных GUI.

## СПИСОК ЛИТЕРАТУРЫ

1. Википедия [Электронный ресурс]. - Режим доступа: [https://ru.wikipedia.org/wiki/Графический\\_интерфейс\\_пользователя](https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя) (дата обращения: 12.03.2017).
2. Bowen J., Reeves S., Formal Models for informal GUI designs// Electronic Notes in Theoretical Computer Science. 2007. Volume 183. P. 57 – 72.
3. Carino S., Andrews. H. J., Dynamically Testing Graphical User Interfaces// Electronic Thesis and Dissertation Repository. 2016. Paper 3476. P. 29 – 34.
4. Maul M. GUI – Graphical User Interface// Arbeitspapiere WI. 1998. P. 3 – 8.
5. Ходаков В. Е., Величко Ю. И., Компоненты адаптации пользовательского интерфейса// Вестник ХНТУ. – 2011. – №2(41). – С. 276 – 281.
6. Silva J. L., Campos J. C., Model-based User Interface Testing with Spec Explorer and Concur Task Trees// Electronic Notes in Theoretical Computer Science. 2007. P. 1 – 17.
7. Beg M., Pepper R. A., Fangohr H., User interfaces for computational science: a domain specific language for OOMMF embedded in Python// Faculty of Engineering and the Environment. 2017. P. 1 – 3.
8. Попов А. А., Эргономика пользовательских интерфейсов в информационных системах: учеб. пособие. – М.: РУСАЙНС, 2016. – 312 с.
9. Слива М. В., Прототипирование графического интерфейса пользователя как неотъемлемая часть процесса разработки программного обеспечения// Вестник Нижневартковского государственного университета. – 2013. – С. 16 – 19.
10. Сайт для разработчиков IBM [Электронный ресурс]. - Режим доступа: <http://www.ibm.com/developerworks/ru/library/wa-websiteapp/index.html> (дата обращения: 23.03.2017).

11. Компьютерная грамотность для начинающих [Электронный ресурс]. - Режим доступа: <http://www.pc-school.ru/chto-takoe-brauzer> (дата обращения: 24.03.2017).
12. Хабрахабр [Электронный ресурс]. - Режим доступа: <https://habrahabr.ru/post/267721/> (дата обращения: 24.03.2017).
13. ИНТУИТ национальный открытый университет [Электронный ресурс]. - Режим доступа: <http://www.intuit.ru/studies/courses/64/64/lecture/1892?page=1> (дата обращения: 24.03.2017).
14. Безъязыкова Н. А., Яковлева М. С., Багаева А. П. Насыщенные интернет приложения // Актуальные проблемы авиации и космонавтики. – 2015. – №11. – С. 514 – 515.
15. Коршунов С. А., Павлов А. И., Николайчук О. А. WEB-ориентированный компонент продукционной экспертной системы // Программные продукты и системы. – 2015. – №2. – С. 20 – 22.
16. Борсук Н. А., Гартман В. А., Кургузов С. Д. Анализ средств разработки web-страниц // Символ науки. – 2016. – №11. – С. 41 – 42
17. Сайт об интернет приложениях [Электронный ресурс]. - Режим доступа: [http://www.internet-technologies.ru/articles/article\\_1828.html](http://www.internet-technologies.ru/articles/article_1828.html) (дата обращения: 25.03.2017).
18. Официальный сайт языка программирования Python [Электронный ресурс]. - Режим доступа: <https://www.python.org> (дата обращения: 25.05.2017).
19. Интернет ресурс по Django на русском языке [Электронный ресурс]. - Режим доступа: <https://djbook.ru> (дата обращения: 25.05.2017).
20. Алекс Маккоу, Веб-приложения на JavaScript: учеб. пособие., Питер, 2012. – 288 с.

21. Сайт – учебник по веб-программированию [Электронный ресурс]. - Режим доступа: <https://html5book.ru/jquery-selectors/> (дата обращения: 26.05.2017).
22. Официальный сайт фреймворка Twitter Bootstrap [Электронный ресурс]. - Режим доступа: <http://getbootstrap.com> (дата обращения: 27.05.2017).
23. Сайт о системе контроля версиями GIT [Электронный ресурс]. - Режим доступа: <https://git-scm.com> (дата обращения: 27.05.2017).
24. Официальный сайт текстового редактора Brackets [Электронный ресурс]. - Режим доступа: <http://brackets.io> (дата обращения: 29.05.2017).

## **ПРИЛОЖЕНИЕ**



## Листинг исходного кода файла build\_tree.html

```
{% load utils %}
{% for row in tree_model %}
    <tr data-tt-id="{{ id_count }}" {{ forloop.counter }} " data-tt-parent-
id="{{ id_count }}">
        {% for attr in row.all_attributes %}
            {% if header|get_index:forloop.counter0 in primary_key
%}
                <td class="tree-pk background-pk">{{ attr }}</td>
            {% elif header|get_index:forloop.counter0 not in
fk_columns_list %}
                <td class="tree-attr">{{ attr }}</td>
            {% endif %}
        {% endfor %}
    {% if row.children %}
        {% include "build_tree.html" with
tree_model=row.children
id_count=id_count|concat:forloop.counter|add:". "%}
    {% endif %}
    </tr>
{% endfor %}
```

## Листинг исходного кода файла tree\_table\_template.html

```
{% load utils %}
{% load staticfiles %}
<h1 class="page-header"><span class="{{ action_class }}" {{
action_item }}">{{ title }}</span></h1>
<table class="table table-tree table-bordered">
    <thead>
        {% include 'head_table.html' %}
    </thead>
    <tbody>
        {% include 'build_tree.html' %}
    </tbody>
</table>
<script src="{% static 'js/tree_view.js' %}"></script>
```

## Листинг исходного кода файла tree\_view.js

```
function tree_view_modal() {
    $('tr[data-tt-parent-id]').each(function(index,elem){
        var href = $(elem).attr("data-tt-parent-id");
        $(elem).attr("data-tt-parent-id", href.substring( 0, href.length - 1
    ));
    });
    $("tr[data-tt-parent-id='']").removeAttr("data-tt-parent-id");
    $(".table-tree").treetable({ expandable: true });
}
$(function() {
    tree_view_modal();
});
```

## Листинг исходного кода файла tree\_view.css

```
.table-tree {
    width: auto !important;
    max-width: none !important;
    min-width: 100%;
}
.tree-pk{
    border-right: 1px solid #DADBDE;
    padding-left: 26px !important;
    vertical-align: middle !important;
    padding-right: 20px !important;
    font-family: Times New Roman;
    font-size: 1.1em;
}
.tree-attr {
    border-right: 1px solid #DADBDE;
    padding-left: 12px !important;
    vertical-align: middle !important;
    font-family: Times New Roman;
    font-size: 1.1em;
}
.indenter a {
    color: #ff8800;
    margin-left: -19px;
    margin-right: 4px;
    font-family: FontAwesome;
```

```

text-decoration: none;
}

.table-tree tr.collapsed span.indenter a:before {
    content: "\f196";
}

.table-tree tr.expanded span.indenter a:before {
    content: "\f147";
}

```

## Листинг исходного кода функции модуля tree\_view в файле handle\_table.py

```

def build_tree_model(sql_table):
    header = sql_table.getColumnsList()
    rows_list = sql_table.get_rows_list()
    table_full_name = sql_table.info['SCHEMA'] + '.' +
sql_table.info['NAME']
    if table_full_name in sql_table.foreign_keys:
        fk_indexes_to_pk_indexes = {int(fk_col_index): pk_col_index
for fk_col_index, pk_col_index in

sql_table.foreign_keys[table_full_name].items()}

```

```

def recur_build_children_nodes(parent_fk_values):
    children_tree_nodes = []
    if len(parent_fk_values) == 0:
        parent_fk_values = {int(fk_col_index): " for fk_col_index
in

sql_table.foreign_keys[table_full_name].keys()}
        search_info = {header[fk_index]: val for fk_index, val in
parent_fk_values.items()}
        found_rows = sql_table.filter(search_info)
        for mapped_row in found_rows:
            tree_node_data = {}
            row_list = list(mapped_row)
            for column_name, val in mapped_row.items():
                row_list[sql_table.column_to_index(column_name)] =
val

            tree_node_data['all_attributes'] = row_list
            tree_node_data['children'] = recur_build_children_nodes(
                {int(fk_index):
row_list[fk_indexes_to_pk_indexes[int(fk_index)]] for fk_index in
                sql_table.foreign_keys[table_full_name].keys()})
            children_tree_nodes.append(tree_node_data)
        return children_tree_nodes

    return recur_build_children_nodes({})
else:
    raise Exception("Таблица { } не является
древовидной".format(table_full_name))

```

## Листинг исходного кода файла grouped\_table\_template.html

```
{% load utils %}
{% load staticfiles %}
<h1 class="page-header"><span class="{{ action_class }}" {{
action_item }}">{{ title }}</span></h1>
{% for col, val in filtered_pk_column_all_values.items %}
  <label for="{{ col }}" class="group-label">
    {% for column in header %}
      {% if column == col %}
        <span class="head-attr">{{ column }}</span><br>
        <span class="head-col-descr gr-descr">{{
header_descrlget_index:forloop.counter0 }}</span>
      {% endif %}
    {% endfor %}
  <select class="{{ col }}" group-select" name="{{ col }}">
    <option></option>
    {% for value in val %}
      <option>{{ value }}</option>
    {% endfor %}
  </select>
</label>
{% endfor %}
<button type="button" class="btn btn-success group-
btn">Отобразить</button>
<table class="group-table table table-bordered">
  <thead>
    {% include 'head_table.html' %}
  </thead>
  <tbody>
    <tr>
      <td> Задайте значения в селекторах для группировки.</td>
    </tr>
  </tbody>
</table>
<script src="{% static 'js/group_view.js' %}"></script>
```

## Листинг исходного кода файла group\_view.js

```
$(document).ready(function() {
  $(".group-select").select2({
    placeholder: "Выберите значение",
    allowClear: true
```

```
});
$(".group-btn").on("click", function(){
  $(".group-table").toggle();
  $("tbody tr").remove();

  var array_ajax = {};
  $(".group-select").each(function(index, elem){
    if (!$elem.val()) {
      alert("Одно из значений не задано");
      array_ajax = {};
      array_ajax['error'] = 'error';
      $(".tbody").append('<tr><td>Одно из значений не задано!
Задайте все значения группировки!</td></tr>');
      $(".group-table").toggle();
      return false;
    }
    else{
      // alert($(".elem").val());
      array_ajax[$(elem).attr('name')] = $(".elem").val();
    }
  })
  // console.log(array_ajax);
  // console.log(typeof(array_ajax));
  if(array_ajax['error'] != 'error'){
    classList = $(".main>h1>span").attr('class').split(/\s+/);
    data = {
      'cmd': 0,
      'action': 4,
      'action_class': classList[0],
      'action_item': classList[1],
      'action_title': $(".main>h1>span").text(),
      'filter': array_ajax
    }
    // console.log(data);
    ajax_request(data, function (data) {
      // console.log(data);
      var arr_pk = [];
      $(".thead .head-pk").each(function(index, elem){
        arr_pk.push($(".elem").text());
      });
      var arr_head = [];
      $(".th span:first-child").each(function(index, elem){
        arr_head.push($(".elem").text());
      });
      // console.log(arr_pk);
      // console.log(arr_head);
      var resp = JSON.parse(data);
```

```

var rows = resp['filtered_rows_list'];
// console.log(rows);
if(rows.length != 0){
  for (var i in rows){
    var row = rows[i];
    $('tbody').append('<tr>');
    for (var row_i in row){
      $('tbody tr:last-child').append('<td></td>');
    }
    for (var row_i in row){
      var pos = arr_head.indexOf(row_i);
      $('tr:last-child td').eq(pos).text(row[row_i]);
      if(arr_pk.indexOf(row_i) != -1) {
        $('tr:last-child
td').eq(pos).addClass("background-pk");
      }
    }
    $('tbody').append('</tr>');
  }
  $('.group-table').toggle();
}
else {
  $('tbody').append('<tr><td>Ничего не найдено!
Попробуйте другие значения группировки!</td></tr>');
  $('.group-table').toggle();
}
})
}

});
});

```

## Листинг исходного кода файла group\_view.css

```

.group-select {
  width: auto;
  min-width: 200px;
  padding-right: 20px;
}

.group-btn {
  height: 28px;
  line-height: 100%;
}

.group-table {
  margin-top: 30px;
}

```

```

.selection {
  font-weight: 300 !important;
}

.head-pk,
.head-fk,
.head-attr,
.head-col-descr {
  font-family: 'Times New Roman';
}

.gr-descr {
  width: 200px !important;
  display: block;
}

```

## Листинг исходного кода функции модуля group\_view в файле handle\_table.py

```

elif view_mode == 'GROUPED_TAB_VIEW':
    table_request = {'CMD': 2,
                     'USER': request.user.username,
                     'PASSWORD': user_pass,
                     'TABLE_NAME': 'GCDDDB.sys.aiprv'
                     }

    aiprv = get_table(node, table_request)

    try:
        ai_params_row = aiprv.filter({'aisid': data['action_item'],
                                      'prmid': 'SGROUPED_DFL_HEADNUM'})[0]

        filtered_pk_columns_number =
            int(ai_params_row['aiprv!'])
    except:
        filtered_pk_columns_number =
            len(context['primary_key']) - 1

        filtered_pk_column_names =
            context['primary_key'][:filtered_pk_columns_number]
        filtered_pk_column_selected_values = None
        print(data)
        if 'filter' in data.keys():
            filtered_pk_column_selected_values = data['filter']
        else:
            filtered_pk_column_selected_values = {col_name:
            rows_list[0][context['header'].index(col_name)] for col_name in
            filtered_pk_column_names}

        filtered_rows_list =
            sql_table.filter(filtered_pk_column_selected_values)
        filtered_rows_list_excluding_pk = []
        for row_ in filtered_rows_list:
            filtered_rows_list_excluding_pk.append({col: row_[col]

```

```

for col in set(context['header']) - set(filtered_pk_column_names)):
    context['filtered_rows_list'] =
filtered_rows_list_excluding_pk

    filtered_pk_column_all_values = {col_name: set() for
col_name in filtered_pk_column_names}
    for row_ in rows_list:
        for col_ in filtered_pk_column_names:

filtered_pk_column_all_values[col_].add(row_[context['header'].inde
x(col_)])

```

```

    context['filtered_pk_column_all_values'] = {col_name :
list(values_set) for col_name, values_set in
filtered_pk_column_all_values.items()}
    else:
        context['rows_list'] = rows_list
        context['foreign_keys'] = sql_table.foreign_keys
        return context
    else:
        return {'error': 'Ошибка при получении %s' %
(table_name)}

```

## Листинг исходного кода файла

### graph\_visualizer.html

```
{% load utils %}
{% load staticfiles %}
<select class="solver" id="solver">
{% for solver in solvers %}
    <option></option>
    <option>{{ solver }}</option>
{%endfor%}
</select>
<button type="button" class="btn btn-success graph-
btn">Построить</button>
<div id="mynetwork"></div>
<script src="{% static 'js/graph_build.js' %}"></script>
```

## Листинг исходного кода файла

### graph\_build.js

```
var mass_color = ['#dead26', '#9753e6', '#48b12f', '#f46d91',
'#86dd21', '#5535db'];
var mass_colr = {};
var color_default = '#db2c2c';
var index_color = 0;
function color_label(key_parent){
    if(key_parent){
        return mass_colr[key_parent];
    }
    else {
        return color_default;
    }
}

function node_width(len){
    if (len < 15) {
        if (len < 6) {
            return len*45;
        }
        else{
            return len*27;
        }
    }
    else{
        return len*24.3;
    }
}
```

```
}

function build_one_label(key, node, nodes, gr_levels, key_parent){
    var max_width_label = node.length;
    var maxwidth = node_width(max_width_label);
    var svg = '<svg xmlns="http://www.w3.org/2000/svg"
height="156px" width="" + maxwidth + "px" >' +
    '<rect x="0" y="0" width="100%" height="100%" fill="#3e3836"
></rect>' +
    '<foreignObject x="0" y="0" width="100%" height="100%">' +
    '<div xmlns="http://www.w3.org/1999/xhtml" style="font-
size:34px; text-align:center; ">' +
    '<div style="color:white; padding: 20px; padding-top: 60px;
background-color:' + color_label(key_parent) + '; height:156px; ">' +
    node + '</div>' +
    '</div>' +
    '</foreignObject>' +
    '</svg>';
    var url = "data:image/svg+xml;charset=utf-8,"+
encodeURIComponent(svg);
    nodes.push({id: key, image: url, shape: 'image', level:
gr_levels[key]});
}

function build_all(resp, nodes, gr_pred_parent, gr_links_parent,
key_parent){
    console.log("-----");
    console.log(resp);
    //
    var gr_nodes = resp.states_description;
    // console.log("Узлы");
    // console.log(gr_nodes);
    var gr_links = resp.state_predicate_links;
    gr_links = $.extend(true, gr_links, gr_links_parent);
    // console.log("Связи");
    // console.log(gr_links);
    var gr_pred = resp.predicates_description;
    gr_pred = $.extend(true, gr_pred, gr_pred_parent);
    // console.log("Предикаты");
    // console.log(gr_pred);
    var gr_levels = resp.levels_description;
    // console.log(gr_levels);
    var gr_keys = Object.keys(gr_nodes);
    var array_index_pred = [];
    for (var i in gr_keys) {
        if (typeof(gr_nodes[gr_keys[i]]) == 'object') {
            mass_colr[gr_keys[i]] = mass_color[index_color];
            index_color++;
        }
    }
}
```

```

    }
  }
  console.log(index_color);
  for (var i in gr_keys) {
    if (typeof(gr_nodes[gr_keys[i]]) == ['object']) {
      build_all(gr_nodes[gr_keys[i]], nodes, gr_pred, gr_links,
gr_keys[i]);
    }
    else {
//      console.log(gr_keys[i]);
      if(gr_keys[i] in gr_links) {
        var many_pred = gr_links[gr_keys[i]];
        var one_pred = gr_pred[gr_links[gr_keys[i]]];
        if(Array.isArray(many_pred)) {
          array_index_pred.push(gr_keys[i]);
        }
        else {
//          console.log(one_pred);
          var max_width_predicate = one_pred.length;
          var max_width_label = gr_nodes[gr_keys[i]].length;
          if(max_width_label > max_width_predicate) {
            var maxwidth = node_width(max_width_label);
          }
          else {
            var maxwidth = node_width(max_width_predicate);
          }
          var svg = '<svg xmlns="http://www.w3.org/2000/svg"
height="156px" width="" + maxwidth + 'px' >' +
            '<rect x="0" y="0" width="100%" height="100%"
fill="#3e3836" ></rect>' +
            '<foreignObject x="0" y="0" width="100%"
height="100%">' +
            '<div xmlns="http://www.w3.org/1999/xhtml"
style="font-size:34px; text-align:center; ">' +
            '<div style="color:white; padding: 20px; background-
color:' + color_label(key_parent) + '; ">' + gr_nodes[gr_keys[i]]
+ '</div>' +
            '<div style="color:white; padding: 20px; background-
color:#624343; ">' + one_pred + '</div>' +
            '</div>' +
            '</foreignObject>' +
            '</svg>';
          var url = "data:image/svg+xml;charset=utf-8,"+
encodeURIComponent(svg);
          nodes.push({id: gr_keys[i], image: url, shape: 'image',
level: gr_levels[gr_keys[i]]});
        }
      }
    }
  }
  else{
    build_one_label(gr_keys[i], gr_nodes[gr_keys[i]], nodes,

```

```

gr_levels, key_parent );
  }
}
}
for(var i in array_index_pred) {
//  console.log(array_index_pred[i]);
  build_one_label(array_index_pred[i],
gr_nodes[array_index_pred[i]], nodes, gr_levels, key_parent );
  var predicates = gr_links[array_index_pred[i]]
  for(var pred_i in predicates) {
    var maxwidth =
node_width(gr_pred[predicates[pred_i]].length);
    var svg = '<svg xmlns="http://www.w3.org/2000/svg"
height="156px" width="" + maxwidth + 'px' >' +
      '<rect x="0" y="0" width="100%" height="100%"
fill="#3e3836" ></rect>' +
      '<foreignObject x="0" y="0" width="100%"
height="100%">' +
      '<div xmlns="http://www.w3.org/1999/xhtml" style="font-
size:34px; text-align:center; ">' +
      '<div style="color:white; padding: 20px; background-
color:#624343; height:156px; padding-top: 60px; ">' +
gr_pred[predicates[pred_i]] + '</div>' +
      '</div>' +
      '</foreignObject>' +
      '</svg>';
    var url = "data:image/svg+xml;charset=utf-8,"+
encodeURIComponent(svg);
    nodes.push({id: predicates[pred_i], image: url, shape:
'image', level: gr_levels[predicates[pred_i]]});
  }
}
}

```

```

$(document).ready(function() {
  $(".solver").select2({
    placeholder: "Выберите решатель",
    allowClear: true
  });
  $(".graph-btn").on("click", function(){
    var nodes = null;
    var edges = null;
    var network = null;
    mass_colr = {};
    index_color = 0;
    var value = $(".#solver").val();
    if (value) {
      nodes = [];
      edges = [];
      data = {
        'cmd': 0,

```

```

        'action': 1,
        'action_class': 'PYWEBS',
        'action_item': 'graph_visualizer',
        'solver': value
    }
    if(data){
        ajax_request(data, function (data) {
//            console.log(data);
            var resp = JSON.parse(data);
//            console.log(resp);
            if (!resp.error) {
                graph_descr = resp.graph_description;
                var lvl_sep = 100 + resp.max_edge_label_length * 6;
//                alert(lvl_sep);
                build_all(graph_descr, nodes, null, null, null);
                var DOTstring = resp.dot_string;
                var parsedData = vis.network.convertDot(DOTstring);
                var options = {
                    physics : false,
                    layout: {
                        randomSeed: undefined,
                        improvedLayout:true,
                        hierarchical: {
                            enabled:true,
                            levelSeparation: lvl_sep,
                            nodeSpacing: 360,
                            treeSpacing: 300,
                            blockShifting: true,
                            edgeMinimization: true,
                            parentCentralization: false,
                            direction: 'UD',    // UD, DU, LR, RL
                            sortMethod: 'directed' // hubs, directed
                        }
                    },
                    edges:{
                        color: '#d8c1c1',
                        font: {
                            align: 'top',
                            size: 10,
                        }
                    },
                },
            }
            var container =
document.getElementById('mynetwork');

            var data = {
                nodes: nodes,
                edges: parsedData.edges
            };

            network = new vis.Network(container, data, options);
        }
    }

```

```

        else {
//            alert(resp.error);
            $('#mynetwork').text(resp.error);
        }
    })
}

}

else {
    alert("Значение не выбрано");
}

}))
});

```

## Листинг исходного кода файла graph\_build.css

```

#solver {
    width: auto;
    padding-right: 20px;
}

.graph-btn {
    height: 28px;
    line-height: 100%;
}

#mynetwork {
    width: 1000px;
    height: 700px;
    border: 1px solid #eaeaea;
    margin-top: 20px;
    font: 16px 'Times New Roman';
    padding: 20px;
}

```

## Листинг исходного кода файла graph\_visualizer.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from django.shortcuts import render
from django.http import JsonResponse
from content.scripts.general import *

def handle_ajax_request(request, node, data):

```



```

if 'action' not in data.keys():
    return render(request, 'error.html', {'error': 'Неверный формат
данных. Неизвестное действие'})
user_pass = request.COOKIES.get('pass')
if not user_pass:
    return render(request, 'error.html', {'error': 'Ошибка при
авторизации.'})
# Действие 0 возвращает html-код для выбора графа для
отображения
if data['action'] is 0:
    # Запрос таблицы slvrs
    table_request = {'CMD': 2,
                     'USER': request.user.username,
                     'PASSWORD': user_pass,
                     'TABLE_NAME': 'GCDDDB.com.slvrs'
                    }
    sql_table = get_table(node, table_request)
    if type(sql_table) is not cls_SqlTable:
        return render(request, 'error.html', {'error': 'Ошибка доступа
к базе таблиц'})
    solvers = [row['slvr'] for row in sql_table.rows]
    context = {
        'action_item' : 'graph_visualizer',
        'action_class' : 'PYWEBS',
        'solvers' : solvers
    }

    return render(request, 'graph_visualizer.html', context)
elif data['action'] is 1: # Действие 1 строит модель графа для
отображения выбранного решателя
    if 'solver' not in data.keys():
        return render(request, 'error.html', {'error': 'Неверный
формат данных. Не задан решатель для отображения'})

    table_request = {'CMD': 2,
                     'USER': request.user.username,
                     'PASSWORD': user_pass,
                     'TABLE_NAME': 'GCDDDB.com.slvrs'
                    } # решатели
    slvrs = get_table(node, table_request)
    table_request['TABLE_NAME'] = 'GCDDDB.com.slnks' # ребра
графа (функции) (соединяют состояния)
    slnks = get_table(node, table_request)
    table_request['TABLE_NAME'] = 'GCDDDB.com.slprc' #
соответствия функция -> обработчик, предикат
    slprc = get_table(node, table_request)
    table_request['TABLE_NAME'] = 'GCDDDB.com.strep' #
состояния графа, которые на самом деле являются решателями
    strep = get_table(node, table_request)
    if type(slvrs) is not cls_SqlTable or type(slnks) is not
cls_SqlTable or

```

```

type(slprc) is not cls_SqlTable or type(strep) is not
cls_SqlTable:
    return render(request, 'error.html', {'error': 'Ошибка при
загрузке таблиц'})
    # Вычитываем сетевую модель, соотв. решателю
(com.slvrs.ntmdl by com.slvrs.solver)
    solver = data['solver']
    try:
        graph_description, dot_string, _, _, max_edge_label_length
= get_network_model_for_web(solver, slvrs, slnks, slprc, strep,
solver, 1)
    except BadGraphStructureException as err:
        return JsonResponse({'error': str(err)})

    response_data = {
        'graph_description' : graph_description,
        'dot_string' : 'digraph %s { %s }' % (solver, dot_string),
        'max_edge_label_length' : max_edge_label_length,
    }

    #print('states_description')
    #print(response_data['graph_description']['states_description'])
    #print('predicates_description')

    #print(response_data['graph_description']['predicates_description'])
    #print('state_predicate_links')

    #print(response_data['graph_description']['state_predicate_links'])
    #print('levels_description')
    #print(response_data['graph_description']['levels_description'])
    #print('dot_string')
    #print(response_data['dot_string'])
    #print(response_data)
    return JsonResponse(response_data)
else: # Если обработчик для действия неопределён
    return render(request, 'error.html', {'error': 'Неизвестная
ошибка'})

def get_network_model_for_web(solver, slvrs, slnks, slprc, strep,
states_prefix, ref_level):
    # 'UNIQUE_STATE_NAME' = solver + '_' + 'STATE_NAME'
    # 'UNIQUE_PREDICATE_NAME' =
'UNIQUE_INITIAL_STATE_NAME' + '_' + 'PREDICATE_NAME'
+ '_' + 'UNIQUE_TERMINATE_STATE_NAME'
    # states_description = {
    #     'UNIQUE_STATE_NAME' : 'SHOWED_NAME',
    #     'UNIQUE_STATE_NAME' : {'states_description': {},
'predicates_description': {}, 'state_predicate_links': {} }
    # }
    #
    # predicates_description = {

```

```

# 'UNIQUE_PREDICATE_NAME' : 'SHOWED_NAME',
# }
#
# state_predicate_links = {
# 'UNIQUE_STATE_NAME' :
'UNIQUE_PREDICATE_NAME',
# 'UNIQUE_STATE_NAME' :
['UNIQUE_PREDICATE_NAME_1',
'UNIQUE_PREDICATE_NAME_2'],
# }

# Вычитываем сетевую модель, соотв. решателю
(com.slvrsl.mdl by com.slvrsl.solver)
network_model = slvrsl.filter({'slvr': solver})[0]['ntmdl']
if network_model == "":
    raise BadGraphStructureException('Для решателя {} не задана
сетевая модель'.format(solver))

# Вычитываем ребра сетевой модели (com.slvrsl.slstc
(начальное состояние), com.slvrsl.slprc (ребро), com.slvrsl.slstn
(конечное состояние) by com.slvrsl.ntmdl)
edges_rows = slvrsl.filter({'ntmdl': network_model})
states_description = {}
predicates_description = {}
state_predicate_links = {}
graph_view = {}
starting_states = []
ended_states = []

def _make_unique_state_name(state_name):
    return '_'.join([states_prefix, state_name])

for edge_row in edges_rows:
    start_state = edge_row['slstc']
    end_state = edge_row['slstn']
    unique_start_state_name =
_make_unique_state_name(start_state)
    unique_end_state_name =
_make_unique_state_name(end_state)
    starting_states.append(start_state)
    ended_states.append(end_state)
    func = edge_row['slprc']
    if not unique_start_state_name in graph_view:
        graph_view[unique_start_state_name] = []
    else:
        is_the_same_states = False
        for start_state_data in graph_view[unique_start_state_name]:
            if start_state_data['end_state'] == unique_end_state_name:
                is_the_same_states = True
        if is_the_same_states:
            continue

```

```

graph_view[unique_start_state_name].append({
    'func' : func,
    'end_state' : unique_end_state_name
})

states_description[unique_start_state_name] = start_state
states_description[unique_end_state_name] = end_state

terminate_states = set(ended_states) - set(starting_states)
if len(terminate_states) != 1:
    raise BadGraphStructureException('Неправильное кол-во
выходных узлов графа: {}'.format(len(terminate_states)))
    terminate_state = terminate_states.pop() # если
len(terminate_states) != 1, то структура графа неправильная
    initial_states = set(starting_states) - set(ended_states)
    if len(initial_states) != 1:
        raise BadGraphStructureException('Неправильное кол-во
выходных узлов графа: {}'.format(len(initial_states)))
        initial_state = initial_states.pop() # если len(initial_states) != 1, то
структура графа неправильная

graph_main_states = {
    'unique_initial_state_name': _make_unique_state_name(initial_state),
    'unique_terminate_state_name':
_make_unique_state_name(terminate_state),
    'terminate_state_level': 0,
}

# Определяем, какие состояния должны быть заменены на
подграфы
substs = {}
substs_rows = slvrsl.filter({'slvr': solver, 'ntmdl':
network_model})
substs = {_make_unique_state_name(subst_row['slstt']) :
subst_row['slrep'] for subst_row in substs_rows}
substs_data = {}

def _fill_substituted_state_data(unique_state, current_level):
    sub_dot = "
    if not unique_state in substs_data:
        sub_descr, sub_dot, sub_init_state, sub_term_state,
sub_term_state_level, sub_max_edge_label_length = \
        get_network_model_for_web(substs[unique_state], slvrsl,
slvrsl, slprc, slrep, unique_state, current_level)
        substs_data[unique_state] = {
            'init_state' : sub_init_state,
            'term_state' : sub_term_state,
            'term_state_level' : sub_term_state_level,
            'max_edge_label_length' : sub_max_edge_label_length
        }

```

```

        states_description[unique_state] = sub_descr
    return sub_dot

# Строим дерево для определения уровней графа
levels_description = {}
processed_unique_states = []

def _recur_build(unique_parent_state_name, parent_level, dot,
max_edge_label_length):
    # parent_state -- На текущем
    def _add_level(node, level):
        # Обновляем уровень на БОльший только для
терминального узла
        if (not node in levels_description) \
            or (unique_parent_state_name ==
graph_main_states['unique_terminate_state_name'] and level >
levels_description[node]):
            levels_description[node] = level
        else:
            return levels_description[node]
        return level

    if (unique_parent_state_name in processed_unique_states) or
(not unique_parent_state_name in graph_view): # вторая опция
должна выполняться только для терминального узла
        real_level = _add_level(unique_parent_state_name,
parent_level)
        if unique_parent_state_name ==
graph_main_states['unique_terminate_state_name']:
            graph_main_states['terminate_state_level'] = real_level
            return dot, parent_level, max_edge_label_length
        processed_unique_states.append(unique_parent_state_name)
        parent_edge_data = graph_view[unique_parent_state_name]
        # unique_start_state_name -- имя для "крепления" узлов в
dot-строке, может прийти из рекурсивного узла.
        # unique_parent_state_name -- имя для узла на текущем
"слое" графа, не может прийти из рекурсивного узла
        unique_start_state_name = unique_parent_state_name
        unique_parent_state =
states_description[unique_parent_state_name]
        if unique_parent_state_name in substs:
            dot +=
_fill_substituted_state_data(unique_parent_state_name, parent_level)
            if unique_parent_state_name ==
graph_main_states['unique_initial_state_name']:
                graph_main_states['unique_initial_state_name'] =
substs_data[unique_parent_state_name]['init_state']
            if
substs_data[unique_parent_state_name]['max_edge_label_length'] >
max_edge_label_length:
                max_edge_label_length =

```

```

substs_data[unique_parent_state_name]['max_edge_label_length']
        parent_level =
substs_data[unique_parent_state_name]['term_state_level']
        unique_start_state_name =
substs_data[unique_parent_state_name]['term_state']
        _add_level(unique_parent_state_name, parent_level)
        highest_state_level = parent_level
        for edge in parent_edge_data:
            # unique_end_state_name -- имя для "крепления" узлов в
dot-строке, может прийти из рекурсивного узла.
            # unique_child_state_name -- имя для узла на текущем
"слое" графа, не может прийти из рекурсивного узла
            unique_end_state_name = edge['end_state']
            unique_child_state = edge['end_state']

            # Вычитываем данные ребра (com.sprcs.fnprf,
com.sprcs.prdcf by com.sprcs.solver, com.sprcs.ntmdl,
com.sprcs.ntprc)
            func_row = slprc.filter({'ntprc':edge['func']})[0]
            processing_func = func_row['fnprf']
            predicate = func_row['prdcf']
            unique_predicate_name = '_' + join([unique_start_state_name,
predicate, unique_end_state_name])
            predicates_description[unique_predicate_name] = predicate
            children_level = parent_level + 1
            if len(parent_edge_data) != 1:
                _add_level(unique_predicate_name, children_level)
                children_level += 1
            if not unique_start_state_name in state_predicate_links:
                state_predicate_links[unique_start_state_name] = []

state_predicate_links[unique_start_state_name].append(unique_predi
cate_name)
            dot += ' { } -- { } { } '.format(unique_start_state_name,
unique_predicate_name, unique_predicate_name)
        else:
            state_predicate_links[unique_start_state_name] =
unique_predicate_name
            dot += ' { } { } '.format(unique_start_state_name)
            sub_dot = "
            if unique_child_state in substs:
                sub_dot = _fill_substituted_state_data(unique_child_state,
children_level)
                if unique_child_state ==
graph_main_states['unique_terminate_state_name']:
                    graph_main_states['unique_terminate_state_name'] =
substs_data[unique_child_state]['term_state']
                    graph_main_states['terminate_state_level'] =
substs_data[unique_child_state]['term_state_level']
                if
substs_data[unique_child_state]['max_edge_label_length'] >

```

```

max_edge_label_length:
    max_edge_label_length =
substs_data[unique_child_state]['max_edge_label_length']
    unique_end_state_name =
substs_data[unique_child_state]['init_state']
    children_level =
substs_data[unique_child_state]['term_state_level']
    dot += ' -> {}[label="{}"] '.format(unique_end_state_name,
processing_func)
    if len(processing_func) > max_edge_label_length:
        max_edge_label_length = len(processing_func)
    dot += sub_dot
    dot, term_state_level, max_edge_label_length =
_recur_build(unique_child_state, children_level, dot,
max_edge_label_length)
    if term_state_level > highest_state_level:
        highest_state_level = term_state_level
    return dot, highest_state_level, max_edge_label_length

```

```

dot_string, terminate_state_level, max_edge_label_length =
_recur_build(graph_main_states['unique_initial_state_name'],
ref_level, "", 0)

description = {
    'states_description' : states_description,
    'predicates_description' : predicates_description,
    'state_predicate_links' : state_predicate_links,
    'levels_description' : levels_description
}

return description, dot_string,
graph_main_states['unique_initial_state_name'],
graph_main_states['unique_terminate_state_name'],
graph_main_states['terminate_state_level'], max_edge_label_length

class BadGraphStructureException(Exception):
    pass

```

## Листинг исходного кода файла

### head\_table.html

```
{% load utils %}
<tr>
  {% if not tree_model %}
    <th class="head-number">
      #
    </th>
  {% endif %}
  {% for column in header %}
    {% if forloop.counter0 in PK_indices %}
      <th class="background-pk">
        <span class="head-pk">{{ column }}</span>
        <span class="head-col-descr">{{
header_descr|get_index:forloop.counter0 }}</span>
      </th>
    {% endif %}
  {% endfor %}
  {% for column in header %}
    {% if forloop.counter0 not in PK_indices %}
      {% if column in fk_columns_list %}
        {% if not tree_model %}
          <th>
            <span class="head-fk">{{ column }}</span>
            <span class="head-col-descr">{{
header_descr|get_index:forloop.counter0 }}</span>
          </th>
        {% endif %}
      {% else %}
        <th>
          <span class="head-attr">{{ column }}</span>
          <span class="head-col-descr">{{
header_descr|get_index:forloop.counter0 }}</span>
        </th>
      {% endif %}
    {% endif %}
  {% endfor %}
</tr>
```

## Листинг исходного кода

### оптимизированного файла

### table.css

```
/* обработка таблицы */
```

```
.background-pk {
  background-color: #f3f3f3;
}
.table {
  font-family: Times New Roman;
}
th span {
  font-weight: 600;
  display: block;
  margin-bottom: 2px;
}
th {
  vertical-align: top !important;
}
.head-number {
  vertical-align: middle !important;
}
.head-pk,
.head-fk,
.head-attr{
  color: #f3924a;
}
.table>tbody>tr:hover {
  background-color:#f5f5f5;
}
.table>tbody>tr>td>input {
  padding: 0;
  margin: 0;
  height: auto;
  border: none;
  display: table-cell;
}
.btn-default {
  border-color: transparent;
  border-radius: 0;
}
.d-inline-block {
  display: inline-block;
}
/* обработка таблицы */
```

## Листинг исходного кода оптимизированного файла table.js

```
function onRowDeleteSuccess(row, element) {
    row.fadeOut(150, function () {
        var newRowExists = $(".new-row").length;
        this.remove();
        if (newRowExists) {
            insertNewRowButton();
        } else {
            var itemSet = $(".table>tbody>tr>th")
            var itemSetLen = itemSet.length
            itemSet.each(function (index, element) {
                element.innerHTML = index + 1;
                if (index == itemSetLen - 1) {
                    $(".table-info>span").text(element.innerHTML);
                }
            });
        }
        element.remove();
    })
}

// Проверка первичного ключа и создание json объекта из строки
// таблицы (action: 1 - добавить, 2 - изменить, 3 - удалить)
function getJsonFromRow(row, action) {
    var data = {
        "cmd": 0,
        "action": 1,
        "action_item": "",
        "action_class": "",
        "row": 0,
        "primary_key": {},
        "changed_data": {}
    }
    data["action"] = action;
    classList = $(".main>h1>span").attr('class').split(/\s+/);
    if (classList.length != 2) {
        alert('Ошибка в запросе на изменение таблицы');
        return 0;
    }
    data["action_class"] = classList[0];
    data["action_item"] = classList[1];
    data["table_name"] = $(".#table-name>span").text();
    var header = $(".#main-table:first>thead>tr:first").children();
    var error = false;
    row.children().each(function (index) {
        // alert(this.innerHTML);
        var headOfColumn = header.eq(index);
        // alert((headOfColumn).children().attr('class'));
```

```
        var cellValue = this.innerHTML;
        var columnName = headOfColumn.text();
        columnName = columnName.replace(/\s+/, ""); //удаляем
        пробелы слева
        columnName = columnName.split(" ", 1); // отбираем только
        значение без описания
        // alert(columnName);
        if (headOfColumn.find(':first-child').attr('class') == 'head-pk') {
            if (!cellValue && action != 2) {
                alert("Не все значения первичного ключа (*PK)
                заполнены");
                if (!$(".new-row").length) {
                    row.removeClass();
                }
                row.addClass("danger");
                $(this).text($(this).data("old"));
                error = true;
            } else {
                if ($(this).hasClass("changed") || row.hasClass("new-
                row")) {
                    data.changed_data[columnName] = cellValue;
                    if ($(this).data("old") && !row.hasClass("new-row")) {
                        cellValue = $(this).data("old");
                    }
                }
                data.primary_key[columnName] = cellValue;
            }
        } else {
            if ($(this).attr("scope") == "row") {
                data.row = cellValue;
            } else {
                if ($(this).hasClass("changed") || row.hasClass("new-row")
                || this.id == 'changing_element') {
                    data.changed_data[columnName] = cellValue;
                }
            }
        }
    });
    if (error) {
        return 0;
    } else {
        return data;
    }
}
```

```
// Вызов функции (main), когда страница загружена и готова
$(document).ready(function () {
    // Обработка нажатия на строку таблицы
    $(document).on("click", "#main-table>tbody>tr", function () {
        if (!$(".new-row").length) {
            $(".table>tbody>tr").removeClass("info danger success");
```

```

        $(this).addClass("info");
        insertDeleteButton();
    } else {
        $(this).removeClass("danger");
    }
});
$(document).on("hidden.bs.modal", "#fk-modal-table", function(){
    $("#changing_element").removeAttr("id");
});
var key_column = null; // переменная, которая хранит значение
ключа при вызове модальной таблицы
// Обработка двойного нажатия на ячейку таблицы
$(document).on("dblclick", "#main-table>tbody>tr>td, #modal-
table>tbody>tr>td", function () {
    var tableCell = $(this);
    cellText = this.innerHTML;

    //Если открыто модальное окно
    if ($('#fk-modal-table').is(':visible')) {
        var index_column;
        $('#fk-modal-table thead th').each(function(index,elem){
            if ($(elem).children().hasClass("head-pk")){
                if ($(elem).children(".head-pk").text().trim() ==
key_column.trim()){
                    index_column = index; //узнаем индекс колонки с
ключами
                }
            }
        });
        // console.log(index_column);
        cellText =
$(this).parent("tr").children().eq(index_column).text();
        cellText = cellText.trim();
        cellText = cellText.replace(/<[^>]+>/g, ""); // очищение
символов для древовидного представления
        cellText = cellText.replace(/&nbsp;/g, " ");
        var cell_to_change =
document.getElementById('changing_element');
        cell_to_change.innerHTML = cellText;
        $("#fk-modal-table").modal('toggle');
        row = $(cell_to_change).closest("tr");
        changed_data = getJsonFromRow(row, 2);
        cell_to_change.id = 'changed';

        //Отправка полученных данных
        if (changed_data) {
            ajax_request(changed_data, function (changed_data) {
                var resp = JSON.parse(changed_data);
                if (resp['ERROR_CODE'] != 0) {
                    row.find(".changed").each(function () {
                        this.innerHTML = $(this).data("old");
                    });
                    onRowChangeSuccess(row);
                    row.addClass("danger");
                } else {
                    onRowChangeSuccess(row);
                    row.addClass("success");
                    setTimeout(function() { row.removeClass("success");
}, 2000);
                }
            });
        }
    }
}

```

```

    });
    onRowChangeSuccess(row);
    row.addClass("danger");
} else {
    onRowChangeSuccess(row);
    row.addClass("success");
    setTimeout(function() { row.removeClass("success");
}, 2000);
}
});
}
else {
    if (tableCell.closest("tr").hasClass("info")) {

        //Проверка на наличие внешних ключей или их
получение
        this.id = 'changing_element'
        var row = tableCell.closest("tr");
        var pos = row.context['cellIndex'];
        //alert(pos);
        var widthOfCell = $(this).width();
        //alert(widthOfCell);
        a = $(this);

        data = getJsonFromRow(row, 4);
        data['action_title'] = 'take_foreign_keys';

        for (i in data['changed_data']){
            key_column = i;
        }

        //Определяем, каким образом менять значение ячейки,
для этого запрашиваем внешние ключи
        change_table_data = ajax_request(data, function (data) {
            var resp = JSON.parse(data);
            //Проверяем на ошибки
            if (resp['ERROR_CODE'] != 0) {
                row.find(".changed").each(function () {
                    this.innerHTML = $(this).data("old");
                });
                onRowChangeSuccess(row);
                row.addClass("danger");
            } else {
                onRowChangeSuccess(row);
                row.addClass("success");
                setTimeout(function() { row.removeClass("success");
}, 2000);
            }
        });
    }
}

```

```

//Ищем столбец с изменяемой ячейкой во внешних
ключях
if (resp["foreign_keys"] != null) {
    table_to_take = null;
    for (foring_table in resp["foreign_keys"]) {
        var num = resp["foreign_keys"][foring_table]
        if ((pos - 1) in num) {
            table_to_take = foring_table;
            break;
        } else {
            continue;
        }
    }
}

//Если наш столбец есть во внешних ключах,
получаем его
if (table_to_take != null) {
    data = {
        'cmd': 0,
        'action': 0, // 0 - make html
        'action_class': 'SBPLDB',
        'action_item': table_to_take,
        'action_title': 'get_secondary_table'
    }
    secondary_table = ajax_request(data, function (data) {

        //Вырезаем таблицу из результата запроса
        begin = data.indexOf('<table');
        if (begin == -1) {
            table = data;
            table = table.replace('get_secondary_table',
table_to_take)
        } else {
            end = data.lastIndexOf('</table>') +
'</table>'.length;
            table = data.substring(begin, end);
        }
        //Подставляем полученную таблицу в модальное
окно
        $('#fk-modal-table .modal-body').html(table);
        table = $('#fk-modal-table .modal-
body')[0].firstChild;
        table.id = 'modal-table';
        //table.setAttribute('modal-table')

        //открыть модальное окно
        $('#fk-modal-table').modal('toggle');
        tree_view_modal();
    });
} else {
    //Если столбца с ячейкой нет во внешних ключах,
    вводим вручную
    tableCell.html('<input type="text" class="form-
control" value="" + cellText + ">');
    tableCell.width(widthOfCell);

    var tableCellInput = $(tableCell['context'].firstChild);
    console.log(tableCellInput);
    tableCell.data("old", cellText);
    tableCellInput.focus().select();

    tableCellInput.change(function () {
        row = tableCell.closest("tr");
        row.addClass("warning");
        $(this).parent().addClass("changed");
        $(this).replaceWith(this.value);
        console.log(this.value);
        if (!$(".new-row").length) {
            data = getJsonFromRow(row, 2);
            console.log(data);
            if (data) {
                ajax_request(data, function (data) {
                    console.log(data);
                    var resp = JSON.parse(data);
                    console.log(resp);
                    if (resp['ERROR_CODE'] != 0) {
                        row.find(".changed").each(function () {
                            this.innerHTML = $(this).data("old");
                        });
                        onChangeSuccess(row);
                        row.addClass("danger");
                    } else {
                        onChangeSuccess(row);
                        row.addClass("success");
                        setTimeout(function() {
                            row.removeClass("success"); }, 2000);
                            tableCell.removeAttr("id class style");
                        }
                    );
                }
            );
            $(tableCell['context'].firstChild).blur(function () {
                tableCell.html( cellText );
                tableCell.removeAttr("id class style");
            });
        }
    });
}

```



```

$(tableCell['context'].firstChild).keyup(function (e) {
    if (e.keyCode == 27) {

$(tableCell['context']).replaceWith($(tableCell['context']).parent().dat
a("old"));

    }
    });
    }
    });
    }
    })
})

// Обработка нажатия на кнопку "Добавления Строки"
$(document).on("click", "#new-row-btn", function () {
    this.blur();
    var firstRow = $("#main-table>tbody>tr:first");
    var newRow = firstRow.clone();
    var rowNum =
parseInt($(".table>tbody>tr:last").find("th:first").text()) + 1;
    newRow.find("th:first").html(rowNum);
    firstRow.parent().find(".info").removeClass("info");
    newRow.removeClass().addClass("new-row info warning");
    newRow.find("td").empty();
    firstRow.before(newRow);
    insertDeleteButton();
    insertSaveRowButton();

//
    $(".html, body").animate({ scrollTop:
$(".table>tbody>tr:last").offset().top }, "fast");
});
// Обработка нажатия на кнопку "Удаления Строки"
$(document).on("click", "#delete-btn", function () {
    this.blur();
    var row = $(".table>tbody>tr.info:first");
    data = getJsonFromRow(row, 3);
    if (!$(".new-row").length && data) {
//
        alert(JSON.stringify(data));

        row.addClass("warning");
        var element = this;
        ajax_request(data, function (data) {
            var resp = JSON.parse(data);
            if (resp['ERROR_CODE'] != 0) {
//
                alert(JSON.stringify(resp));
            } else {
//
                alert(JSON.stringify(resp));
                onRowDeleteSuccess(row, element);
            }
        });
    } else {
        onRowDeleteSuccess(row, this);
    }
});
});

```

### Текст доклада

#### Слайд 1

Темой моей дипломной работы является разработка веб-ориентированных динамических пользовательских интерфейсов для распределенных систем инженерного анализа.

#### Слайд 2

Цель этой работы формулируется следующим образом: создать веб-ориентированную графическую подсистему формирования графических пользовательских интерфейсов для использования в рамках CAE систем с организованным удаленным доступом для прикладных вычислительных задач микромеханики композиционных материалов.

#### Слайд 3

В соответствии этой цели были поставлены основные глобальные задачи: разработать веб-ориентированные программные модули в рамках существующей системы обеспечивающие автоматизацию построения...

- GUI (пользовательский графический интерфейс) на основе данных в БД.
- графических представлений вычислительных инструментов ("решателей"), применяемых для решения задач, в рамках технологии GBSE (Графо-ориентированная технология разработки программных реализаций сложных вычислительных методов).

Глобальными эти задачи называются потому, что каждая из них разбивается на несколько подзадач, о которых чуть позже конкретнее пойдет речь.

#### Слайд 4

Но прежде чем перейти к основным модулям, следует сказать несколько

слов о той системе, в которой вся работа была реализована. Это «Распределенная вычислительная система GCD», которая располагается в МГТУ им. Баумана и существует уже около 20 лет. Основная деятельность связана с композиционными материалами и направления представлены на этом слайде.

Как вы можете видеть, это и расчеты, и проведение инженерного анализа, и т.д.

Если переходить конкретнее к моей работе, то она связана с веб-клиентом этой системы, который представлен на следующем слайде.

## **Слайд 5**

Это внешняя структура веб-клиента, который состоит из набора компонентов: это стили и скрипты js и css, которые применяются к html шаблонам, используется СУБД PostgreSQL, за серверную часть отвечает язык Python с фреймворком Django. И все это осуществляется через систему контроля версиями GIT.

После того как мы познакомились с веб-клиентом системы перейдем к практической разработке.

## **Слайд 6**

Первая задача - построение GUI на основе данных в БД. Задача нацелена на то, чтобы повысить общий уровень интерфейса веб-клиента, а также сделать использование данных гораздо удобнее, за счет создания новых представлений.

Она разбивается на две подзадачи: древовидное и группированное представление данных.

## **Слайд 7**

Разберем древовидное представление. На этом слайде вы видите пример отображения данных, оно возможно в том случае, когда в одной таблице есть строки, ссылающиеся на строки в этой же таблице. В итоге образуются родительские и дочерние элементы. С помощью элемента управления можно сворачивать и разворачивать содержимое родительских элементов.

## Слайд 8

Вторая подзадача связана с группированным представлением, пример которого представлен на слайде. По названию понятно, что здесь происходит группировка данных по одному или нескольким общим полям. Это является удобным в случае, когда в таблице содержится много данных и необходимо решить проблему избыточности информации.

## Слайд 9

Веб-клиент представляет собой взаимодействие двух основных частей: визуальную и серверную, в данном клиенте их взаимодействие осуществляется через ајах-протокол. И для задачи построения GUI на визуальной части было разработано 3 шаблона html и 2 файла стилей и скриптов, которые применяются к шаблонам. На серверной части был доработан существующий плагин python двумя новыми функциями, основная цель которых в формировании и отправке данных в соответствующие шаблоны.

## Слайд 10

Теперь перейдем ко второй задаче – технология GBSE. Прежде чем приступить к реализации этой задачи, ее следовало изучить и понять.

**Основная идея** заключается в представлении сложных вычислительных методов в виде ориентированного графа, при интерпретации которого реализуется алгоритм решения данной конкретной задачи. Также граф часто называют графовой моделью.

В этой системе вычислительные методы, это численные методы механики сплошной среды: методы конечных элементов, методы гомогенизации и другие.

## Слайд 11

Как сказано ранее, алгоритм решения метода представляется в виде ориентированного графа или графовой модели. И для этого представления были разработаны основные компоненты модели, которые вы видите на слайде: узел с

одним условием, узел с несколькими условиями, узел без условий и функция перехода. Узел характеризует состояние данных, которые существуют в данной модели, а функция перехода – является преобразованием этих данных.

Для примера представим простую задачу: на входе имеются данные, например, плотность или температура. Эти условия проверяют не вышло ли численное значение за допустимые пределы, и в зависимости от результата выполняется соответствующая функция перехода.

## **Слайд 12**

Теперь рассмотрим реальный пример модели, которая построилась для пользователя в браузере. Слева вы можете увидеть увеличенную часть, чтобы понять взаимосвязь элементов.

Обращаю ваше внимание, что моя разработка заключается правильном визуальном представлении графовой модели, и конкретные наименования не имеют большого значения для реализации.

## **Слайд 13**

На визуальной части был разработан шаблон, который отображает модели. Самое важное происходит в js файле, который обрабатывает Json формат данных, а также применяет настройки для модели с помощью библиотеки vis.js.

На серверной части был разработан целый плагин, который формирует справочник с описанием узлов, ребер и условий между ними, также вычисляет уровни для каждого узла и формирует dot строку с описанием графа.

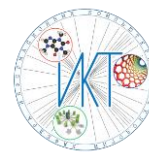
## **Слайд 14**

- ✓ Разработаны модули для построения интерфейса и отображения данных в древовидном и группированном представлении;
- ✓ Разработаны модули, обеспечивающие автоматизацию построения графических представлений вычислительных инструментов;
- ✓ Оптимизирована структура файлов стилей и скриптов веб-клиента с помощью обобщения кода и написания новых функций;

## Иллюстрационные материалы к докладу



Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
Российский химико-технологический университет им. Д.И.Менделеева  
Факультет информационных технологий и управления  
Кафедра информационных компьютерных технологий



### Выпускная квалификационная работа бакалавра на тему:

«Разработка веб-ориентированных динамических пользовательских интерфейсов для распределенных систем инженерного анализа»

**Студент:** Кириллов Никита Дмитриевич

**Руководители:** профессор, д.т.н., Кольцова Элеонора Моисеевна  
доцент, к.ф.-м.н., Соколов Александр Павлович

2017

## Цель работы

Создать веб-ориентированную графическую подсистему формирования графических пользовательских интерфейсов для использования в рамках CAE систем с организованным удаленным доступом для прикладных вычислительных задач микромеханики композиционных материалов.

\*CAE (*Computer-aided engineering*) — общее название для программ и программных пакетов, предназначенных для решения различных инженерных задач: расчётов, анализа и симуляции физических процессов.

Слайд 2

# Задачи

---

Разработать веб-ориентированные программные модули в рамках существующей системы обеспечивающие автоматизацию построения...

- GUI (пользовательский графический интерфейс) на основе данных в БД;
- графических представлений вычислительных инструментов ("решателей"), применяемых для решения задач, в рамках технологии GBSE (графо-ориентированная технология разработки программных реализаций сложных вычислительных методов);

Слайд 3

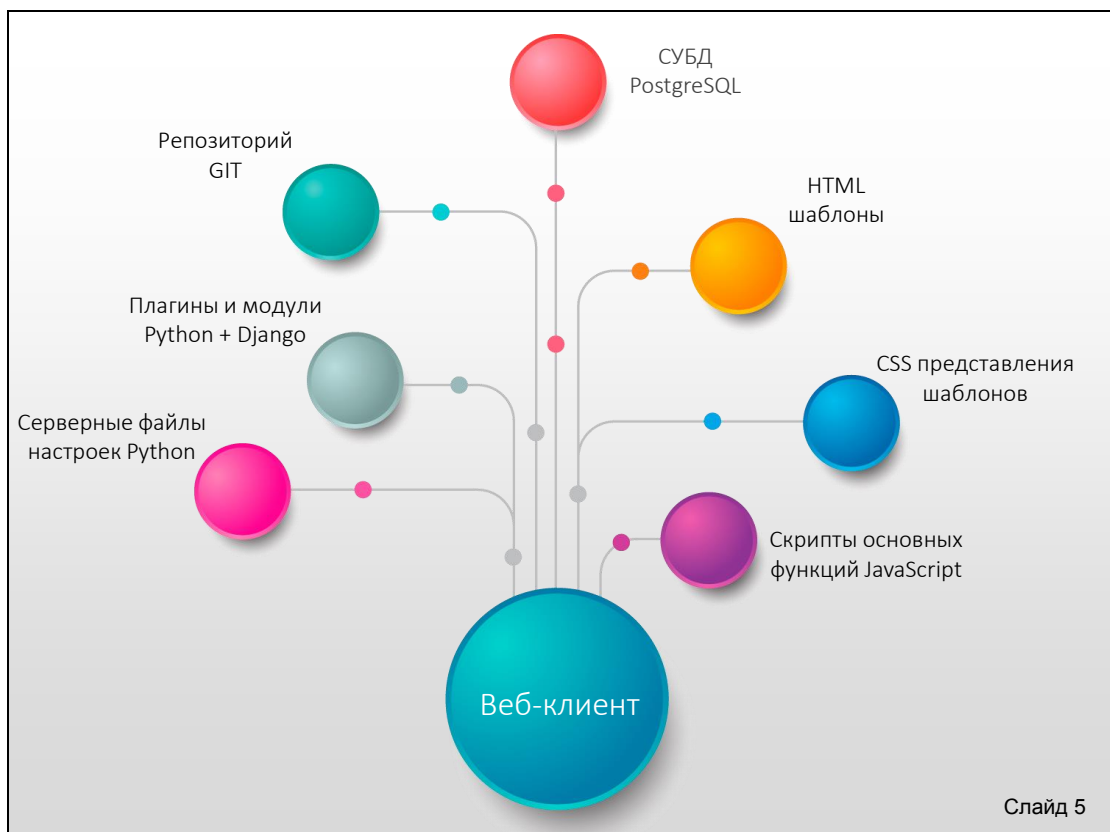
# Распределенная вычислительная система GCD

---

Работы проводятся для различных отраслей промышленности: авиастроение, ракетостроение, судостроение, медицина, машиностроение, ВПК, строительство, по следующим основным направлениям:

- Проведение расчетов характеристик композиционных материалов (КМ), в том числе решение задач проектирования КМ с заранее заданными свойствами.
- Инженерный анализ конструкций (задачи о напряженно - деформированном состоянии, теплопроводность, термоупругость и пр.).
- Решение задач многомерной многокритериальной оптимизации.

Слайд 4



## Построение GUI на основе данных в БД

Задача нацелена на то, чтобы повысить общий уровень интерфейса веб-клиента, а также сделать использование данных гораздо удобнее, за счет создания новых представлений.

Разделяется на 2 подзадачи:

- Древовидное представление данных
- Группированное представление данных

Слайд 6



## Древовидное представление

<b>atype</b> Имя предметного типа	<b>atdes</b> Описание	<b>dscrb</b> Подробное описание
⊕ REPORTS	ОТЧЕТЫ	Отчеты. Модули формирования отчетов.
⊖ RNDSUBSYS	Исследования и разработки	
PATRND	НИРы, ОКРы, ОТРы	
⊖ PATENTS	Патентные исследования	
IPCLASSIF	МПК классификация	

Слайд 7

## Группированное представление

**pbsid**

Публикация

dimitrienko7



Отобразить

<b>field</b> Атрибут	<b>pfval</b> Значение атрибута
author	Димитриенко Ю.И., Кашкаров А.И., Макашов А.А.
journal	Современные естественно-научные и гуманитарные проблемы: Сборник трудов конференции 40 лет факультета ФН
pages	485-498
title	Конечно-элементное моделирование процесса разрушения пространственно-армированных композитов с периодической структурой

Слайд 8

## Разработанные модули



Слайд 9

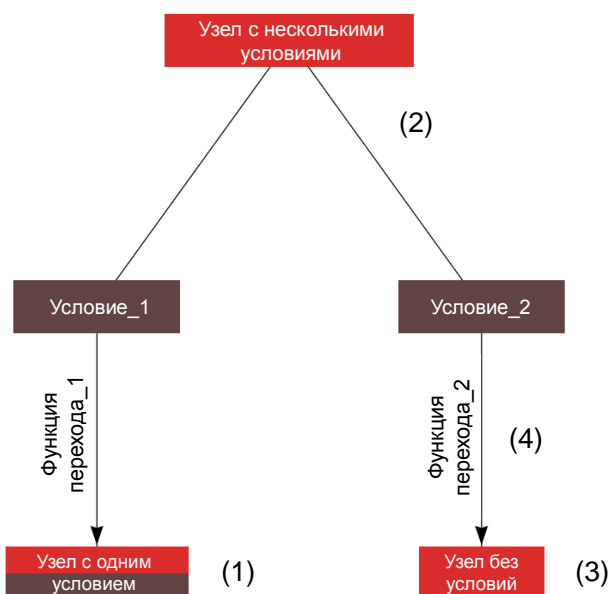
## GBSE технология

**Область применения подхода** – программные реализации численных методов механики сплошной среды: методы конечных элементов, методы гомогенизации и другие.

**Основная идея** заключается в представлении сложных вычислительных методов в виде ориентированного графа, при интерпретации которого реализуется алгоритм решения данной конкретной задачи.

Слайд 10

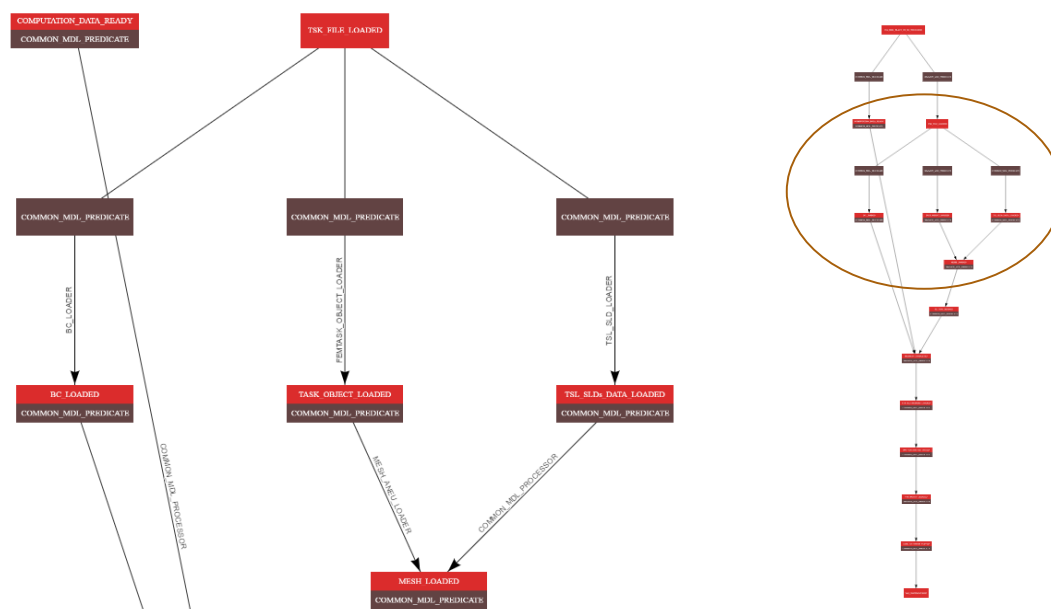
# Компоненты графовой модели



1. Узел с одним условием
2. Узел с несколькими условиями (в данном случае - два условия)
3. Узел без условий (обычно является конечным)
4. Стрелка - функция перехода (к следующему узлу)

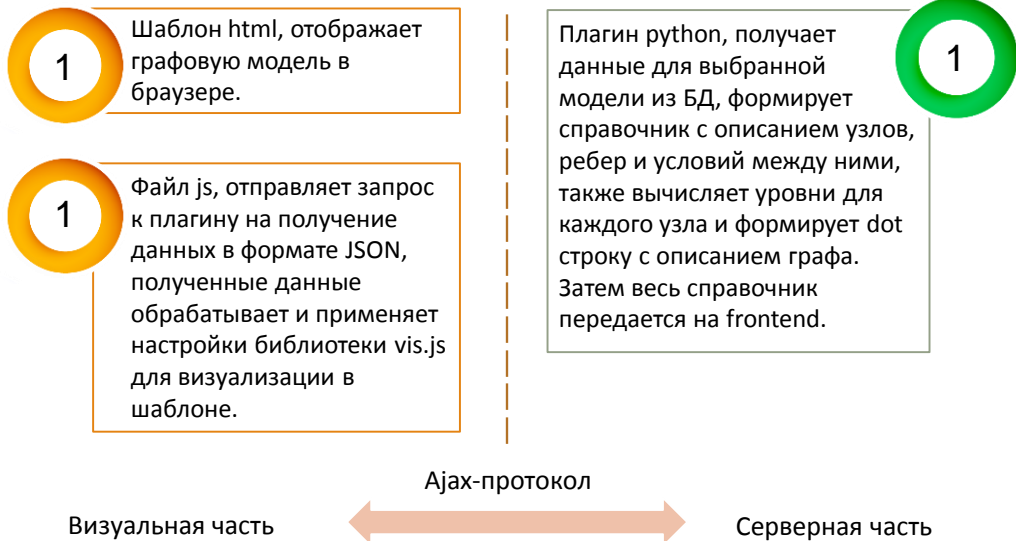
Слайд 11

# Пример построения модели



Слайд 12

## Разработанные модули



Слайд 13

## Выводы

- ✓ Разработаны модули для построения интерфейса и отображения данных в древовидном и группированном представлении;
- ✓ Разработаны модули, обеспечивающие автоматизацию построения графических представлений вычислительных инструментов;
- ✓ Оптимизирована структура файлов стилей и скриптов веб-клиента с помощью обобщения кода и написания новых функций;

Слайд 14