Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Системы автоматизированного проектирования (РК-6)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Разработка web-ориентированного программного обеспечения, реализующего автоматизированное построение динамических графических пользовательских интерфейсов»

Студент ___РК6-83Б____ _____ _____А.Р. Василян___
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР _____ _____А.П. Соколов___
(Подпись, дата) (И.О.Фамилия)

Нормоконтролер _____ _____
(Подпись, дата) (И.О.Фамилия)

2023 г.

РЕФЕРАТ

выпускная квалификационная работа: 60 с., 24 рис., 34 листингов, 14 слайдов, 10 источников.

ГРАФИЧЕСКИЙ ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, GUI, АВТОМАТИЗИРОВАННОЕ ПОСТРОЕНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА, ГЕНЕРАЦИЯ ИНТЕРФЕЙСА, DJANGO, DOCKER, PYPARSING, HTML, CSS.

Работа посвящена изучению методов построения графического пользовательского интерфейса и разработке web-ориентированного программного обеспечения, реализующего автоматизированное построение динамических графических пользовательских интерфейсов.

Тип работы: выпускная квалификационная работа.

Тема работы: «Разработка web-ориентированного программного обеспечения, реализующего автоматизированное построение динамических графических пользовательских интерфейсов».

Объект исследования: Графический пользовательский интерфейс.

Основная задача, на решение которой направлена работа: Разработка web-ориентированного программного обеспечения, реализующего автоматизированное построение динамических графических пользовательских интерфейсов.

Цели работы: рассмотреть существующие подходы к разработке графического пользовательского интерфейса, разработать библиотеку, обеспечивающую автоматизацию построения динамических пользовательских интерфейсов.

В результате выполнения работы: 1) были рассмотрены существующие подходы разработки GUI; 2) была разработана программа для преобразования данных формата aINI в HTML-код; 3) в рамках Django и Docker было разработано web-приложение. 4) Используя сгенерированный интерфейс в web-приложении, приложение было запущено на тестовом сервере sandbox.rk6.bmstu.ru и проверено корректность работы всего программного решения.

СОКРАЩЕНИЯ

GUI – Graphical User Interface

DSL – Domain-Specific Language

aINI – Advanced INI

URL – Uniform Resource Locator

CSS – Cascading Style Sheets

JS – JavaScript

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Интерфейс — это совокупность средств, методов и правил взаимодействия, управления, контроля и т.д. между элементами системы [1].

Пользовательский интерфейс — это разновидность интерфейсов, в котором одна сторона представлена человеком-пользователем, другая — машиной-устройством [1].

Графический пользовательский интерфейс (Graphical User Interface, GUI) — это разновидность пользовательского интерфейса, в котором элементы интерфейса, представленные пользователю на дисплее, исполнены в виде графических изображений [1].

GUI имеет важное значение во взаимодействии человека с программным обеспечением. Именно от него зависит удобство при эксплуатации какого-либо программного продукта.

Автоматизированные решения прикладных задач часто представляют собой программное обеспечение, включающее в свой состав графический польз интерфейс. Для постановки некоторой актуальной прикладной задачи понадобился интерфейс (форма ввода) для получения входных данных. Разработка такой формы ввода может быть не такой трудоемкой и занять мало времени, если требуется немного таких форм ввода и, если имеется разработчик, обладающий соответствующими навыками программирования. Но если требуется существенное количество таких форм ввода, которые в свою очередь имеют переключаемые вкладки, то время на разработку интерфейса будет значительным, тем более если нет требуемого специалиста. В таких случаях разумным решением будет разработка инструмента для генерации таких форм ввода или интерфейса на основе простого описания элементов каждой формы.

ПОСТАНОВКА ЗАДАЧИ

На рис. 1 представлена схема базового принципа генерации GUI на основе данных, определяемых на предметно-ориентированных языках (Domain-Specific Language, DSL). С помощью представленных на рисунке DSL (слева на схеме) можно описать элементы интерфейса для дальнейшей генерации на их основе.

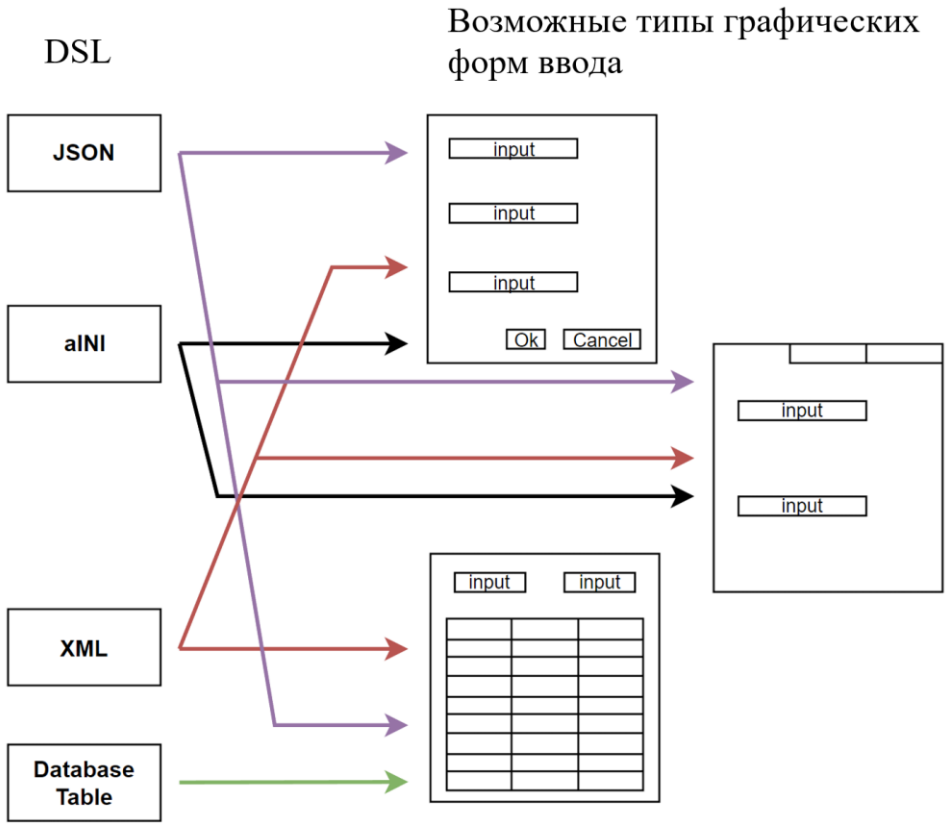

Рис. 1. Схема базового принципа генерации GUI на основе DSL

Одним из известных языков для описания входных данных является aINI (формат для описания входных данных различных задач, основанный на языке INI) [2]. Он обладает сравнительно простым синтаксисом и может быть использован в качестве DSL для генерации GUI.

В следующей работе требуется разработать программу, которая принимает на вход данные в формате aINI, преобразует полученные данные и выводит в HTML-файл, который далее будет использован для разметки web-приложения. Программа в том числе должна генерировать дополнительный код для получения и сохранения в файл данных, введённых пользователем.

Перед разработкой будут рассмотрены некоторые подходы к созданию интерфейса, для оценки актуальности работы. Были рассмотрены такие методы как метод построения оконного интерфейса пользователя на основе моделирования пользовательских целей, методический подход к созданию средства построения пользовательского интерфейса, построение пользовательского интерфейса с использованием интерактивного машинного обучения.

1 Подходы к разработке пользовательского интерфейса

1.1 Метод построения оконного интерфейса пользователя на основе моделирования пользовательских целей

При изучении статьи [3] были выделены два метода взаимодействия пользователя и ЭВМ: ограничительный и направляющий.

Ограничительный метод

Для средств взаимодействия пользователя и ЭВМ с оконным интерфейсом типичным является ограничительный метод. Метод заключается в том, что пользователю предоставляется некий набор операций, благодаря которым он может выполнять определенные действия с описанными в ЭВМ объектами своей деятельности. Операция имеет название, входные данные и результаты, представляющие собой объекты воздействия операции. Пользователь решает, какую из операций необходимо выбрать в данный момент для выполнения своего задания, выбирает нужную операцию и задает для нее входные данные. После чего ЭВМ выполняет указанную операцию, активируя соответствующие функции приложения, и выдаёт результаты операции пользователю.

По итогам выполнения очередной операции пользователь решает, какую следующую операцию ему нужно выбрать, передаёт ее ЭВМ на выполнение и т.д. Этот процесс он должен продолжать до тех пор, пока в итоге выполнения операций не будет достигнут желаемый результат, соответствующий выполненному заданию. Следовательно, пользователь должен сам планировать ход выполнения своего задания из предоставляемых ему операций. Обобщенная схема ограничительного метода взаимодействия приведена на рис. 2. Для ограничительного метода считается, что у пользователя присутствуют процедурные знания, необходимые для планирования процесса выполнения своих заданий.

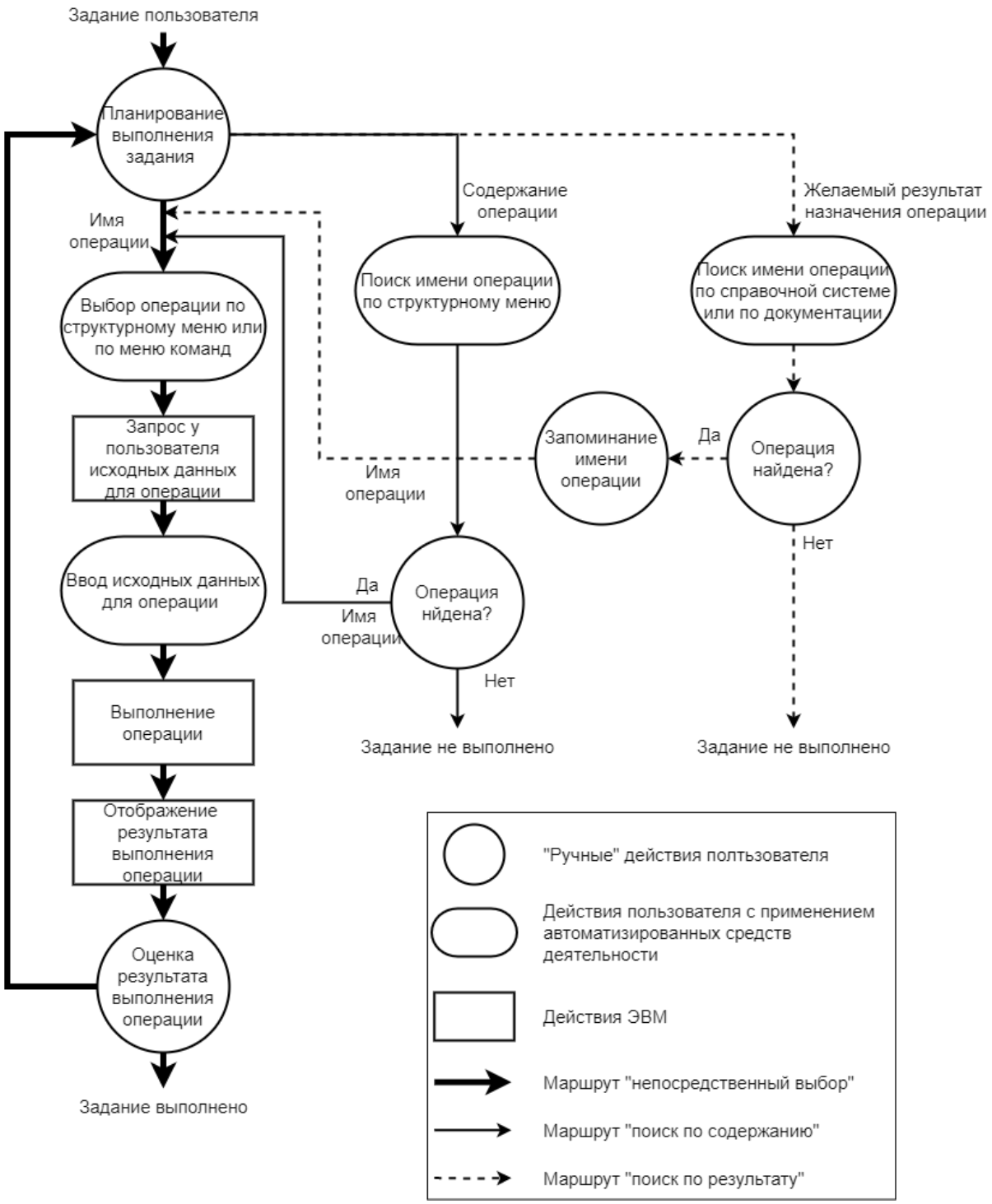

Рис. 2. Обобщённая схема ограничительного метода взаимодействия “пользователь-ЭВМ” для оконного интерфейса

Направляющий метод.

Основой направляющего метода является DT-модель (модель диалоговой транзакции).

В отличие от ограничительного метода, направляющий метод основан на описании в ЭВМ модели пользовательского задания как цели. Каждая из целей соответствует определенному пользовательскому заданию, которое может выполнить ЭВМ во взаимодействии с пользователем. Выбор и целенаправленное упорядочивание подзадач, приводящих к выполнению пользовательского задания, совершает не пользователь, а ЭВМ. От пользователя требуется в ответ на запросы ЭВМ ввести входные данные, необходимые для этих операций.

Направляющий метод взаимодействия “пользователь-ЭВМ” состоит из следующих основных этапов:

- информирование пользователя о множестве допустимых заданий, которые может выполнять ЭВМ в рамках данного приложения;
- выбор пользователем задания по меню заданий и передача его ЭВМ на выполнение;
- планирование процесса взаимодействия при выполнении задания;
- ввод пользователем данных, необходимых ЭВМ для выполнения задания;
- передача пользователю результатов выполнения задания и их оценка пользователем.

Обобщенная схема направляющего метода взаимодействия приведена на рис. 3. В соответствии с этой схемой пользователь должен выбрать в меню заданий свое задание и передать его ЭВМ на выполнение. Выбранное задание рассматривается в ЭВМ как цель пользователя. Достижение этой цели обеспечивается в ходе последующего диалога с пользователем, в котором инициатива взаимодействия принадлежит ЭВМ.

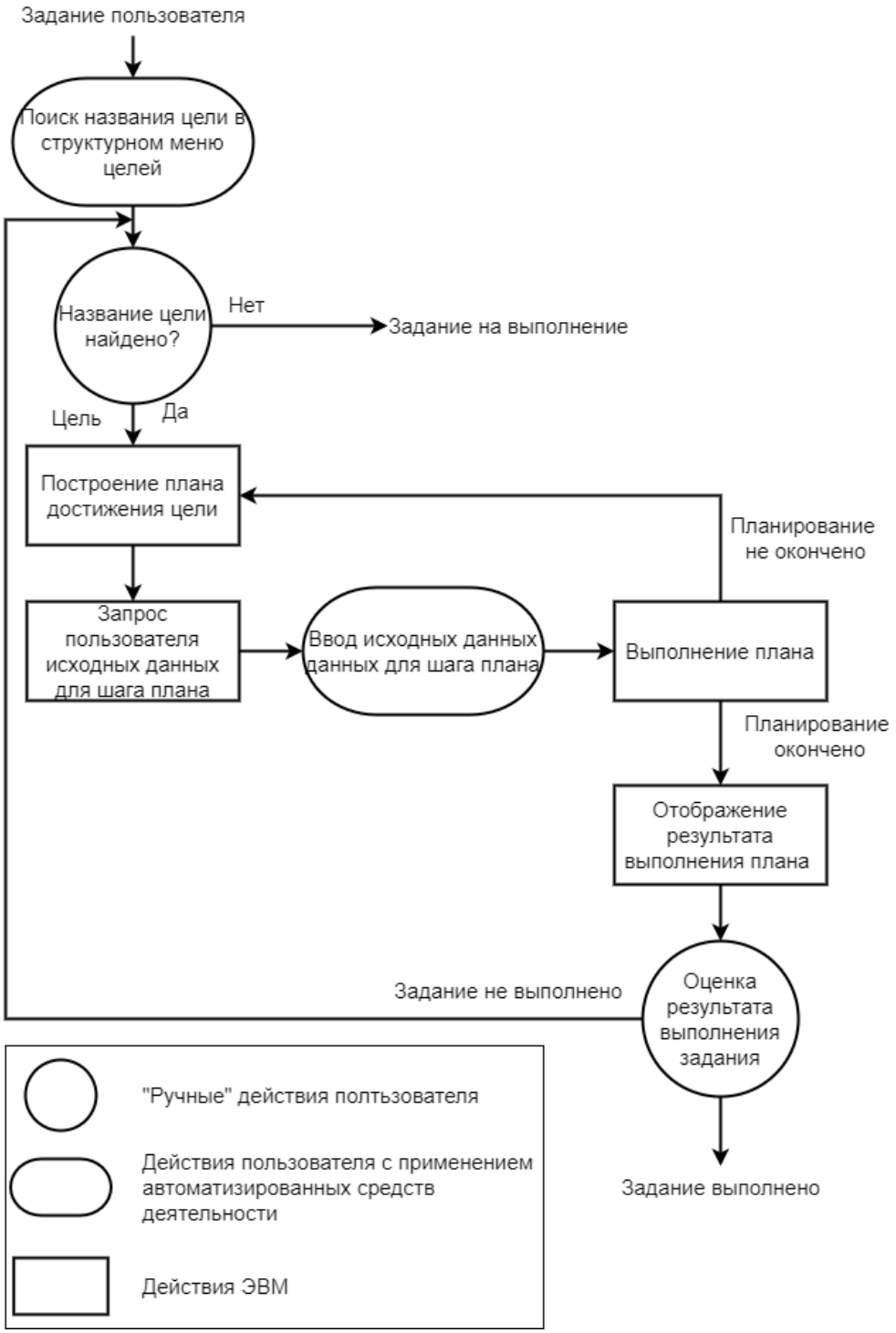

Рис. 3. Обобщённая схема направляющего метода взаимодействия “пользователь-ЭВМ” для оконного интерфейса

В отличие от схемы ограничительного метода, в рассматриваемой схеме существует только один маршрут движения, т. е. здесь нет зависимости результата от степени пользовательской подготовки. После передачи пользовательской цели в ЭВМ дальнейшее взаимодействие не требует

инициативы пользователя вплоть до этапа оценки результата достижения цели. Планирование процесса взаимодействия на множестве ДТ осуществляет ЭВМ в соответствии с DT-моделью диалога. Данные, необходимые ЭВМ для достижения принятой от пользователя цели, вводятся пользователем только по запросу, т.е. на этой стадии он выступает в роли реагирующего участника.

Итак, главное отличие направляющего метода взаимодействия “пользователь-ЭВМ” по сравнению с ограничительным методом состоит в том, что инициатива взаимодействия по достижению цели пользователя принадлежит не пользователю, а ЭВМ. Это означает, что ЭВМ играет активную роль в целенаправленном взаимодействии по выполнению задания пользователя.

Сопоставление схемы направляющего метода взаимодействия (рис. 2) со схемой ограничительного метода (рис. 3) позволяет наглядно судить об упрощении работы пользователя за счет того, что пользователю уже не требуется знать, как выполнить то или иное задание, поскольку планирование процесса взаимодействия выполняет не он сам, а ЭВМ.

Некоторые принципы направляющего метода используются в настоящей работе. Метод выше предполагает построение интерфейса вручную.

1.2 Методический подход к созданию универсального пользовательского интерфейса

При изучении статьи [4] были выделены составные элементы подхода к созданию универсального средства построения пользовательского интерфейса программных средств.

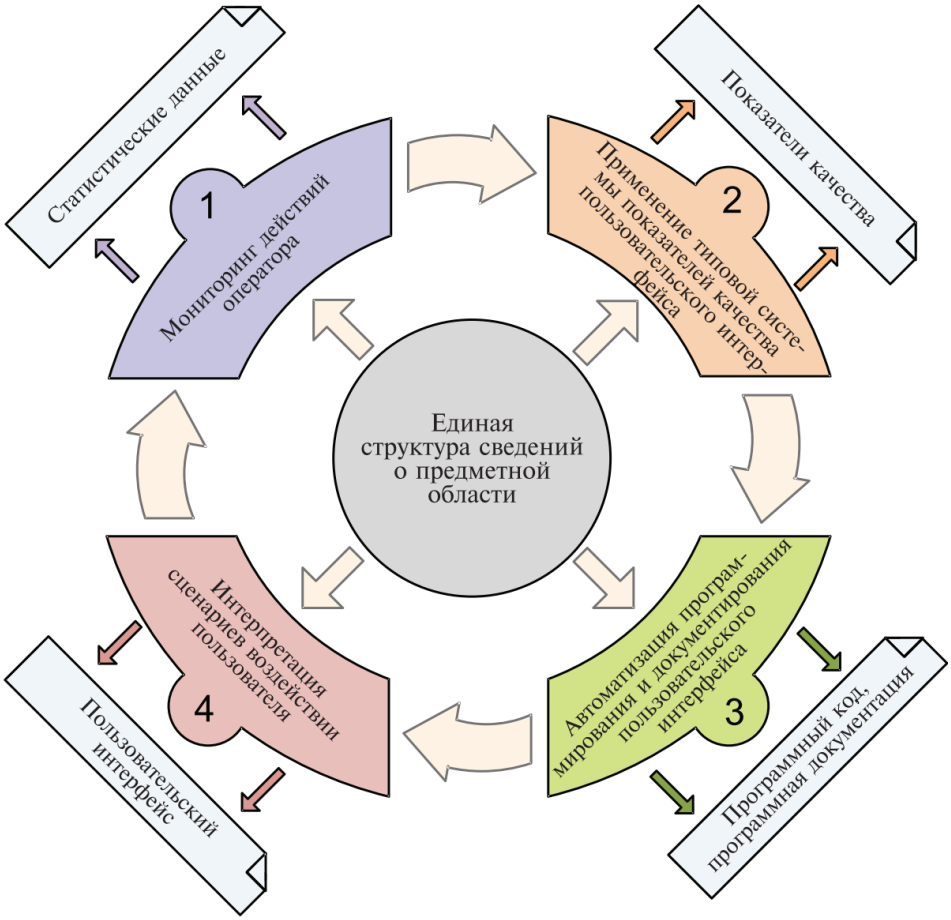

Рис. 4. Составные элементы подхода к созданию универсального средства построения пользовательского интерфейса программных средств

Мониторинг действий оператора (блок 1 на рис. 4) позволяет осуществлять сбор и накопление статистики деятельности оператора во время эксплуатации программных средств. Оператор вводит входные параметры с помощью технических средств ввода данных: клавиатуры, манипулятора «мышь», фотокамеры, микрофона, сканера, специализированных панелей кнопок и переключателей, внешних носителей информации и т. п. Каждая атомарная операция основывается на низкоуровневых сигналах от средства ввода, преобразуемых программной средой в воздействия ввода оператора за счет событийной обработки.

Оценка качества пользовательского интерфейса обеспечивается за счет:

- применения типовой системы показателей качества (блок 2 на рис. 4);
- разработки методики оценки качества;

- выполнения мероприятий по оценке показателей качества.

Система показателей качества пользовательского интерфейса основывается на сведениях представленных типов:

- аппаратные события – события, которые возникают от действия всех технических средств при работе с периферийным оборудованием, а также вследствие использования каналов связи и удаленных подключений;
- события приложения – события, связанные с операциями получения, обработки и преобразования данных, обеспечения политики безопасности и уровней доступа, т. е. все эти события появляются в процессе функционирования программы в программной среде;
- пользовательские события – события, предусмотренные программой, происходят в результате воздействий пользователя на элементы управления интерфейса.

Величины, с помощью которых можно определить качество выполнения операции:

- среднее время выполнения операции;
- число ошибок (количество итераций), возникающих за период работы;
- среднее время возникновения ошибки.

Показатели качества пользовательского интерфейса выражаются в формате:

- время выполнения операций (действий);
- число действий, необходимых для выполнения операции;
- число атомарных операций, необходимых для выполнения действия;
- число пустых воздействий, выполненных оператором;
- число ошибочных атомарных операций.

“Автоматизация программирования и документирования пользовательского интерфейса” (блок 3 на рис. 4) подразумевает возможность автоматизированного документирования интерфейса программы.

Основные мероприятия жизненного цикла пользовательского интерфейса:

- задание требований;
- проектирование;
- разработка, программирование;
- оценка качества и испытания;
- документирование (описание).

Каждое мероприятие жизненного цикла интерфейса взаимосвязано с UML моделями, описывающими интерфейс, которые в ходе итерационного процесса разработки корректируются и используются для получения документов.

В соответствии с подходом предусматривается автоматизация программирования макетов пользовательского интерфейса, основанная на таком выборе паттерна проектирования программного кода, который позволит инкапсулировать механизмы обработки данных и управления от элементов управления интерфейса, но при этом обеспечит связь с моделью данных и мониторинг действий оператора.

Можно использовать UML модель сценариев применения пользовательского интерфейса (модель сценариев воздействий пользователя (СВП)), описывающая различные стороны функционирования программы, которая выражает на определенном уровне абстракции порядок взаимодействия пользователя с программным средством. Модель используется для декомпозиции и формирования однозначного понимания сведений по совокупности функций и режимов работы разрабатываемого программного средства, а также для обеспечения возможности автоматизированного документирования и построения кода интерфейса программы.

С помощью UML модели СВП решаются следующие задачи:

- прототипирование и/или макетирование пользовательского интерфейса при разработке программного средства;
- формализация существующего пользовательского интерфейса;

- описание деятельности пользователя при эксплуатации программных средств, выраженное в виде набора осуществленных сценариев воздействий на элементы управления программы.

Отображение некоторого абстрактного сценария осуществляет механизм его интерпретации (блок 4 на рис. 4) в стандартные программные процедуры, характерные для выбранной программно-аппаратной платформы. Наличие интерпретатора предписывает применение некоторой формальной логики (языка), с помощью лексем которой выражаются любые сценарии. Одна лексема описывает типовую атомарную операцию над элементом управления пользовательского интерфейса, идентифицирующие сведения о которой содержатся в параметрах лексемы.

Каждая атомарная операция, которая вызвана воздействиями пользователя, на диаграмме СВП представляет собой дугу графа, вершины графа — элементы управления пользовательского интерфейса, веса графа — комплексные показатели, характеризующие:

- количество выполнений атомарной операции;
- время выполнения атомарной операции;
- количество ошибок, связанных с выполнением атомарной операции.

Данный метод подобен тому, что предлагается в данной работе, но он комплексный, сложный в реализации и больше подойдёт для создания интерфейса некоторого интернет-ресурса, который требует существенного количество специальных элементов управления.

1.3 Построение пользовательского интерфейса с использованием интерактивного машинного обучения

В процессе изучения статьи [5] были выделены этапы построения GUI с использованием интерактивного машинного обучения.

На первом этапе производится сбор входных данных. В качестве таких данных будут выступать частота, последовательность, достигаемый результат и время между применениями рассматриваемых функций. Исследованием в данной области занимается человеко-компьютерное взаимодействие, где в настоящее время основную роль играет машинное обучение.

На основании собранных данных проводится обучение, целью которого является сократить путь для достижения конкретного результата, сокращение количества шагов и затрачиваемого времени для выполнения идентичных задач. Для обучения используется алгоритм дерева градиентного повышения.

Алгоритм обучения выбирает лучший порог для каждого. Использование матрицы Гессиана и весов позволяет вычислять прирост информации, вызванный применением каждой функции и правила принятия решения для узла.

Обучение может производиться на любом приложении с графическим пользовательским интерфейсом, имеющем длинные цепочки выполнения действий, например, пакет офисных приложений. По результатам обучения строится последовательность действий для достижения необходимого результата. При её построении учитывается время поиска элемента интерфейса, его доступность (подмножество действий необходимых к выполнению для получения доступа к данному элементу), соответствие описания и ожидаемого результат (использование других элементов, требующих большего числа шагов для достижения результата).

На основании полученных результатов вносятся корректировки в существующий интерфейс, после чего обучение продолжается.

Данный является трудоемким с точки зрения реализации, так как производится с использованием машинного обучения.

Все рассмотренные методы трудны в реализации и не предполагают автоматизацию. В результате обзора литературы было подтверждена необходимость программного обеспечения для простой генерации интерфейса. Предлагаемый метод сравнительно прост в реализации и позволяет с минимальными трудозатратами строить формы ввода специалистам, не обладающим навыками программирования.

2 Разработка web-приложения

Перед тем, как описать предлагаемый в данной работе метод построения интерфейса, далее предоставлено описание разработки web-приложения, которое будет работать со сгенерированным интерфейсом и дополнительным кодом для корректной работы с полями ввода интерфейса.

2.1 Использованные программные решения

В рамках курсового проекта использовались Django, Docker и Nginx.

Django — это высокоуровневый Python веб-фреймворк для бэкенда, который позволяет быстро создавать безопасные и поддерживаемые веб-сайты. Фреймворк — это программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта [6].

Главная цель фреймворка Django — позволить разработчикам вместо того, чтобы снова и снова писать одни и те же части кода, сосредоточиться на тех частях своего приложения, которые являются новыми и уникальными для их проекта.

Достоинства Django:

- Масштабируемый. Django использует компонентную архитектуру, то есть каждая её часть независима от других и, следовательно, может быть заменена или изменена, если это необходимо. Чёткое разделение частей означает, что Django может масштабироваться при увеличении трафика, путём добавления оборудования на любом уровне;
- Разносторонний. Django может быть использован для создания практически любого типа веб-сайтов;
- Безопасный. Django помогает разработчикам избежать многих распространённых ошибок безопасности, предоставляя фреймворк, разработанный чтобы «делать правильные вещи» для автоматической защиты сайта. Например, Django предоставляет безопасный способ

управления учётными записями пользователей и паролями, избегая распространённых ошибок, таких как размещение информации о сеансе в файлы cookie, где она уязвима или непосредственное хранение паролей вместо хэша пароля;

- Переносным. Django написан на Python, который работает на многих платформах;
- Удобным в сопровождении. Код Django написан с использованием принципов и шаблонов проектирования, которые поощряют создание поддерживаемого и повторно используемого кода.

Docker — программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска веб-приложений в средах с поддержкой контейнеризации [7]. Он нужен для более эффективного использование системы и ресурсов, быстрого развертывания готовых программных продуктов, а также для их масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы.

Основной принцип работы Docker — контейнеризация приложений. Этот тип виртуализации позволяет упаковывать программное обеспечение по изолированным средам — контейнерам. Каждый из этих виртуальных блоков содержит все нужные элементы для работы приложения. Это дает возможность одновременного запуска большого количества контейнеров на одном хосте.

Достоинства использования Docker:

- Минимальное потребление ресурсов — контейнеры не виртуализируют всю операционную систему (ОС), а используют ядро хоста и изолируют программу на уровне процесса;
- Скоростное развертывание — вспомогательные компоненты можно не устанавливать, а использовать уже готовые docker-образы (шаблоны);
- Удобное скрытие процессов — для каждого контейнера можно использовать разные методы обработки данных, скрывая фоновые процессы;

- Работа с небезопасным кодом — технология изоляции контейнеров позволяет запускать любой код без вреда для ОС;
- Простое масштабирование — любой проект можно расширить, внедрив новые контейнеры;
- Удобный запуск — приложение, находящееся внутри контейнера, можно запустить на любом docker-хосте;
- Оптимизация файловой системы — образ состоит из слоев, которые позволяют очень эффективно использовать файловую систему.

Определения Docker:

- Docker-образ (Docker-image) — файл, включающий зависимости, сведения, конфигурацию для дальнейшего развертывания и инициализации контейнера;
- Docker-контейнер (Docker-container) — это легкий, автономный исполняемый пакет программного обеспечения, который включает в себя все необходимое для запуска приложения: код, среду выполнения, системные инструменты, системные библиотеки и настройки;
- Docker-файл (Docker-file) — описание правил по сборке образа, в котором первая строка указывает на базовый образ. Последующие команды выполняют копирование файлов и установку программ для создания определенной среды для разработки;
- Docker-клиент (Docker-client / CLI) — интерфейс взаимодействия пользователя с Docker-демоном. Клиент и Демон — важнейшие компоненты “движка” Докера (Docker Engine). Клиент Docker может взаимодействовать с несколькими демонами;
- Docker-демон (Docker-daemon) — сервер контейнеров, входящий в состав программных средств Docker. Демон управляет Docker-объектами (сети, хранилища, образы и контейнеры). Демон также может связываться с другими демонами для управления сервисами Docker;

- Том (Volume) — эмуляция файловой системы для осуществления операций чтения и записи. Она создается автоматически с контейнером, поскольку некоторые приложения осуществляют сохранение данных;
- Реестр (Docker-registry) — зарезервированный сервер, используемый для хранения docker-образов;
- Docker-хаб (Docker-hub) или хранилище данных — репозиторий, предназначенный для хранения образов с различным программным обеспечением. Наличие готовых элементов влияет на скорость;
- Docker-хост (Docker-host) — машинная среда для запуска контейнеров с программным обеспечением;
- Docker-сети (Docker-networks) — применяются для организации сетевого интерфейса между приложениями, развернутыми в контейнерах.

2.2 Разработка тестового web-приложения

На рис. 5 представлено содержание проекта.

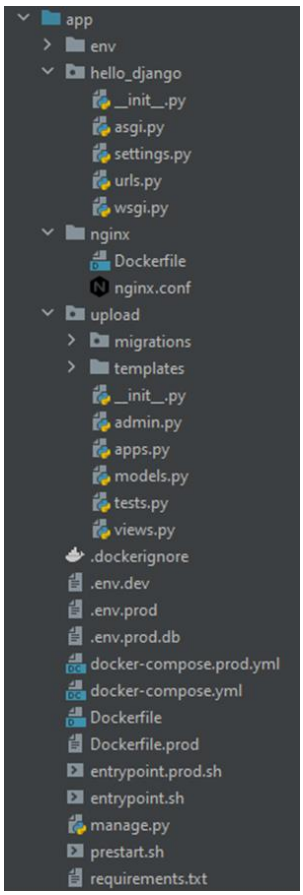

Рис. 5. Содержания проекта

- Файл settings.py содержит в себе все настройки проекта. Здесь регистрируются приложения, задаётся размещение статичных файлов, настройки базы данных и т.д.
- Файл urls.py задаёт ассоциации URL (унифицированный указатель ресурса) адресов с представлениями. Функция представления – это функция Python, которая принимает веб-запрос и возвращает веб-ответ. В случае данной работы ответом будет HTML-содержимое веб-страницы.
- wsgi.py используется для определения связи между Django приложением и веб-сервером.
- manage.py используется для создания приложений, работы с базами данных и для запуска отладочного сервера.
- В файле Dockerfile.prod (листинг А.1) описана последовательность команд, которые надо выполнить. На основе этого файла будет создан образ (docker image).

В Dockerfile.prod используется многоступенчатая сборка (multi-stage build), чтобы уменьшить конечный размер образа. builder — это временный образ, которое используется для сборки Python. Затем он копируются в конечный производственный образ, а образ builder отбрасывается. Так же был создан пользовател app без полномочий root. По умолчанию Docker запускает контейнерные процессы как root внутри контейнера, и если кто-то получит доступ на сервер и к контейнеру, то проникший на сервер пользователь тоже станет root, что, очевидно, не желательно. Таким образом увеличивается безопасность.

Вначале указан образ, на котором будем основываться. В нашем случае python 3.9. Устанавливается рабочая директория в контейнере с помощью WORKDIR. Далее устанавливаются переменные окружения:

- PYTHONDONTWRITEBYTECODE означает, что Python не будет пытаться создавать файлы .pyc;
- PYTHONUNBUFFERED гарантирует, что вывод консоли выглядит знакомым и не буферизируется Docker.

Затем команды для установки соответствующих пакетов, необходимых для Psycopg2. Из директории, где находится Dockerfile.prod, копируется файл зависимостей requirements.txt в рабочую директорию контейнера. Далее с помощью команды RUN исполняются перечисленные в ней команды, что приводит к установкам зависимостей. Затем копируется вся дериктория проекта в контейнер.

Были созданы папки staticfiles и mediafiles, так как docker-compose монтирует именованные тома как root. И так как используется пользователь app не обладает полномочиями root, таким образом можно получить ошибку отказа в разрешении при запуске команды collectstatic, если каталог еще не существует.

Был создан файл docker-compose.prod.yml (листинг 1). docker-compose позволяет управлять многоконтейнерностью. То есть можно запустить сразу несколько контейнеров, которые будут работать между собой. В нашем случае мы создаем контейнер с базой данных db, наш основной контейнер web и nginx.

Листинг 1. docker-compose.prod.yml

```
version: '3.8'

services:
  web:
    build:
      context: ./
      dockerfile: Dockerfile.prod
    command: gunicorn hello_django.wsgi:application --
bind 0.0.0.0:8080
    volumes:
      - static_volume:/home/app/web/staticfiles
      - media_volume:/home/app/web/mediafiles
    env_file:
      - ./.env.prod
    depends_on:
      - db
  db:
    image: postgres:13.0-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    env_file:
      - ./.env.prod.db
  nginx:
    build: ./nginx
    volumes:
      - static_volume:/home/app/web/staticfiles
      - media_volume:/home/app/web/mediafiles
    ports:
      - 8084:80
      - 443:443
    depends_on:
      - web
volumes:
  postgres_data:
  static_volume:
  media_volume:
```

В самом начале указывается версия docker-compose. Далее указываются services (контейнеры).

- web. Сбор проекта (build) делается на основе Dockerfile. Далее указывается команда для запуска сервера. Указываются volumes, что значит, что всё из

указанных директорий используется в контейнере. Далее указываются порты (сервер Django запускается в контейнере на порте 8084 и этот порт перебрасывается на нашу хост машину). И в конце сервиса web указывается зависимость от сервиса db, то есть web не сможет работать без db.

- db. Выбирается образ postgres. Далее указываются volumes, чтобы сохранять наши данные. Также настраиваются переменные среды.
- nginx. Чтобы он действовал как обратный прокси-сервер для обработки клиентских запросов, а также для обслуживания статических файлов. Для работы Nginx была создана новая директория с соответствующим названием. В данной директории были созданы файлы Dockerfile (листинг 2) и nginx.conf (листинг 3).

Листинг 2. Dockerfile в директории nginx

```
FROM nginx:1.21-alpine
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d
# удаляем настройки по умолчанию
# копируем наши настройки
```

Листинг 3. nginx.conf

```
server {
    listen 80;
    location /static/ {
        alias /home/app/web/staticfiles/;
    }
    location /media/ {
        alias /home/app/web/mediafiles/;
    }
    location / {
        proxy_pass http://web:8080;
        proxy_set_header Host $http_host;
        proxy_set_header Connection "upgrade";
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
```

```
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header Upgrade $http_upgrade;
        proxy_connect_timeout       600;
        proxy_send_timeout          600;
        proxy_read_timeout          600;
        send_timeout                600;
    }
}
```

Команды, представленные на листинге 4, позволяют получить переменные окружения записанные в файл .env.prod (листинг 6) и записать их в переменные в файле проекта settings.py.

Листинг 4. Обновлённые переменные в settings.py

```
SECRET_KEY = os.environ.get("SECRET_KEY")

DEBUG = int(os.environ.get("DEBUG", default=0))

ALLOWED_HOSTS =
os.environ.get("DJANGO_ALLOWED_HOSTS").split(" ")
```

SECRET_KEY — это секретный ключ, который используется Django для поддержки безопасности сайта.

DEBUG. Включает подробные сообщения об ошибках, вместо стандартных HTTP статусов ответов. Должно быть изменено на False на сервере, так как эта информация очень много расскажет взломщикам.

ALLOWED_HOSTS — это список хостов/доменов, для которых может работать текущий сайт.

Так же в setings.py были обновлены настройки базы данных (листинг 5) на основе переменных окружения, определённых в .env.prod.

Листинг 5. Обновлённые переменные в settings.py

```
    DATABASES = {
        "default": {
```

```
            "ENGINE": os.environ.get("SQL_ENGINE",
"django.db.backends.sqlite3"),
            "NAME": os.environ.get("SQL_DATABASE",
BASE_DIR / "db.sqlite3"),
            "USER": os.environ.get("SQL_USER", "user"),
            "PASSWORD": os.environ.get("SQL_PASSWORD",
"password"),
            "HOST": os.environ.get("SQL_HOST",
"localhost"),
            "PORT": os.environ.get("SQL_PORT", "5432"),
        }
    }
```

Листинг 6. Содержимое файла .env.prod

```
DEBUG=0
SECRET_KEY=change_me
DJANGO_ALLOWED_HOSTS=localhost 195.19.40.68 [::1]
SQL_ENGINE=django.db.backends.postgresql
SQL_DATABASE=hello_django_prod
SQL_USER=hello_django
SQL_PASSWORD=hello_django
SQL_HOST=db
SQL_PORT=5432
DATABASE=postgres
```

Для работы с медиафайлами был создан новый модуль Django под названием upload, то есть была создана новая директория upload (новый модуль создаётся командой в терминале: “ docker-compose exec web python manage.py startapp upload”) и добавлен новый модуль в INSTALLED_APPS в settings.py.

В созданной директории был изменен файл views.py (листинг 8), была создана папка для шаблонов “templates” и в ней был создан новый шаблон title.html для тестового запуска. Так же был изменен файл app/hello_django/urls.py (листинг 7).

Листинг 7. urls.py

```
from django.contrib import admin
from django.urls import path
from django.conf import settings
```

```
from django.conf.urls.static import static
from upload.views import image_upload
urlpatterns = [
    path("", gui, name="gui"),
    path("image_upload/", image_upload, name="upload"),
    path("admin/", admin.site.urls),]

if bool(settings.DEBUG):
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

Листинг 8. views.py

```
from django.shortcuts import render
from django.core.files.storage import FileSystemStorage

def gui(request):
    return render(request, "title.html")
```

2.3 Запуск web-приложения на тестовом сервере

После входа на сервер через консоль и запуска приложения, при переходе по http://195.19.40.68:8084, открывается страница (рис. 6).

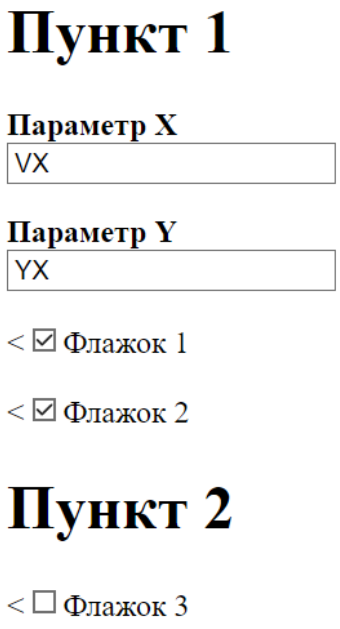

Рис. 6. Тестовая страница

Далее проверим работу базы данных. Входим в контейнер и создаём суперпользователя (рисунок 7).

```
PS D:\PyCharm\django-on-docker\app> docker-compose exec web sh
/usr/src/app # ls
Dockerfile              docker-compose-old.yml   entrypoint.prod.sh      hello_django            requirements.txt
Dockerfile.prod         docker-compose.prod.yml  entrypoint.sh           manage.py
catalog                 docker-compose.yml       env                     nginx
/usr/src/app # python manage.py createsuperuser
Username (leave blank to use 'root'): Arthur
Email address:
Password:
Password (again):
Superuser created successfully.
/usr/src/app #
```

Рис.7 Создание суперпользователя

Переходим по адресу http://195.19.40.68:8084/admin и вводим имя и пароль суперпользователя. Вход выполнен (рисунок 8).

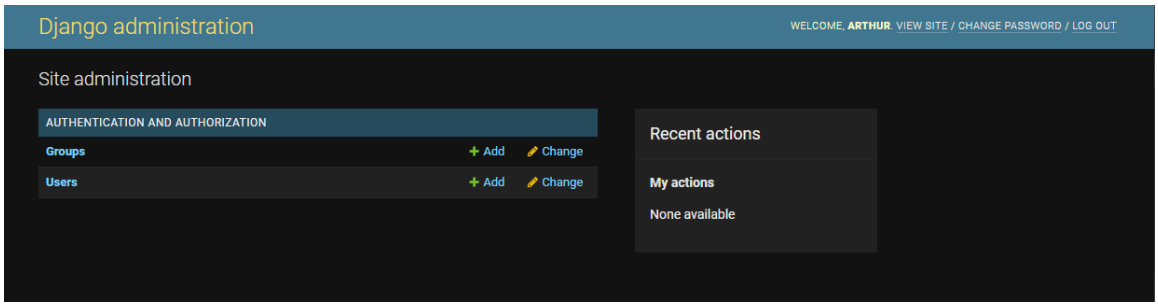

Рис. 8 Вход успешно выполнен

Проверка на создание таблиц по умолчанию на рисунке 9.

```
PS D:\PyCharm\django-on-docker\app> docker-compose exec db psql --username=hello_django --dbname=hello_django_dev
psql (13.0)
Type "help" for help.

hello_django_dev=# \l
                                  List of databases
      Name       |    Owner     | Encoding |  Collate   |   Ctype    |       Access privileges
-----------------+--------------+----------+------------+------------+-------------------------------
 hello_django_dev | hello_django | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres         | hello_django | UTF8     | en_US.utf8 | en_US.utf8 |
 template0        | hello_django | UTF8     | en_US.utf8 | en_US.utf8 | =c/hello_django             +
                  |              |          |            |            | hello_django=CTc/hello_django
 template1        | hello_django | UTF8     | en_US.utf8 | en_US.utf8 | =c/hello_django             +
                  |              |          |            |            | hello_django=CTc/hello_django
(4 rows)
```

Рис. 9. Проверка таблиц

3 Генерация GUI на основе aINI

В работе предлагается автоматизация генерации пользовательских интерфейсов на основе интерпретации текстовых файлов, подготовленных заранее в формате aINI. aINI (Advanced INI) - формат для описания входных данных различных задач, основанный на языке INI [1].

Предположим, дан следующий aINI-файл (листинг 8), тогда на выходе будет получена страница, представленная на рисунке 10 и рисунке 11.

Листинг. 8. Пример aINI-файла

```
[sec1]//Вкладка 1
x=25//Параметр X
y=@y@//Параметр Y
box1=[0]{0|1}//Флажок 1
box2=[1]{0|1}//Флажок 2
[sec2]//Вкладка 2
q=ABC//Параметр Q
box3=[0]{0|1}//Флажок 3
ParametersFile=[file]//Выберите требуемый файл
//МГТУ им. Н. Э. Баумана
```

Рис. 10 HTML-страница (первая вкладка)

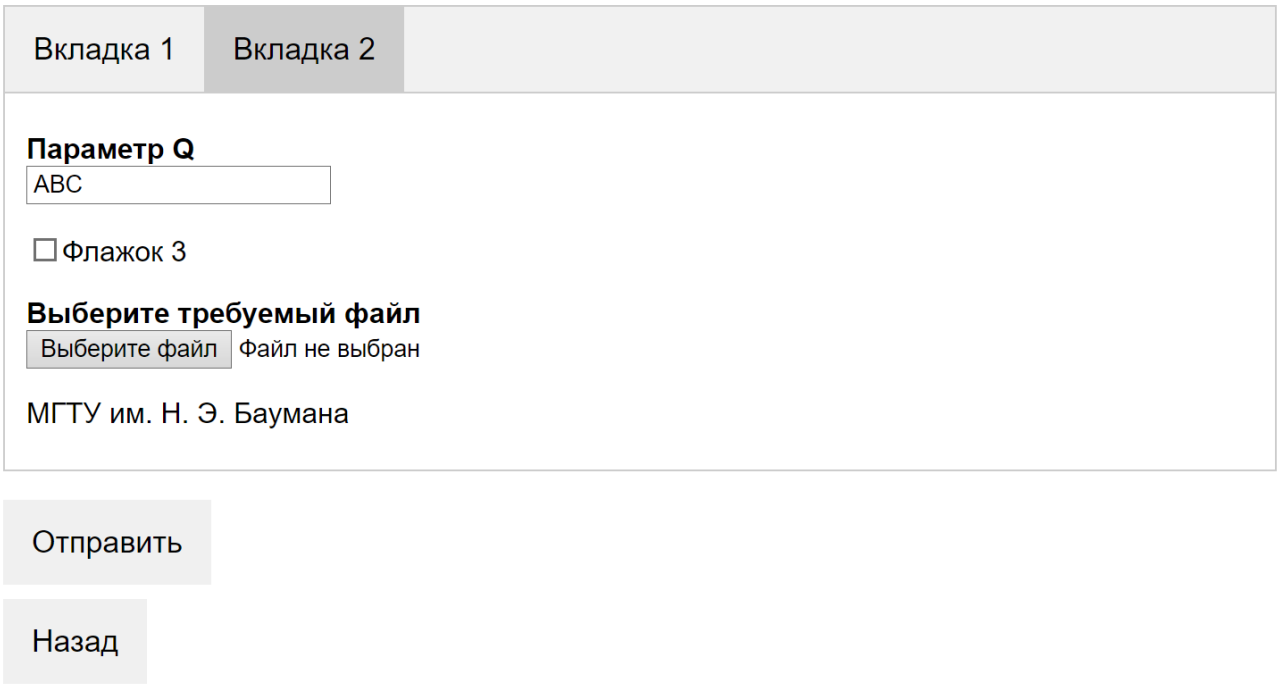

Рис.11 HTML-страница (вторая вкладка)

Из небольшого числа строк на aINI было получено около сотни строк кода на языке HTML с использованием CSS и JavaScript, что позволило создать на одной странице несколько вкладок, наполненных различными формами ввода и дополнительной информацией.

Каждая строка на языке aINI – это элемент интерфейса, который будет представлен пользователю на экране. Некоторые из них имеют начальные значения, которые в том числе задаются в aINI файле. Синтаксис каждого элемента на aINI относительно прост в понимании. Поэтому специалист, не обладающий навыками программирования, сможет генерировать интерфейс с минимальными трудозатратами.

Далее будет рассмотрен синтаксис каждого элемента в aINI-коде и что будет выведено на пользовательский экран на основе этого кода.

3.1 Соответствие aINI кода элементам интерфейса

Была реализована генерация следующих элементов интерфейса:

- переключаемые вкладки (рис. 12),

- текст (рис. 13),
- ссылка (рис. 14),
- поле ввода (рис. 15),
- флажок (рис. 16),
- поле выбора файла (рис. 17).

Переключаемые вкладки

Описание в aINI:

```
[sec1]//Пункт 1
[sec2]//Пункт 2
```

Выше описаны две вкладки: Пункт 1, Пункт 2. В квадратных скобках в начале строки написано название переменной вкладки, оно будет использоваться и в качестве самого наименования вкладки, если будет отсутствовать название, которое пишется в конце строки объявления вкладки после символов «//».

На рисунке 12 показан элемент, описанный выше, выведенный на экране пользователя.

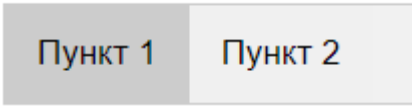

Рис. 12. Переключаемые вкладки

Вывод в HTML-коде предоставлен на листинге 9. В нем были использованы названия для вкладок из описания выше («Пункт 1» и «Пункт 2»). Будут созданы две кнопки с классом tablinks. Класс определяет стиль для индивидуального элемента веб-страницы. По умолчанию нажатой считается самая левая кнопка, то есть та, что была определена самой первой в aINI файле, поэтому, при открытии страницы, на экране будут показаны элементы, принадлежащие этой вкладке.

Листинг. 9. HTML-код кнопки вкладки

```
<div class="tab">
  <button class="tablinks"
onclick="tabs(event,'section_0')"
id="defaultOpen">Пункт 1</button>
  <button class="tablinks"
onclick="tabs(event,'section_1')">Пункт 2</button>
</div>
```

Содержимое каждой вкладки в виде HTML-кода обладает классом tabcontent и скрыто по умолчанию. С помощью CSS (листинг А.2) и JavaScript (JS) (листинги 10 - 12) достигается функционирование кнопок перехода между вкладками и изменение стилей кнопок и вкладок. Когда пользователь нажимает на кнопку, на экране демонстрируется содержимое вкладки, которое соответствует этой кнопке.

На листинге 10 код, отвечающий за получение всех элементов с классом tabcontent (то есть элементы, принадлежащие вкладкам) и скрытие их с экрана. Это позволит скрыть все вкладки, чтобы потом показать только ту, что будет активна.

Листинг. 10. JavaScript код скрытия вкладок

```
    tabcontent =
document.getElementsByClassName("tabcontent");
    for (i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = "none";
    }
```

На листинге 11 показан код, отвечающий за получение всех элементов с классом tablinks (то есть кнопки для перехода между вкладками) и удаления у них класса active. Класс active обеспечивает определение стиля нажатой для кнопки на экране в форме вывода на веб странице. Поэтому при удалении класса active у элемента кнопки, кнопка не будет представлена на экране пользователя не нажатой, а в первоначальном состоянии, которое определяется стилями, написанными на языке CSS.

Листинг. 11. JavaScript код удаления класса active

```
tablinks = document.getElementsByClassName("tablinks");
for (i = 0; i < tablinks.length; i++) {
   tablinks[i].className = tablinks[i].className.replace(" active", "");
}
```

На листинге 12 показан код для изображения на экране текущей вкладки и добавление класса active кнопке, которая открыла текущую вкладку.

Листинг. 12. JavaScript код изображения текущей вкладки

```
    document.getElementById(tab_id).style.display =
"block";
    evt.currentTarget.className += " active";
```

Функция tabs (функция, которая включает в себя весь JavaScript-код выше), вызывается при событии нажатие пользователем на элемент-кнопку, у элементов кнопок вкладок, например, листинг 9.

Итого весь JS-код реализует следующее. Когда пользователь открывает страницу, на экране показаны все доступные кнопки для вызова отображения вкладок, но на экране вначале выведена вкладка по умолчанию (самая первая определённая в aINI файле вкладка), визуализируя кнопку этой вкладки как нажатую. Далее, если пользователь нажимает на кнопку, соответствующую какой-либо другой вкладке, нажатая кнопка принимает вид активной, а прошлая вкладка исчезает, а на её месте выводится вкладка с полями ввода, которая была связана с нажатой кнопкой.

Текст

Входные aINI-данные:

```
//здесь пишется текст
```

На рисунке 13 представлен вывод на экран текста, который был дан в описании

на aINI выше («здесь пишется текст»).

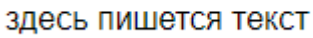

Рис.13. Текст

Выходной HTML-код, который выведет на экран текст, на листинге 8. Текст выводится в элементе <p>, определяющий текстовый абзац в HTML.

Листинг. 13. HTML-код текста

```
<p>здесь пишется текст</p>
```

Ссылка

Входные aINI-данные:

```
[https://bmstu.ru/about]//Дополнительная информация
```

На рисунке 14 показан вывод ссылки на экран на основе aINI-кода выше. Текст для ссылки («Дополнительная информация») и сам адрес, по которому будет произведён переход при нажатии, соответствуют тому, что в данных на aINI.

Рис. 14. Ссылка

HTML-код для вывода на экран ссылки представлен на листинге 14. Ссылка выводится с помощью элемента <a>, определяющий ссылки в HTML.

Листинг. 14. HTML-код ссылки

```
<p><a href="https://bmstu.ru/about">Дополнительная
информация</a></p>
```

Поле ввода

Входные aINI-данные:

```
x=12//Параметр X
```

На рисунке 15 представлен поле ввода данных, построенное на основе aINI-кода. Текст над полем ввода, отражающий суть пола, и значение по умолчанию были взяты из описания на aINI выше. Если вместо значения по умолчанию записать “@переменная@” (в данном случае @x@), то поле будет пустым.

Рис. 15. Поле ввода

HTML-код поля ввода представлен на листинге 15. Имя поля (name), текст над ним и значение по умолчанию (value) соответствуют тому, что в данных на aINI.

Листинг 15. HTML-код поля ввода

```
    <p><b>Параметр X</b><br>
    <input type="text" name="x" value="12"></p>
```

Функция представления отвечает за то, как будут обрабатываться значения, введенные пользователем в форму ввода. Часть функции представления, отвечающая за сохранение в изначальный aINI-файл записанных в данное поле ввода значений, на листинге 16. С помощью request.POST получаются данные, записанные в форму. Открывается файл с изначальными данными, функцией readlines считывается файловый объект input_f построчно. Среди строк файла производится поиск той строки, которая отвечает за рассматриваемую форму ввода, затем в данной строке заменяется значение на

новое. Все рассматриваемые строки последовательно записываются в переменную new_data, чтобы потом выгрузить всё обратно в aINI файл.

Листинг. 16. Часть функции представления для поля ввода

```
    if request.method == "POST":
        if request.POST["x"]:
            x = request.POST["x"]
            new_data = ''
            input = open('/home/app/web/input/input1',
'r', encoding='utf-8')
            input_f = File(input)
            old_data = input_f.readlines()
            for l in old_data:
                if l.find('//Параметр X\n') != -1:
                    l = 'x=' + x + '//Параметр X\n'
                new_data = new_data + str(l)
            output = open('/home/app/web/input/input1',
'w', encoding='utf-8')
            output_f = File(output)
            output_f.write(new_data)
            input_f.close()
            input.close()
            output_f.close()
            output.close()
```

Флажок

Входные aINI-данные:

```
box1=[1]{0|1}//Флажок 1
```

В квадратных скобках указывается значение по умолчанию. «1» – флажок будет активен по умолчанию с помощью добавления к HTML-коду атрибута checked, «0» - флажок будет неактивен.

На рисунке 16 показан флажок на экране пользователя. Текст около флажка и состояние флажка по умолчанию были взяты из описания на aINI выше.

Рис. 16. Флажок

HTML-код флажка на листинге 17. Имя поля (name), текст около флажка и значение элемента (value) и активность флажка соответствуют тому, что в данных на aINI.

Листинг. 17. HTML-код флажка

```
<p><input type="checkbox" name="box1" value="box1"
checked>Флажок 1</p>
```

Часть функции представления, отвечающая за сохранение в изначальный aINI-файл изменений состояния флажка, на листинге 18. С помощью request.POST проверяется был ли нажат флажок. Если да, то переменная box1 будет равна 1, в обратном случае - 0. Открывается файл с изначальными данными, функцией readlines считывается файловый объект input_f построчно. Среди строк файла производится поиск той строки, которая отвечает за рассматриваемый флажок, затем в данной строке заменяется значение на новое. Все рассматриваемые строки последовательно записываются в переменную new_data, чтобы потом выгрузить всё обратно в aINI файл.

Листинг. 18. Часть функции представления для флажка

```
        if "box1" in request.POST:
            box1 = '1'
        else:
            box1= '0'
        new_data = ''
        input = open('/home/app/web/input/input1', 'r',
encoding='utf-8')
        input_f = File(input)
        old_data = input_f.readlines()
        for l in old_data:
```

```
            if l.find('//Флажок 1\n') != -1:
                l = 'box1=[' + box1 + ']{0|1}//Флажок
1\n'
            new_data = new_data + str(l)
        output = open('/home/app/web/input/input1', 'w',
encoding='utf-8')
        output_f = File(output)
        output_f.write(new_data)
        input_f.close()
        input.close()
        output_f.close()
        output.close()
```

Поле выбора файла

Входные aINI-данные:

```
ParametersFile=[file]//Выберите
```

На рисунке 17 представлен поле выбора файла построенное на основе aINI данных выше. Текст над элементом идентичен тому, что содержится в aINI-коде.

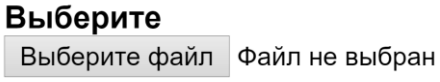

Рис. 17. Поле выбора файла

HTML-код элемента на листинге 19. Текст над элементом выбора файла и имя поля (name) соответствуют тому, что в данных на aINI.

Листинг. 19. HTML-код поля выбора файла

```
    <p><b>Выберите файл</b><br>
    <input type="file" name="ParametersFile"></p>
```

Часть функции представления, отвечающая за сохранение в папку с входными данными файла, выбранного пользователем, на листинге 20. С помощью request.FILES данные файла записываются в переменную ParametersFile. Далее определяется переменная класса FileSystemStorage. Класс FileSystemStorage реализует базовое хранилище файлов в локальной файловой

системе [8]. Далее загруженный пользователем файл сохраняется в файловой системе web-приложения.

Листинг. 20. Часть функции представления для файла

```
        if "ParametersFile" in request.FILES:
            ParametersFile =
request.FILES["ParametersFile"]
            fs = FileSystemStorage()
            filename = fs.save(ParametersFile.name,
ParametersFile)
            file_url = fs.url(filename)
```

3.2 Разработка программы

Блок-схема программы представлена на рисунке 18. Во время разработки программы преобразования данных формата aINI в HTML-код использовался модуль синтаксического анализа для языка Python – Pyparsing [9].

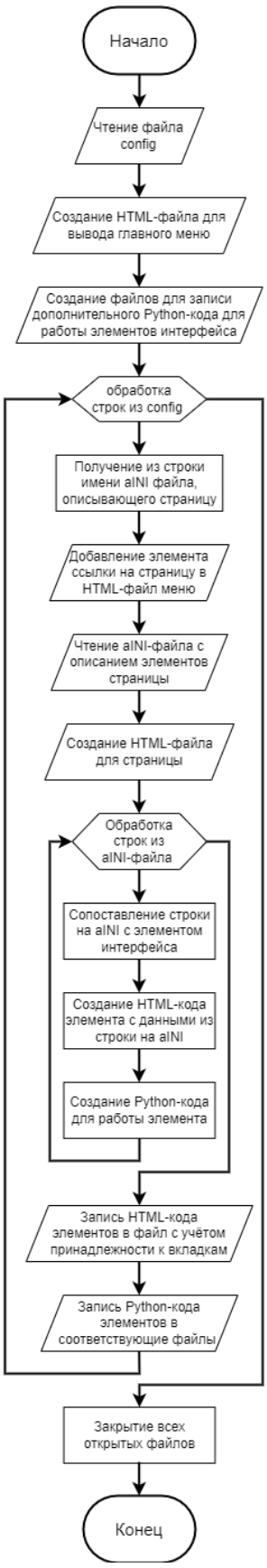

Рис. 18. Блок-схема транслятора с aINI в HTML

После запуска программа в функции main (листинг А.3) вначале открывается файл config с входными данными и создаются файлы для записи

выходных данных. Далее считываются построчно данные из файла config (пример содержимого файла config на листинге 21). В данном файле представлены в квадратных скобках названия искомых файлов, на основе которых будут генерироваться HTML-страницы (содержимое такого файла представлено на листинге 22), до знака "=" прописывается URL для соответствующей страницы, а после "//" наименование страницы. На основе файла config также будет создана страница menu (пример HTML-кода меню – листинг А.4), из которой можно будет перейти на все страницы, перечисленные в config, при нажатии на кнопку, соответствующую одной из страниц. В функции main создаются ещё два текстовых файла: txt_for_urls, txt_for_views. В этих файлах будет записан дополнительный код, который пользователю необходимо будет вставить в соответствующие файлы в web-приложения для корректного взаимодействия приложения с HTML-файлами (в комментариях в данных файлах присутствует информация о том, куда должен быть скопирован код). В файле txt_for_urls.txt записывается код для файла urls.py web-приложения, то есть элементы массива urlpatterns (список всех URL, которые обрабатываются web-приложением). В файле txt_for_views записывается код для файла views.py web-приложения, то есть код функций представления (функция, которая обрабатывает запрос по соответствующему URL адресу) для каждого обрабатываемого URL. Пример файлов txt_for_urls и txt_for_views представлен на листингах 28 и А.5. Наименование страницы и название файла с данными формата aINI считанные из config передаются функции aini_to_html (листинг 28).

Листинг. 21. Пример файла config

```
F1 = [input1]//Test1
F2 = [input2]//Test2
F3 = [input3]//Test3
```

Листинг. 22. Пример полученного input1

```
[sec1]//Вкладка 1
x=25//Параметр X
y=@y@//Параметр Y
```

```
box1=[0]{0|1}//Флажок 1
box2=[1]{0|1}//Флажок 2
[sec2]//Вкладка 2
q=ABC//Параметр Q
box3=[0]{0|1}//Флажок 3
ParametersFile=[file]//Выберите требуемый файл
//Московский государственный технический университет им.
Н. Э. Баумана - российский национальный исследовательский
университет, научный центр, особо ценный объект
культурного наследия народов России.
[https://bmstu.ru/about]//Дополнительная информация
```

В функции aini_to_html(листинг А.6) построчно считываются данные из aINI файла и происходит распознавание строки в соответствии с шаблонами в парсере (листинг 23). Содержимое каждой вкладки формы (то есть элементы, описанные между вкладками в aINI файле) записывается последовательно в один элемент списка elements_list, а сами вкладки записываются в список sections_list. При использовании pyparsing, парсер вначале был написан для отдельных ключевых элементов (например, числовые значения, текст из английских и русских букв, ссылка и т. п.), а потом из отдельных частей получается парсер для всей строки aINI-кода.

Распознавание строк происходит в функции parsing (листинг 24), которая возвращает список из распознанных частей строки и номер распознанного элемента интерфейса для дальнейшей генерации. Распознавание строки происходит в блоке с помощью функции parseString(s).asList() молуля pyparsing. Данная функция получает на вход строку s и проводит синтаксический разбор данной строки по шаблону, через который вызвана функция (например, в случае parse_file_selection.parseString(s).asList() строка будет разобрана как строка элемента выбора файла). Если разбор был выполнен без возврата исключения (тип данных в Python, сообщающий программисту об ошибках [10]), то в возвращаемые переменные записываются соответствующие значения, иначе благодаря конструкции try-except происходит перехват исключения.

На основе списка и номера элемента интерфейса создается HTML-код с помощью вызова внутри функции aini_to_html соответствующей функции для

каждого из предусмотренных элементов интерфейса (листинг А.7). Каждая из данных функций на вход получает список частей строки элемента интерфейса. Полученные части строки будут использованы для генерации текста на языке HTML, описывающего соответствующий полученному aINI-коду элемент интерфейса. Код для каждого из элементов записан заранее, но атрибуты будут взяты из полученной строки из файла с входными aINI-данными (например, значение по умолчанию, название формы ввода, текст рядом и т.п.). Для некоторых элементов интерфейса (окно для ввода текста, флажок, выбор файла и т. п.) функция создания HTML-кода возвращает код для функции представления, который далее будет выведен в ранее упомянутый файл txt_for_views.

Сгенерированные функции представления (например, листинг А.8) получают данные из форм ввода, записанные после взаимодействия с пользователем, после чего с помощью функции find производится поиск строки элемента формы ввода в файле, описывающем всю страницу, с которой взаимодействовал пользователь. Новые значения записываются заместо старых значений. В случае элемента выбора файла, выбранный пользователем файл сохраняется в папку, где находятся файлы с входными данными.

Листинг. 23. Парсер

```
rus_alphas='ёйцукенгшщзхъфывапролджэячсмитьбюЁЙЦУКЕНГШЩЗХ
ЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ'

bool_var = Word('0' + '1')
word_num = Word(nums + alphas + '@')
variable = Word(nums + alphas + '_')
rus_eng_word_num = Word(alphas + rus_alphas + ' ' + nums)
link = Word(alphanums + '-' + '+' + '=' + '/' + '.' + '_'
+ '#' + ':' + '&' + '?' + '%')
text = Word(printables + rus_alphas + ' ')
menu_link = Word(alphanums + '-' + '+' + '=' + '.' + '_'
+ '#' + ':' + '&' + '?' + '%')

parse_section = '[' + variable + ']' + '//' +
rus_eng_word_num
```

```
parse_text_box = variable + '=' + word_num + '//' +
rus_eng_word_num
parse_box = variable + '=' + '[' + bool_var + ']' +
'{0|1}' + '//' + rus_eng_word_num
parse_file_selection = variable + '=' + '[file]' + '//' +
rus_eng_word_num
parse_link = '[' + link + ']' + '//' + rus_eng_word_num
parse_text = '//' + OneOrMore(text)
aini_file_name = menu_link + '=' + '[' + variable + ']' +
'//' + rus_eng_word_num
```

Листинг. 24. Функция parsing

```
def parsing(s):
    try:
        test = parse_section.parseString(s).asList()
        patern_id = 1
    except ParseException:
        try:
            test = parse_text_box.parseString(s).asList()
            patern_id = 2
        except ParseException:
            try:
                test = parse_box.parseString(s).asList()
                patern_id = 3
            except ParseException:
                try:
                    test =
parse_file_selection.parseString(s).asList()
                    patern_id = 4
                except ParseException:
                    try:
                        test =
parse_link.parseString(s).asList()
                        patern_id = 5
                    except ParseException:
                        try:
                            test =
parse_text.parseString(s).asList()
                            patern_id = 6
                        except ParseException:
                            print('The string is not
parsed:' + s + '')
                            quit()
    return [patern_id, test]
```

4 Тестирование

Было произведено тестирование web-приложения с сгенерированным интерфейсом, элементами списка urlpatterns (листинг 25) и функциями представления (листинг А.8). Примеры главного меню и одного из трех HTML-файлов представлены в листингах А.4 и А.5 соответственно, они были сгенерированы на основе aINI-кода из листингов 24 и 25.

Листинг. 25. Пример файла txt_for_urls

```
#Ниже представлен код, который должен быть вставлен в
файл urls.py в список urlpatterns

    path("", menu, name="menu"),
    path("F1/", input1, name="input1"),
    path("F2/", input2, name="input2"),
    path("F3/", input3, name="input3"),

#Также вставьте следующую строку в файл urls.py до списка
urlpatterns
from upload.views import menu, input1, input2, input3
```

После запуска web-приложения, открывается главное меню (рис. 19). На данной странице можно выбрать один из трех ранее описанных вариантов.

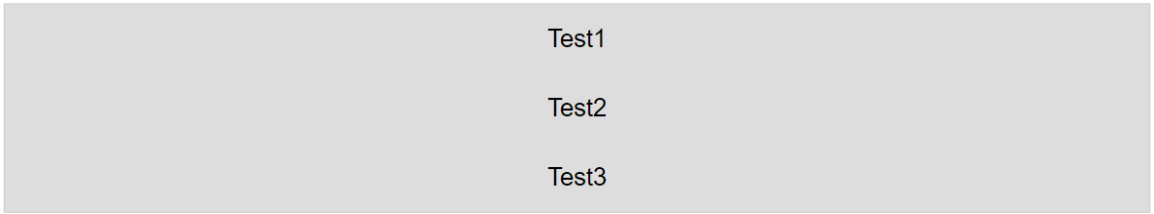

Рис. 19. Главное меню

При выборе первого варианта Test1 происходит переход по ссылке http://195.19.40.68:8084/F1/ и открывается страница показанная на рис. 20, в которой уже присутствуют некоторые значения по умолчанию. Можно

переключаться между вкладками данной страницы, что показано на рис. 21 (в данном случае вкладки 2).

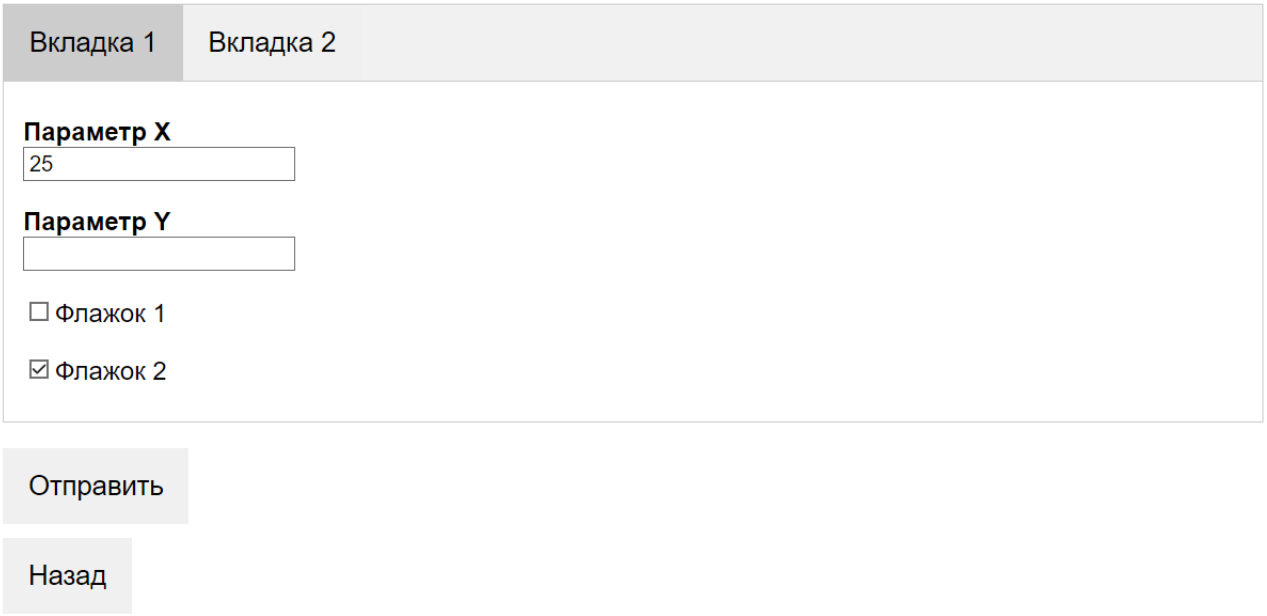

Рис. 20. Страница после перехода по первой ссылке

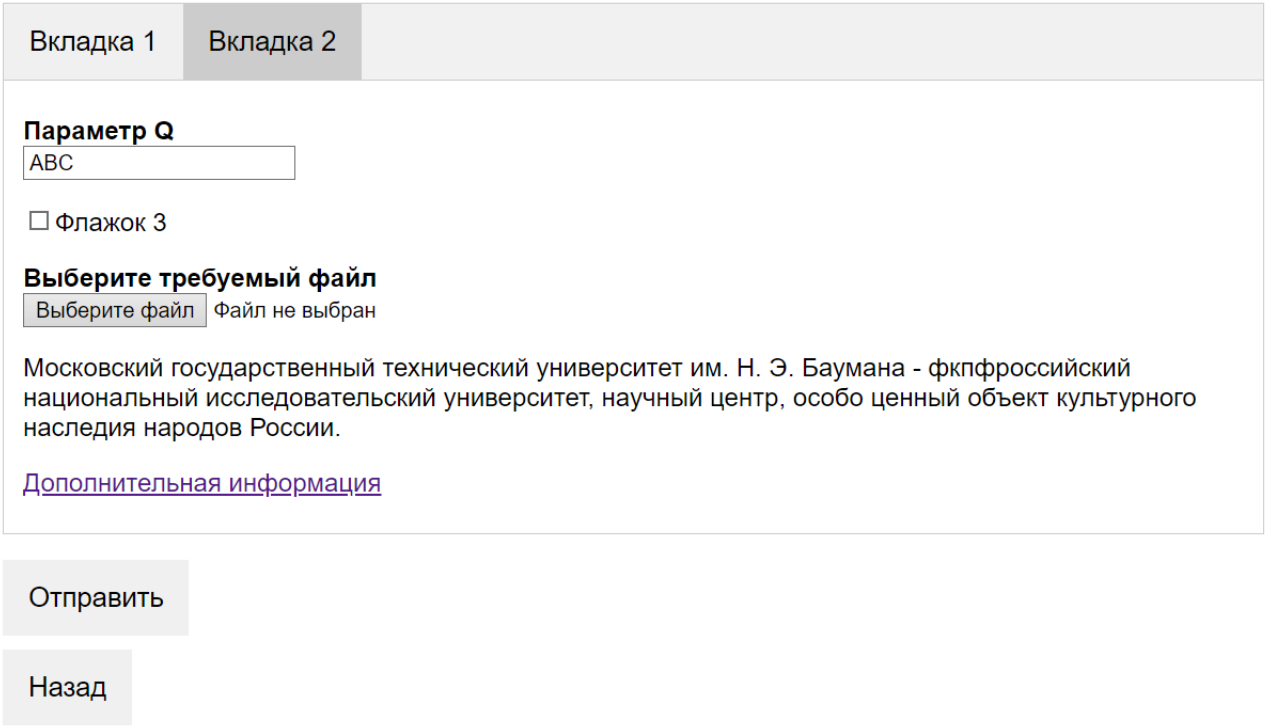

Рис. 21. Демонстрация перехода на вторую вкладку

Были введены значения в поля ввода, нажаты некоторые флажки, и выбран файл (рисунки 22 и 23).

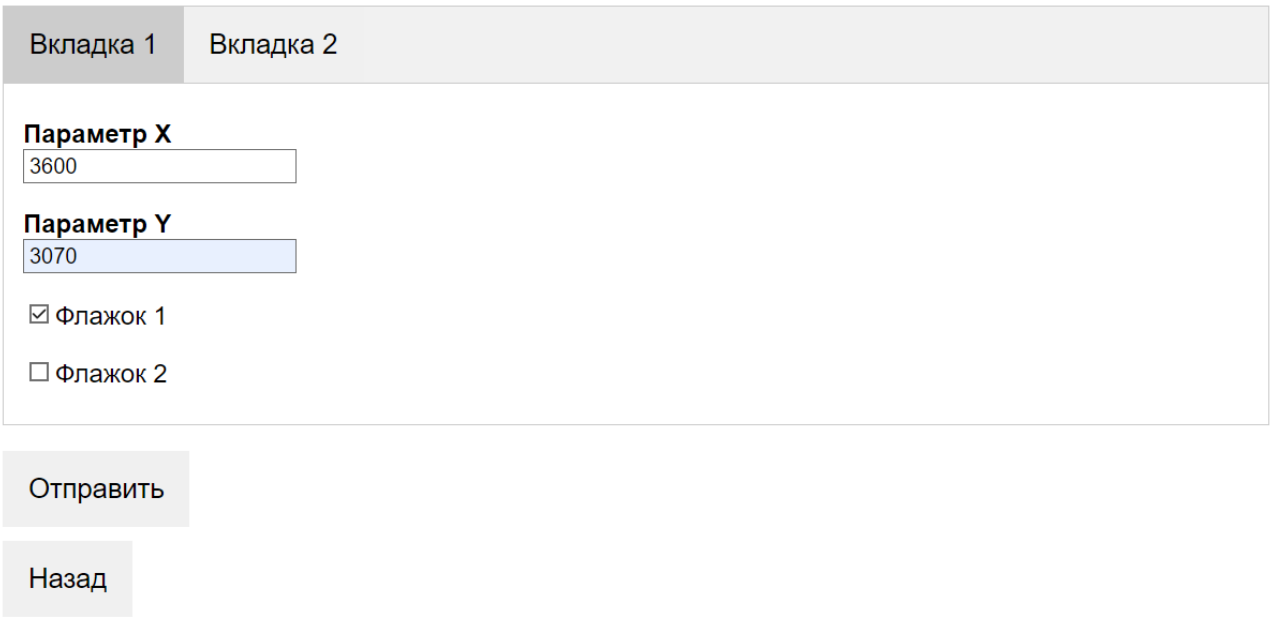

Рис. 22. Первая вкладка страницы с новыми значениями

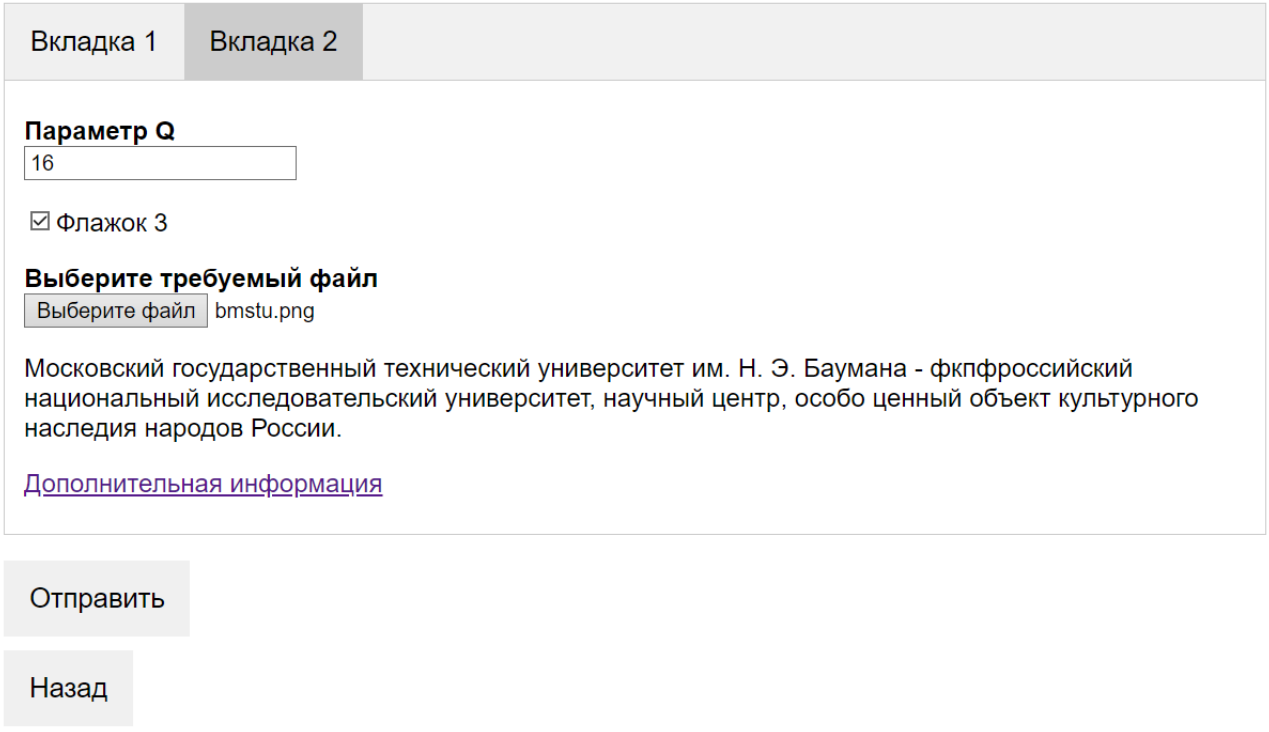

Рис. 23. Вторая вкладка страницы с новыми значениями

После нажатия кнопки «Отправить» значения были записаны в изначальный aINI файл, что можно увидеть на листинге 26, а выбранный файл был сохранен в папку mediafiles в докере (рис. 24), откуда его можно загрузить в

папку, находящуюся у хоста с помощью команды “ `docker cp app-web-1:/home/app/web/input/bmstu.png D:\PyCharm\django-on-docker`”.

Листинг. 26. Скорректированный файл с aINI данными страницы

```
[sec1]//Вкладка 1
x=3600//Параметр X
y=3070//Параметр Y
box1=[1]{0|1}//Флажок 1
box2=[0]{0|1}//Флажок 2
[sec2]//Вкладка 2
q=16//Параметр Q
box3=[1]{0|1}//Флажок 3
ParametersFile=[file]//Выберите требуемый файл
//Московский государственный технический университет им. Н. Э. Баумана - фкпфроссийский национальный исследовательский университет, научный центр, особо ценный объект культурного наследия народов России.
[https://bmstu.ru/about]//Дополнительная информация
```

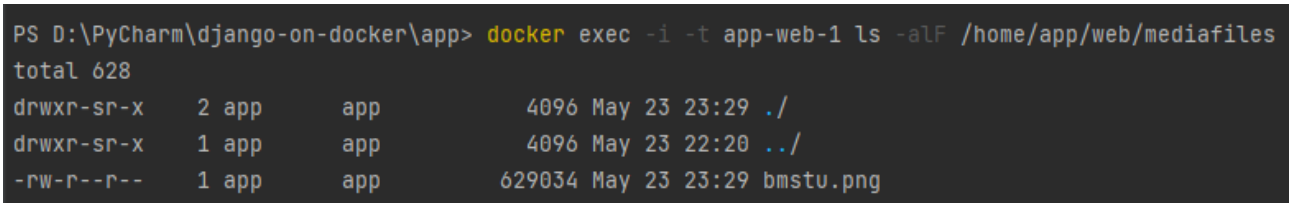

Рис. 24. Демонстрация присутствия в докере сохранённого файла

ЗАКЛЮЧЕНИЕ

- В результате обзора литературы было подтверждена необходимость программного обеспечения для построения интерфейса инженерного программного обеспечения.
- Для проверки работоспособности предложенного подхода применялось web-приложение, разработанное с использованием Django и Docker.
- Разработано программное обеспечение для преобразования файлов в формате aINI в файлы формата HTML. Указанное программное обеспечение автоматически формулирует исходных код на языке Python, при интерпретации которого реализуется сохранение данных, введенных пользователем.
- Web-приложение запущенно на сервере с использованием интерфейса, сгенерированного разработанной программой.
- Автоматизированное построение GUI на основе данных в формате с простым синтаксисом (например, aINI) позволяет быстро создавать графические формы ввода.
- Разработанное web-ориентированное программное обеспечение, в том числе удобно тем, что формат ввода и вывода стандартизированы, что обеспечивает доступность результата генерации автору исходного файла в формате aINI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лукьянов Д.В., (2012) // «Разработка графического пользовательского интерфейса». Новые информационные технологии в автоматизированных системах. URL: https://cyberleninka.ru/article/n/razrabotka-graficheskogo-polzovatelskogo-interfeysa
2. Соколов А.П., (2020) // «Описание формата данных aINI (advanced INI)».
3. Санковский Ю.Е., (1998) // «Метод построения оконного интерфейса пользователя на основе моделирования пользовательских целей». URL: https://tekhnosfera.com/metod-postroeniya-okonnogo-interfeysa-polzovatelya-na-osnove-modelirovaniya-polzovatelskih-tseley
4. Казаков Г.В., Корянов В.В., Чемирисов В.В., Уваров А.В., (2020) // «Методический подход к созданию универсального пользовательского интерфейса». Наука и инновации. № 11. doi: 10.18698/2308-6033- 2020-11-2034
5. Юркин В.А., Сараджишвили С.Э., (2020) // «Построение пользовательского интерфейса с использованием интерактивного машинного обучения» СОВРЕМЕННАЯ НАУКА: АКТУАЛЬНЫЕ ПРОБЛЕМЫ ТЕОРИИ И ПРАКТИКИ. № 3. URL: http://www.nauteh-journal.ru/files/e94b8829-1ce0-4abb-bf6b-f08d3635a0db
6. Django введение. [Электронный ресурс] // MDN Web Docs — URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/Introduction. (Дата обращения 24.10.2022).
7. Что такое Docker и как его использовать в разработке. [Электронный ресурс] // Eternalhost — URL: \url{https://eternalhost.net/blog/razrabotka/chto-takoe-docker?utm_source=google.com&utm_medium=organic&utm_campaign=google.com&utm_referrer=google.com. (Дата обращения 24.10.2022).

8. File storage API [Электронный ресурс] // Django Documentation — URL: https://docs.djangoproject.com/en/4.2/ref/files/storage/. (Дата обращения 03.05.2023)
9. Pyparsing. [Электронный ресурс] // Xgu.ru — URL: http://xgu.ru/wiki/pyparsing#:~:text=%5Bправить%5D%20Pyparsing%20—%20модуль%20синтаксического,достаточно%20большое%20количество%20синтаксических%20анализаторов. (Дата обращения 28.02.2023).
10. Errors and Exceptions. [Электронный ресурс] // Python Documentation — URL: https://docs.python.org/3/tutorial/errors.html. (Дата обращения 28.02.2023).

ПРИЛОЖЕНИЕ А

Листинги выпускной квалификационной работы

Листинг А.1. Dockerfile.prod

```
# BUILDER
FROM python:3.9.6-alpine as builder

# установка рабочей директории
WORKDIR /usr/src/app

# установка переменных окружения
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# установите зависимости psycopg2
RUN apk update \
    && apk add postgresql-dev gcc python3-dev musl-dev

RUN pip install --upgrade pip
RUN pip install flake8
COPY . .
#RUN flake8 --ignore=E501,F401 .

# установка зависимостей
COPY ./requirements.txt .
RUN pip wheel --no-cache-dir --no-deps --wheel-dir
/usr/src/app/wheels -r requirements.txt

# FINAL

FROM python:3.9.6-alpine

# создание каталога для пользователя app
RUN mkdir -p /home/app

# создание пользователя app
RUN addgroup -S app && adduser -S app -G app

# создание необходимых директорий
ENV HOME=/home/app
ENV APP_HOME=/home/app/web
RUN mkdir $APP_HOME
RUN mkdir $APP_HOME/staticfiles
RUN mkdir $APP_HOME/mediafiles
```

```
WORKDIR $APP_HOME

# установка зависимостей
RUN apk update && apk add libpq
COPY --from=builder /usr/src/app/wheels /wheels
COPY --from=builder /usr/src/app/requirements.txt .
RUN pip install --no-cache /wheels/*

# копирование entrypoint.prod.sh
COPY ./entrypoint.prod.sh .
RUN sed -i 's/\r$//g'  $APP_HOME/entrypoint.prod.sh
RUN chmod +x  $APP_HOME/entrypoint.prod.sh

# копирование проекта в контейнер
COPY . $APP_HOME

# chown  файлам пользователя
RUN chown -R app:app $APP_HOME

# переход к пользователю app
USER app

# запуск entrypoint.prod.sh
ENTRYPOINT ["/home/app/web/entrypoint.prod.sh"]
```

Листинг. А.2. CSS стили

```
<style>
body {font-family: Arial;}

.tab {
  overflow: hidden;
  border: 1px solid #ccc;
  background-color: #f1f1f1;
}

.tab button, .clickable {
  float: left;
border: none;
  outline: none;
  cursor: pointer;
  padding: 14px 16px;
  transition: 0.3s;
```

```
  font-size: 17px;
}

.tab button:hover, .clickable:hover {
  background-color: #ddd;
}

.tab button.active {
  background-color: #ccc;
}

.tabcontent {
  display: none;
  padding: 6px 12px;
  border: 1px solid #ccc;
  border-top: none;
}
</style>
```

Листинг А.3. Функция main

```
    def main():
        input_f = open(r'input\config', encoding='utf-8')
        f_menu    =    open(r'output\menu.html',    'w',
encoding='utf-8')
        f_urls  =  open(r'output\txt_for_urls.txt',  'w',
encoding='utf-8')
        f_views = open(r'output\txt_for_views.txt', 'w',
encoding='utf-8')
        list_of_functions  =  'from  upload.views  import
menu'

        f_urls.write('#Ниже  представлен  код,  который
должен  быть  вставлен  в  файл  urls.py  в  список
urlpatterns\n\n\tpath('
                     '"", menu, name="menu"),\n')
        f_views.write(
            '#Ниже представлен код, который должен быть
вставлен в файл views.py\n\n\ndef menu(request):\n\treturn
render('
            'request, "menu.html")\n\n')
        f_menu.write(
            '<!DOCTYPE                          html>\n<html
lang="en">\n<head>\n<meta charset="UTF-8">\n<style>\nbody
{font-family: '
```

```
            'Arial;}\n\n.tab                {\n\toverflow:
hidden;\n\tborder:  1px  solid  #ccc;\n\tbackground-color:
#f1f1f1;\n}\n\n.tab '
            'button {\n\toverflow: hidden;\n\tborder: 1px
solid #ccc;\n\tbackground-color: #ddd;\n\tfloat: '
            'left;\n\tborder:           none;\n\toutline:
none;\n\tcursor:        pointer;\n\tpadding:        14px
16px;\n\ttransition: '
            '0.3s;\n\tfont-size: '
            '17px;\n\tdisplay:           block;\n\twidth:
100%;\n}\n\n.tab button:hover {\n\tbackground-color: '

'#808080;\n}\n\n</style>\n<title>menu</title>\n</head>\n<
body>\n<h1>Здравствуйте! Выберите один из '
            'предложенных           вариантов.</h1>\n<div
class="tab">\n')

        for line in input_f:
            global num_of_section
            num_of_section = -1

            try:
                config                                  =
aini_file_name.parseString(line).asList()
            except ParseException:
                print('The string is not parsed:')
                print(line)
                quit()

            f_menu.write('\t<button
onclick="document.location=\'/' + config[0] + '/\'">' +
config[6] + '</button>\n')
            f_urls.write('\tpath("' + config[0] + '/", '
+ config[3] + ', name="' + config[3] + '"),\n')
            list_of_functions = list_of_functions + ', '
+ config[3]
            list_of_variables  =  aini_to_html(config[3],
config[6], )
            f_views.write('\ndef   '   +   config[3]   +
'(request):\n\tif request.method == "POST":\n')
            for variable in list_of_variables:
                f_views.write(variable)
            f_views.write('\t\treturn  render(request,  "'
+  config[3]  +  '.html")\n\treturn  render(request,  "'  +
config[
```

```
                3] + '.html")\n\n')

        f_menu.write('</div>\n</body>\n</html>\n')
        f_urls.write('\n#Также вставьте следующую строку в файл urls.py до списка urlpatterns\n' + list_of_functions)

        input_f.close()
        f_menu.close()
        f_urls.close()
        f_views.close()
```

Листинг. А.4. menu.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<style>
body {font-family: Arial;}

.tab {
    overflow: hidden;
    border: 1px solid #ccc;
    background-color: #f1f1f1;
}

.tab button {
    overflow: hidden;
    border: 1px solid #ccc;
    background-color: #ddd;
    float: left;
    border: none;
    outline: none;
    cursor: pointer;
    padding: 14px 16px;
    transition: 0.3s;
    font-size: 17px;
    display: block;
    width: 100%;
}

.tab button:hover {
    background-color: #808080;
}
```

```
</style>
<title>menu</title>
</head>
<body>
<h1>Здравствуйте! Выберите один из предложенных вариантов.</h1>
<div class="tab">
    <button onclick="document.location='/F1/'">Test1</button>
    <button onclick="document.location='/F2/'">Test2</button>
    <button onclick="document.location='/F3/'">Test3</button>
</div>
</body>
</html>
```

Листинг. А.5. Пример HTML-файла для одной из страниц

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {font-family: Arial;}

.tab {
    overflow: hidden;
    border: 1px solid #ccc;
    background-color: #f1f1f1;
}

.tab button, .clickable {
    float: left;
border: none;
    outline: none;
    cursor: pointer;
    padding: 14px 16px;
    transition: 0.3s;
    font-size: 17px;
}

.tab button:hover, .clickable:hover {
    background-color: #ddd;
}
```

```
.tab button.active {
    background-color: #ccc;
}

.tabcontent {
    display: none;
    padding: 6px 12px;
    border: 1px solid #ccc;
    border-top: none;
}
</style>
<title>Test1</title>
</head>
<body>

<div class="tab">
    <button class="tablinks"
onclick="tabs(event,'section_0')"
id="defaultOpen">Вкладка 1</button>
    <button class="tablinks"
onclick="tabs(event,'section_1')">Вкладка 2</button>
</div>

<form method="post" enctype="multipart/form-data">

{% csrf_token %}
<div id="section_0" class="tabcontent">
    <p><b>Параметр X</b><br>
    <input type="text" name="x" value="25"></p>
    <p><b>Параметр Y</b><br>
    <input type="text" name="y" value=""></p>
    <p><input type="checkbox" name="box1"
value="box1">Флажок 1</p>
    <p><input type="checkbox" name="box2" value="box2"
checked>Флажок 2</p>
</div>

<div id="section_1" class="tabcontent">
    <p><b>Параметр Q</b><br>
    <input type="text" name="q" value="ABC"></p>
    <p><input type="checkbox" name="box3"
value="box3">Флажок 3</p>
    <p><b>Выберите требуемый файл</b><br>
    <input type="file" name="ParametersFile"></p>
```

```
    <p>Московский государственный технический университет
им. Н. Э. Баумана - фкпфроссийский национальный
исследовательский университет, научный центр, особо
ценный объект культурного наследия народов России.</p>
    <p><a href="https://bmstu.ru/about">Дополнительная
информация</a></p>
</div>

<p><input class="clickable" type="submit"></p>
</form>
<br>
<br>
<br>
<button class="clickable"
onclick="document.location='/'">Назад</button>

<script>
function tabs(evt, tab_id) {
    var i, tabcontent, tablinks;
    tabcontent =
document.getElementsByClassName("tabcontent");
    for (i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = "none";
    }
    tablinks =
document.getElementsByClassName("tablinks");
    for (i = 0; i < tablinks.length; i++) {
        tablinks[i].className =
tablinks[i].className.replace(" active", "");
    }
    document.getElementById(tab_id).style.display =
"block";
    evt.currentTarget.className += " active";
}

document.getElementById("defaultOpen").click();
</script>

</body>
</html>
```

Листинг. А.6. Функция aini_to_html

```
def aini_to_html(file_name, title):
    sections_list = []
```

```
    elements_list = []
    list_of_variables = []

    input_f = open('input/' + file_name, encoding='utf-
8')
    f = html_start(title, file_name)

    for line in input_f:
        id_and_line = parsing(line)
        patern_id = id_and_line[0]
        parsed_line = id_and_line[1]
        print(parsed_line)
        if patern_id <= 3:
            match patern_id:
                case 1:
                    html_code =
section_to_html(parsed_line)
                    sections_list.append(html_code)
                case 2:
                    html_code =
textbox_to_html(parsed_line, file_name)
                    if len(elements_list) ==
num_of_section + 1:
                        elements_list[num_of_section] +=
html_code[1]
                    else:

elements_list.append(html_code[1])

list_of_variables.append(html_code[0])
                case 3:
                    html_code = box_to_html(parsed_line,
file_name)
                    if len(elements_list) ==
num_of_section + 1:
                        elements_list[num_of_section] +=
html_code[1]
                    else:

elements_list.append(html_code[1])

list_of_variables.append(html_code[0])
        else:
            match patern_id:
                case 4:
```

```
                    html_code =
file_selection_to_html(parsed_line)
                    if len(elements_list) ==
num_of_section + 1:
                        elements_list[num_of_section] +=
html_code[1]
                    else:

elements_list.append(html_code[1])

list_of_variables.append(html_code[0])
                case 5:
                    html_code = link_to_html(parsed_line)
                    if len(elements_list) ==
num_of_section + 1:
                        elements_list[num_of_section] +=
html_code
                    else:
                        elements_list.append(html_code)
                case 6:
                    html_code = text_to_html(parsed_line)
                    if len(elements_list) ==
num_of_section + 1:
                        elements_list[num_of_section] +=
html_code
                    else:
                        elements_list.append(html_code)

    for section in sections_list:
        f.write(section)
    f.write('</div>\n\n')

    i = 0
    f.write('<form method="post" enctype="multipart/form-
data">\n\n{% csrf_token %}\n')
    for element in elements_list:
        id = 'section_' + str(i)
        f.write('<div id="' + id + '"
class="tabcontent">\n' + element + '</div>\n\n')
        i += 1
    html_finish(f)
    return list_of_variables
```

Листинг. А.7. Функции для записи HTML-кода

```
def html_start(title, html_file_name):
    folder = 'output\\'
    f = open(folder + html_file_name + '.html', 'w',
encoding='utf-8')
    f.write('<!DOCTYPE html>\n<html>\n<head>\n<meta
name="viewport" content="width=device-width, '
            'initial-scale=1">\n<style>\nbody {font-
family: Arial;}\n\n.tab {\n\toverflow: hidden;\n\tborder:
1px '
            'solid #ccc;\n\tbackground-color:
#f1f1f1;\n}\n\n.tab button, .clickable {\n\tfloat:
left;\nborder: '
            'none;\n\toutline: none;\n\tcursor:
pointer;\n\tpadding: 14px 16px;\n\ttransition:
0.3s;\n\tfont-size: '
            '17px;\n}\n\n.tab
button:hover, .clickable:hover {\n\tbackground-color:
#ddd;\n}\n\n.tab '
            'button.active {\n\tbackground-color:
#ccc;\n}\n\n.tabcontent {\n\tdisplay: none;\n\tpadding:
6px '
            '12px;\n\tborder: 1px solid #ccc;\n\tborder-
top: none;\n}\n</style>\n<title>' + title +
            '</title>\n</head>\n<body>\n \n<div
class="tab">\n')
    return f


def section_to_html(parsed_line):
    if len(parsed_line) != 5:
        section = parsed_line[1]
    else:
        section = parsed_line[4]
    global num_of_section
    num_of_section += 1
    id = "'section_" + str(num_of_section) + "'"
    if num_of_section == 0:
        return '\t<button class="tablinks"
onclick="tabs(event,' + id + ')" id="defaultOpen">' +
section + '</button>\n'
    else:
        return '\t<button class="tablinks"
onclick="tabs(event,' + id + ')">' + section +
'</button>\n'
```

```
def textbox_to_html(parsed_line, file_name):
    html_code = []
    if len(parsed_line) != 5:
        textbox = parsed_line[0]
        html_code.append('\t\tif request.POST["' +
parsed_line[0] + '"]:\n\t\t\t' + parsed_line[0] + ' =
request.POST["'
                         + parsed_line[0] +
'"]\n\t\t\tnew_data = \'\'\n\t\t\tinput =
open(\'/home/app/web/input/'
                         + file_name + '\', \'r\',
encoding=\'utf-8\')\n\t\t\tinput_f =
File(input)\n\t\t\told_data = '

'input_f.readlines()\n\t\t\tfor l in
old_data:\n\t\t\t\tif l.find(\'\\n' +
                         parsed_line[0] + '=\') != -
1:\n\t\t\t\t\tl = \'' + parsed_line[0] + '=\' + ' +
parsed_line[0] +
                         ' + \'//' + parsed_line[0] +
'\\n\'\n\t\t\t\tnew_data = new_data +
str(l)\n\t\t\toutput = '

'open(\'/home/app/web/input/' + file_name +
                         '\', \'w\', encoding=\'utf-
8\')\n\t\t\toutput_f =
File(output)\n\t\t\toutput_f.write('

'new_data)\n\t\t\tinput_f.close()\n\t\t\tinput.close()\n\
t\t\toutput_f.close('
                         ')\n\t\t\toutput.close()\n\n')
    else:
        textbox = parsed_line[4]
        html_code.append('\t\tif request.POST["' +
parsed_line[0] + '"]:\n\t\t\t' + parsed_line[0] + ' =
request.POST["'
                         + parsed_line[0] +
'"]\n\t\t\tnew_data = \'\'\n\t\t\tinput =
open(\'/home/app/web/input/'
                         + file_name + '\', \'r\',
encoding=\'utf-8\')\n\t\t\tinput_f =
File(input)\n\t\t\told_data = '
```

```
'input_f.readlines()\n\t\t\tfor l in
old_data:\n\t\t\t\tif l.find(\'//' +
                            parsed_line[4] + '\\n\') != -
1:\n\t\t\t\t\tl = \'' + parsed_line[0] + '=\' + ' +
parsed_line[0]
                            + ' + \'//' + parsed_line[4] +
'\\n\'\n\t\t\t\tnew_data = new_data +
str(l)\n\t\t\toutput = '

'open(\'/home/app/web/input/' + file_name +
                            '\', \'w\', encoding=\'utf-
8\')\n\t\t\toutput_f =
File(output)\n\t\t\toutput_f.write('

'new_data)\n\t\t\tinput_f.close()\n\t\t\tinput.close()\n\
t\t\toutput_f.close('
                            ')\n\t\t\toutput.close()\n\n')
    print(parsed_line)

    if parsed_line[2] == '@' + parsed_line[0] + '@':
        value = ''
    else:
        value = parsed_line[2]
    html_code.append('\t<p><b>' + textbox +
'</b><br>\n\t<input type="text" name="' + parsed_line[0]
+ '" value="'
                     + value + '"></p>\n')

    return html_code


def box_to_html(parsed_line, file_name):
    html_code = []
    if len(parsed_line) != 8:
        box = parsed_line[0]
        html_code.append('\t\tif "' + parsed_line[0] + '"
in request.POST:\n\t\t\t' + parsed_line[0] +
                            ' = \'1\'\n\t\telse:\n\t\t\t' +
parsed_line[0] +
                            ' = \'0\'\n\t\tnew_data =
\'\'\n\t\tinput = open(\'/home/app/web/input/' +
file_name +
                            '\', \'r\', encoding=\'utf-
8\')\n\t\tinput_f = File(input)\n\t\told_data = '
```

```
                            'input_f.readlines()\n\t\tfor l
in old_data:\n\t\t\tif l.find(\'\\n' + parsed_line[0] +
                            '=\') != -1:\n\t\t\t\tl = \'' +
parsed_line[0] + '=[\' + ' + parsed_line[0] + ' +
\']{0|1}//' +
                            parsed_line[0] +
'\\n\'\n\t\t\tnew_data = new_data + str(l)\n\t\toutput =
open('

'\'/home/app/web/input/' + file_name + '\', \'w\', '

'encoding=\'utf-8\')\n\t\toutput_f = '

'File(output)\n\t\toutput_f.write('

'new_data)\n\t\tinput_f.close('

')\n\t\tinput.close('

')\n\t\toutput_f.close('

')\n\t\toutput.close()\n\n')
    else:
        box = parsed_line[7]
        html_code.append('\t\tif "' + parsed_line[0] + '"
in request.POST:\n\t\t\t' + parsed_line[0] +
                            ' = \'1\'\n\t\telse:\n\t\t\t' +
parsed_line[0] + '= \'0\'\n\t\tnew_data = \'\'\n\t\tinput
= '

'open(\'/home/app/web/input/' + file_name +
                            '\', \'r\', encoding=\'utf-
8\')\n\t\tinput_f = File(input)\n\t\told_data = '
                            'input_f.readlines()\n\t\tfor l
in old_data:\n\t\t\tif l.find(\'//' + parsed_line[7] +
                            '\\n\') != -1:\n\t\t\t\tl = \''
+ parsed_line[0] + '=[\' + ' + parsed_line[0] + ' +
\']{0|1}//'
                            + parsed_line[7] +
'\\n\'\n\t\t\tnew_data = new_data + str(l)\n\t\toutput =
open('

'\'/home/app/web/input/' + file_name + '\', \'w\', '

'encoding=\'utf-8\')\n\t\toutput_f '
```

```
'= File('

'output)\n\t\toutput_f.write('

'new_data)\n\t\tinput_f.close('

')\n\t\tinput.close('

')\n\t\toutput_f.close('

')\n\t\toutput.close()\n\n')
    if parsed_line[3] == '1':
        html_code.append('\t<p><input type="checkbox"
name="' + parsed_line[0] + '" value="' + parsed_line[0]
                         + '" checked>' + box + '</p>\n')
    else:
        html_code.append('\t<p><input type="checkbox"
name="' + parsed_line[0] + '" value="' + parsed_line[0] +
'">'
                         + box + '</p>\n')
    return html_code


def file_selection_to_html(parsed_line):
    html_code = []
    if len(parsed_line) != 5:
        file_selection = parsed_line[0]
    else:
        file_selection = parsed_line[4]
    html_code.append('\t\tif "' + parsed_line[0] + '" in
request.FILES:\n\t\t\t' + parsed_line[0] + ' =
request.FILES["'
                     + parsed_line[0] + '"]\n\t\t\tfs =
FileSystemStorage()\n\t\t\tfilename = fs.save(' +
parsed_line[0]
                     + '.name, ' + parsed_line[0] +
')\n\t\t\tfile_url = fs.url(filename)\n\n')
    html_code.append('\t<p><b>' + file_selection +
'</b><br>\n\t<input type="file" name="'
                     + parsed_line[0] + '"></p>\n')
    return html_code


def link_to_html(parsed_line):
```

```
    if len(parsed_line) != 5:
        link_title = parsed_line[1]
    else:
        link_title = parsed_line[4]
    return '\t<p><a href="' + parsed_line[1] + '">' +
link_title + '</a></p>\n'


def text_to_html(parsed_line):
    return '\t<p>' + parsed_line[1] + '</p>\n'


def html_finish(f):
    f.write('<p><input class="clickable"
type="submit"></p>\n</form>\n<br>\n<br>\n<br>\n<button
class="clickable" '

'onclick="document.location=\'/\'">Назад</button>\n\n'
            '<script>\nfunction tabs(evt, tab_id)
{\n\tvar i, tabcontent, tablinks;\n\ttabcontent = '

'document.getElementsByClassName("tabcontent");\n\tfor (i
= 0; i < tabcontent.length; i++) '
            '{\n\t\ttabcontent[i].style.display =
"none";\n\t}\n\ttablinks = '

'document.getElementsByClassName("tablinks");\n\tfor (i =
0; i < '
            'tablinks.length; i++)
{\n\t\ttablinks[i].className =
tablinks[i].className.replace(" active", '

'"");\n\t}\n\tdocument.getElementById(tab_id).style.displ
ay = "block";\n\tevt.currentTarget.className += " '

'active";\n}\n\ndocument.getElementById("defaultOpen").cl
ick();\n</script>\n\n</body>\n</html>')
    f.close()
```

Листинг. А.8. Пример файла txt_for_views

```
#Ниже представлен код, который должен быть вставлен в
файл views.py

def menu(request):
    return render(request, "menu.html")
```

```
def input1(request):
    if request.method == "POST":
        if request.POST["x"]:
            x = request.POST["x"]
            new_data = ''
            input = open('/home/app/web/input/input1',
'r', encoding='utf-8')
            input_f = File(input)
            old_data = input_f.readlines()
            for l in old_data:
                if l.find('//Параметр X\n') != -1:
                    l = 'x=' + x + '//Параметр X\n'
                new_data = new_data + str(l)
            output = open('/home/app/web/input/input1',
'w', encoding='utf-8')
            output_f = File(output)
            output_f.write(new_data)
            input_f.close()
            input.close()
            output_f.close()
            output.close()

        if request.POST["y"]:
            y = request.POST["y"]
            new_data = ''
            input = open('/home/app/web/input/input1',
'r', encoding='utf-8')
            input_f = File(input)
            old_data = input_f.readlines()
            for l in old_data:
                if l.find('//Параметр Y\n') != -1:
                    l = 'y=' + y + '//Параметр Y\n'
                new_data = new_data + str(l)
            output = open('/home/app/web/input/input1',
'w', encoding='utf-8')
            output_f = File(output)
            output_f.write(new_data)
            input_f.close()
            input.close()
            output_f.close()
            output.close()

        if "box1" in request.POST:
```

```
            box1 = '1'
        else:
            box1= '0'
        new_data = ''
        input = open('/home/app/web/input/input1', 'r',
encoding='utf-8')
        input_f = File(input)
        old_data = input_f.readlines()
        for l in old_data:
            if l.find('//Флажок 1\n') != -1:
                l = 'box1=[' + box1 + ']{0|1}//Флажок
1\n'
            new_data = new_data + str(l)
        output = open('/home/app/web/input/input1', 'w',
encoding='utf-8')
        output_f = File(output)
        output_f.write(new_data)
        input_f.close()
        input.close()
        output_f.close()
        output.close()

        if "box2" in request.POST:
            box2 = '1'
        else:
            box2= '0'
        new_data = ''
        input = open('/home/app/web/input/input1', 'r',
encoding='utf-8')
        input_f = File(input)
        old_data = input_f.readlines()
        for l in old_data:
            if l.find('//Флажок 2\n') != -1:
                l = 'box2=[' + box2 + ']{0|1}//Флажок
2\n'
            new_data = new_data + str(l)
        output = open('/home/app/web/input/input1', 'w',
encoding='utf-8')
        output_f = File(output)
        output_f.write(new_data)
        input_f.close()
        input.close()
        output_f.close()
        output.close()
```

```
        if "box3" in request.POST:
            box3 = '1'
        else:
            box3= '0'
        new_data = ''
        input = open('/home/app/web/input/input1', 'r',
encoding='utf-8')
        input_f = File(input)
        old_data = input_f.readlines()
        for l in old_data:
            if l.find('//Флажок 3\n') != -1:
                l = 'box3=[' + box3 + ']{0|1}//Флажок
3\n'
            new_data = new_data + str(l)
        output = open('/home/app/web/input/input1', 'w',
encoding='utf-8')
        output_f = File(output)
        output_f.write(new_data)
        input_f.close()
        input.close()
        output_f.close()
        output.close()

        if "ParametersFile" in request.FILES:
            ParametersFile =
request.FILES["ParametersFile"]
            fs = FileSystemStorage()
            filename = fs.save(ParametersFile.name,
ParametersFile)
            file_url = fs.url(filename)

        return render(request, "input1.html")
    return render(request, "input1.html")
…
```

ПРИЛОЖЕНИЕ Б

Графическая часть выпускной квалификационной работы

В приложении Б представлены графические материалы, среди которых:

1. Графический пользовательский интерфейс
2. Цель и задачи работы
3. Подходы к разработке пользовательских интерфейсов
4. Подходы к разработке пользовательских интерфейсов
5. Постановка задачи
6. Назначение разрабатываемого ПО
7. Особенности генерации GUI на основе aINI
8. Особенности генерации GUI на основе aINI
9. Разработка тестового web-приложения
10. Особенности генерации интерфейса с применением Django
11. Примеры генерации GUI на основе aINI
12. Примеры генерации GUI на основе aINI
13. Выводы
14. Дальнейшие перспективы развития