Министерство просвещения РФ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»

Институт математики, физики, информатики и технологий

Кафедра информатики, информационных технологий
и методики обучения информатике

ТЕХНОЛОГИЯ РАЗРАБОТКИ СЕРВИСА ПРОЕКТИРОВАНИЯ ФУНКЦИОНАЛЬНОЙ МОДЕЛИ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы ИСиТ–1801
Института математики, физики,
информатики и технологий
Гонаго И.А.

Работа допущена к защите

«____»_____ 2022 г.

Зав. кафедрой _____

Руководитель: кандидат педагогических наук,
доцент кафедры ИИТ и МОИ
Кудрявцев А.В.

Екатеринбург – 2022

РЕФЕРАТ

ГОНАГО И.А. ТЕХНОЛОГИЯ РАЗРАБОТКИ СЕРВИСА ПРОЕКТИРОВАНИЯ ФУНКЦИОНАЛЬНОЙ МОДЕЛИ ИНФОРМАЦИОННОЙ СИСТЕМЫ, выпускная квалификационная работа: 74 стр., рис. 25, библ. 30 назв., приложений 1.

Ключевые слова: ФУНКЦИОНАЛЬАНАЯ МОДЕЛЬ, ВЕБ-СЕРВИС, ПРОЕКТИРОВАНИЕ, БИЗНЕС-ПРОЦЕСС, IDEF0.

Объект разработки – программное средство графического описания функциональной модели ИС.

Цель работы – описать технологию процесса проектирования функциональной модели ИС и разработать сервис для её графического описания.

В работе описаны технология проектирования функциональной модели ИС, а также результаты проектирования и реализации сервиса графического описания функциональной модели ИС. Последний необходим при проектировании любого программного продукта независимо от масштаба и выбора средств разработки. Сервис реализован в виде веб-приложения. Реализация велась средствами языка разметки гипертекста HTML, языка описания стилей и скриптового языка JavaScript. Ядро функционала сервиса реализована при помощи переработанной библиотеки JsPlumb и библиотеки JQuery.

Система была развернута на локальном сервере и протестирована. По результатам тестирования можно сделать вывод, что система отвечает заявленным требованиям.

Введение

Создание современных информационных систем (далее ИС) является сложнейшей задачей, для решения которой требуется применение специальных инструментов и методик. При создании систем небольшого масштаба над проектом могут работать всего несколько человек, и как правило, разработчик модели выступает в роли аналитика и проектировщика. При создании масштабных корпоративных ИС работает группа, состоящая из аналитиков, проектировщиков и разработчиков. На этапе анализа аналитики изучают предметную область, в процессе которой разрабатывается функциональная модель предприятия (модель бизнес-процессов). Задача проектировщиков — исходя из полученной функциональной модели разработать структуру базы данных, а также ее логическую и физическую модель.

Главная цель разработки функциональной модели процесса – точное определение и спецификация всех его функций, существующих в рамках процесса другого, более высокого, уровня иерархии, и описание характера взаимосвязей между ними. Подобная модель обеспечивает исчерпывающее представление о функционировании исследуемого процесса, а также представление обо всех информационных и материальных потоках, имеющих в нем место. Функциональная модель даёт возможность грамотного разграничения ресурсов между отдельными операциями описываемого бизнес-процесса и позволяет оценить эффект от их использования. Из описанного выше становится понятно – грамотно описанная функциональная модель является ключом к созданию качественного продукта. За последние 10–15 лет спрос на инструментальные «CASE-средства» и «CASE-технологии» серьезно вырос, что неудивительно, ведь они позволяют максимально автоматизировать и систематизировать все этапы разработки ПО, которое становится всё сложнее и многообразнее.

Актуальность работы обусловлена актуальностью самого подхода к проектированию систем с использованием метода разработки функциональной модели. И заключается в создании сервиса для решения задачи проектирования функциональной модели ИС, а именно предоставление инструмента для описания (моделирования) контекста системы.

Цель ВКР: описать технологию процесса проектирования функциональной модели ИС и разработать сервис для её графического описания.

Объектами исследования выступают: технология проектирования и реализация программного средства графического описания функциональной модели ИС.

Задачи ВКР:

1. Дать определения основным понятиям технологии процесса проектирования функциональной модели;
2. Сформировать требования к функциональной модели;
3. Рассмотреть теоретические основы технологии процесса проектирования функциональной модели;
4. Сформировать требования к разрабатываемому сервису проектирования функциональной модели ИС;
5. Описать используемые технологии для разработки сервиса;
6. Реализовать MVP (минимально жизнеспособный продукт) описанного сервиса.

Глава 1. Теоретические основы

1.1 Описание методологий и обзор существующих case - средств для проектирования ИС.

Технология проектирования ИС – совокупность методологии, а также средств для организации процесса проектирования (управление процессом разработки и модернизации проекта).

Подход проектирования (метод) – алгоритм реализации определенной стадии создания ИС. Основной принцип построения любой системы – принцип иерархической декомпозиции, включающий две группы методологий:

1. Структурно-функциональные методологии, в основу которых заложен принцип функциональной декомпозиции: структура системы описывается в терминах иерархии ее функций и передачи информации между отдельными функциональными элементами.
2. В основе объектно-ориентированной методологии лежит использование объектной декомпозиции. Предметная область представляется в виде совокупности функциональных элементов (объектов), обменивающихся входными воздействиями (сообщениями) в процессе выполнения программы.

Структурно-функциональная методология: в качестве средств структурного анализа и проектирования, наиболее распространены следующие нотации:

- SADT (Structured Analysis and Design Technique) - ныне IDEF. Используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающие эти функции. Для новых систем IDEF применяется с целью определения требований для разработки системы, реализующей выделенные функции. Для существующих - IDEF0 может быть использована для анализа функций, выполняемых системой.

В нотации IDEF0 модель представляет собой взаимосвязанную и упорядоченных иерархию диаграмм для описания функциональной модели ИС.

После общего описания системы следует функциональная декомпозиция - разбиение на составные фрагменты в нотациях:

- DFD (Data Flow Diagrams) или диаграммы потоков данных. Данные диаграммы обычно строятся для наглядного изображения текущей работы системы документооборота организации. Как правило, их используют в качестве дополнения модели бизнес-процессов, выполненной в IDEF0.
- IDEF3. Методология позволяет описать процессы, фокусируя внимание на течении этих процессов, позволяет рассмотреть конкретный процесс с учетом последовательности выполняемых операций.
- ER (Entity-Relationship Diagrams) или диаграммы "сущность-связь". Методология описания данных (IDEF1X).

Универсальные стандартизированные языки графического описания (моделирования) позволяют обеспечить логическую целостность и полноту описания, которая особо необходима на этапе анализа, для достижения точных и непротиворечивых результатов. Результатом проектирования модели в нотации IDEF0 является модель, состоящая из диаграмм, текста и глоссария, ссылающихся друг на друга.

Объектно-ориентированная методология: принципиальное отличие здесь кроется в способе декомпозиции системы. Данный подход применяет объектную декомпозицию, в которой поведение системы описывается терминами обмена сообщениями между объектами, а статическая структура описывается

терминами объектов и связей между ними. Каждый объект системы обладает индивидуальным поведением, отражающим поведение реального объекта.

Объектно-ориентированная методология (далее ООМ) так же, как и структурная методология, создавалась с целью формализации процесса создания масштабных программных комплексов, тем самым позволив уменьшить их затраты, сложность и время на разработку. ООМ и структурная методология преследуют одни цели, но решают их с разных отправных точек. ООМ чаще всего позволяет управлять более сложными проектами, в отличие от структурной методологии. В ОО-системе алгоритмы (поведение) и структуры данных (внутреннее устройство) объединены в объекты, это уменьшает сложность системы и локализует изменения.

Концептуальной основой объектно-ориентированного подхода выступает объектная модель, которая строится с учетом следующих принципов:

- абстрагирование – это выделение существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов, таким образом четко определяя его концептуальные границы с точки зрения дальнейшего анализа, и игнорируя менее важные или незначительные детали.
- инкапсуляция – физическая локализация свойств и поведения в рамках одной абстракции (рассматриваемой как «черный ящик»), скрывающая их реализацию за общедоступным интерфейсом.
- модульность – это свойство, связанное с возможностью ее декомпозиции на ряд внутренне сильно сцепленных, но слабо связанных между собой подсистем (модулей).
- иерархия – это ранжированная по уровням система абстракций.

Основными понятиями объектно-ориентированного подхода являются объект и класс:

- Объект– предмет или явление, имеющее четко определенное поведение и обладающее индивидуальностью. Из структуры и поведения похожих объектов определяется общий для них класс. Класс представляет из себя множество объектов, которые связаны общностью структуры и поведения. Следующими важными понятиями объектного подхода являются наследование и полиморфизм. Полиморфизм можно интерпретировать как способность класса принадлежать более чем одному типу.
- Наследование– это процесс построение новых классов, на основе существующих с возможностью переопределения или добавления данных и методов. Объекты одного класса имеют одинаковые операции и атрибуты, но значения их атрибутов могут быть разными. Состояние объекта характеризуется значениями его атрибутов, а поведение – его операциями. При вызове операций, которые могут изменить значение атрибутов объекта, возможен переход объекта из одного состояния к другому. Для осуществления функций системы между объектами устанавливаются связи, по которым они обмениваются сообщениями.

UML (Unified Modeling Language) – стандартная нотация визуального моделирования программных систем. UML предоставляет средства для создания визуальных моделей, которые однозначно интерпретируются всеми разработчиками, вовлеченными в проект, и являются средством коммуникации в рамках проекта.

Диаграмма в UML — это графическое представление набора элементов. Диаграммы рисуют для визуализации системы с разных точек зрения. Визуальное моделирование на UML позволяет использовать восемь видов диаграмм, каждая из которых может содержать элементы определенного типа.

В языке UML для этапов анализа предназначены следующие виды диаграмм:

- «use case» - (диаграммы прецедентов);
- «activity» - (диаграммы описаний технологий, процессов, функций);
- «sequence» - (диаграммы последовательностей действий);
- «collaboration» - (диаграммы взаимодействий).

Для этапа проектирования однозначно определены диаграммы:

- «state» (диаграммы состояний);
- «class «(диаграммы классов);
- «component» (диаграммы компонент);
- «deployment» (диаграммы развертывания).

Главной диаграммой в UML выступает «Class diagram», которая является основой для генерации кода и целью проектирования. Это визуальное представление идеи объектно-ориентированного проектирования. На данный момент объектный подход стал очень популярен и характеризуется разработчиками как универсальное средство проектирования. Большим плюсом является простота исправления проекта в соответствии с изменившейся логикой бизнес-процесса, так как в динамической модели совсем необязательна полная перестройка, что характерно для нотаций структурного подхода. Изменение отдельных классов и связей между ними не затрагивает общей концепции модели.

Несмотря на все плюсы, методология для применения унифицированного языка моделирования (UML) на этапах анализа и проектирования описана достаточно слабо (т. е. можно найти описание диаграмм, однако логика использования регламентируется довольно слабо), так что еще рано говорить о UML как о «панацее». На этапе описания контекста системы гораздо лучше подходит нотация IDEF0 структурно-функциональной методологии. Здесь еще на этапе черного ящика формулируются основные вопросы:

- «Какие функции осуществляет данная подсистема?»;
- «Что является входными данными?»;
- «Что является выходными данными?»;
- Что является управляющим компонентом?»;
- Что является механизмом?».

Поэтому лучшим вариантом для описания функциональной модели является проектирование в нотации IDEF0.

Приведем сравнение наиболее популярных case-средств, поддерживающих проектирование в нотации IDEF:

- ARIS Business Performance Edition (IDS Scheer AG);
- CA ERWin Process Modeler, ранее BPWin (CA);
- Business Studio.

- *ARIS Business PERFOMANCE Edition* – платформа, поддерживающая полный цикл управления бизнес-процессами: от описания стратегии до процесса контроллинга. Система позволяет моделировать бизнес-процессы, оптимизировать и публиковать их. Поддерживает следующий набор нотаций:

- IDEF;
- Basic Flowchart;
- EPC;
- BPMN;
- BPEL.

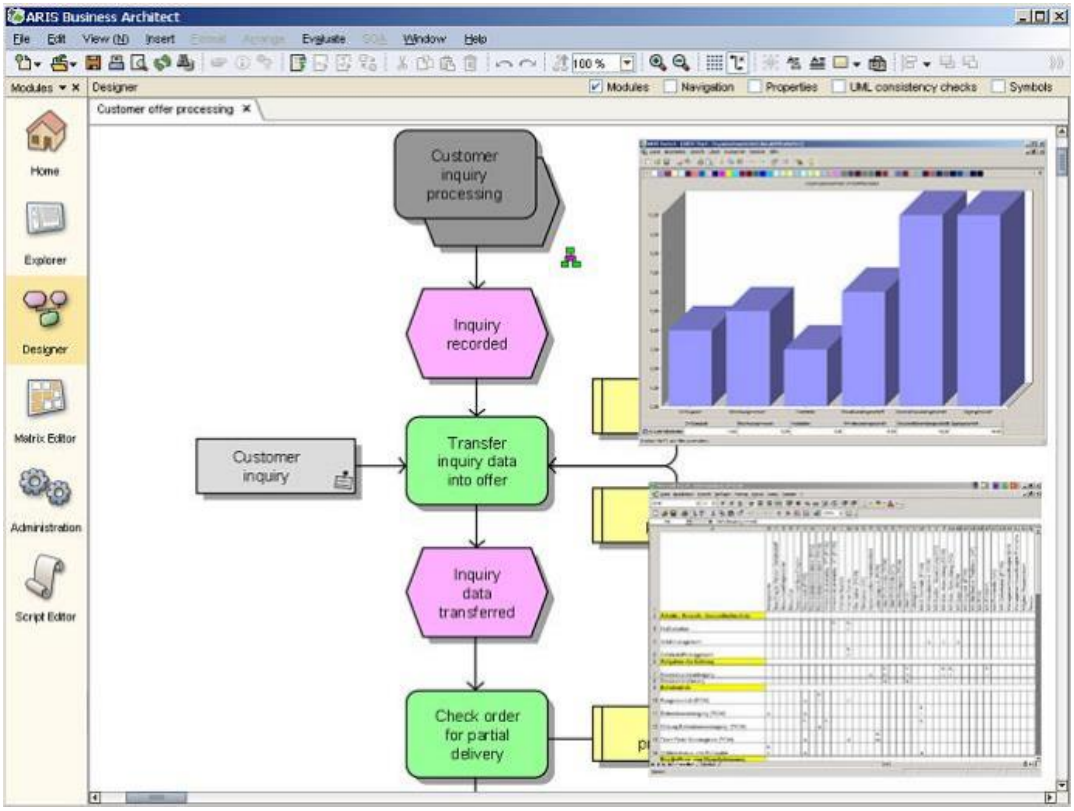


Рисунок 1 - ARIS Business PERFOMANCE Edition

Также имеется возможность создания собственных типов диаграмм и поддержка имитационного моделирования. Стоимость лицензии программного продукта порядка 2500€.

- CA ERWin Process *Modeler*- мощный инструмент моделирования, применяющийся для анализа, документирования и реорганизации сложных бизнес-процессов.

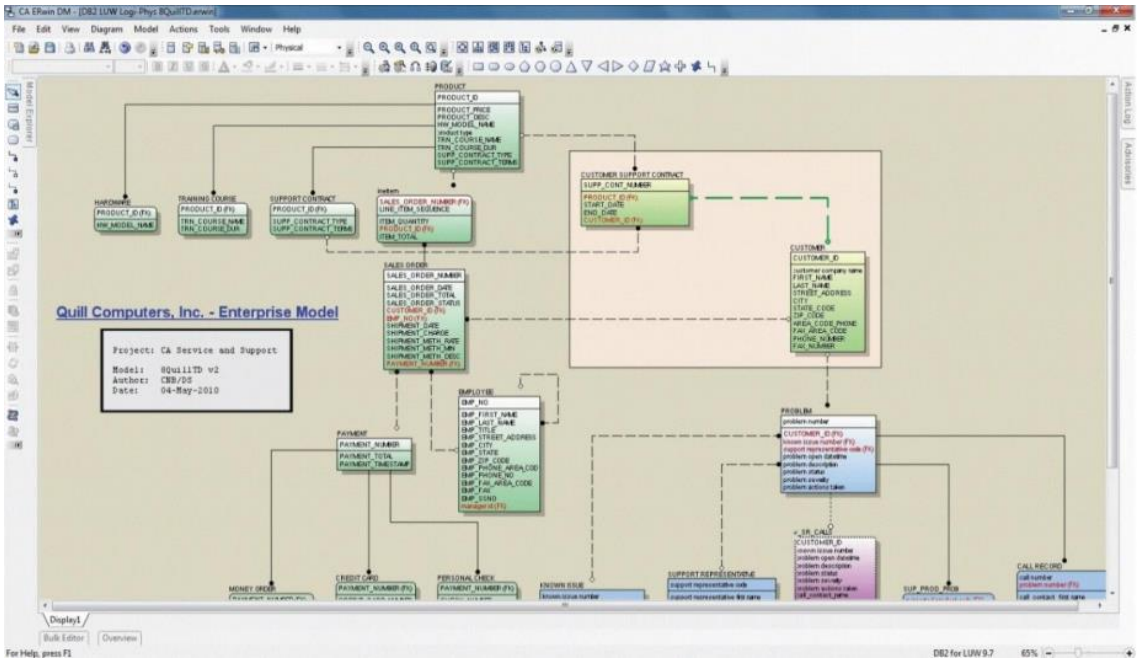


Рисунок 2 - CA ERWin Process Modeler

Process Modeler имеет хорошо развитую систему встроенной отчетности. Стоимость лицензии составляет от 75 000₽ до 135000₽ (зависит от срока и выбранного типа технического сопровождения).

- *Business Studio* – это отечественный программный продукт для моделирования бизнес-архитектур, основанный на Microsoft Visio. Это дает пользователю привычную и знакомую среду проектирования. Моделирование производится по стандартной схеме «сверху вниз». Сначала предлагается построение модели процессов верхнего уровня на основе нотации IDEF0. Для построения нижнего уровня нужно выбрать подходящую нотацию: EPC, Basic Flowchart, Cross Functional Flowchart или BPMN.

Программа включает в себя следующий набор нотаций:

- EPC;
- IDEF0;
- IDEF3;
- BPMN;
- Basic Flowchart.

Программа совместима с Microsoft Word, что даёт возможность генерировать отчеты в привычном формате. Стоимость достигает 76 000₽ (зависит от версии продукта). Имеется и возможность приобретения пробной версии (стоимость от 2 100₽ до 3 200₽ в месяц).

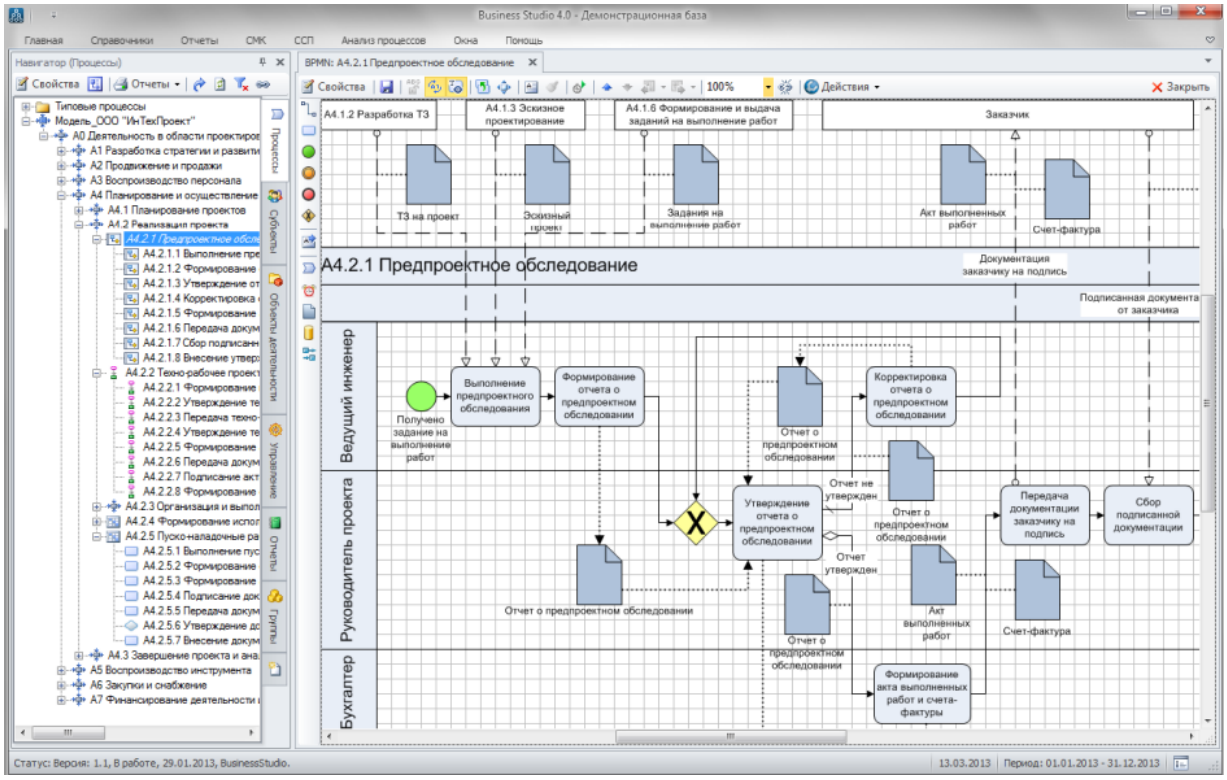


Рисунок 3 - Business Studio

Безусловно, представленные программы являются мощными средствами проектирования. Но у всех них есть ряд недостатков:

- необходимость непосредственно установки средства на ПК;
- отсутствие веб версий приложений;
- необходимость периодического обновления;
- запутанный нагруженный интерфейс приложения;
- высокая стоимость лицензии;
- нередко отсутствие русского языка интерфейса.

Так же, некоторые решения поставляются в составе других нагромождённых программных продуктов (например, диаграммы visual studio).

Это не всегда удобно, а отсутствие ознакомительных или учебных версий, как и русского языка интерфейса, может стать серьезной преградой, к примеру, для изучения моделирования процессов.

1.2 Основные понятия и алгоритм технологии проектирования функциональной модели ИС.

Информация – сведения (сообщения, данные) независимо от формы их представления.

Информационные технологии – процессы, методы поиска, сбора, хранения, обработки, предоставления, распространения информации и способы осуществления таких процессов и методов.

Проектирование информационных систем – это упорядоченная совокупность методологий и средств создания или модернизации информационных систем.

Информационная система – совокупность содержащейся в базах данных информации и обеспечивающих ее обработку информационных технологий и технических средств.

Бизнес-процесс – это совокупность процессов (операций, действий) и взаимодействий между ними, результатом которой является продукция и/или услуги, поставляемые потребителям, а входами – материальные, информационные и трудовые ресурсы, поставляемые внешними поставщиками.

Нотации – это определенные способы представления элементов информационной системы.

Реинжиниринг бизнес-процессов – это фундаментальная реорганизация бизнес-процессов с целью повышения их эффективности.

Системный подход – процесс рассмотрения любой системы в качестве совокупности взаимосвязанных элементов.

Процессный подход – представление любой системы в качестве совокупности процессов.

Функциональный подход – предусматривает четкое закрепление за каждой структурной единицей набора функций. Техническое задание – документ, используемый заказчиком в качестве средства для описания и определения задач, выполняемых при реализации договора.

Модель данных – это система организации данных и управления ими.

Методология проектирования информационных систем – это совокупность принципов проектирования (моделирования), выраженная в определенной концепции.

Перед описанием алгоритма проектирования функциональной модели ИС, нельзя не сказать о жизненном цикле ИС.

Жизненный цикл информационных системы – развитие рассматриваемой системы во времени, начиная от момента возникновения идеи о создании ИС и до момента вывода ее из эксплуатации.

Модель жизненного цикла – структурная основа процессов и действий, относящиеся к жизненному циклу, которая служит в качестве общей ссылки для установления связей и взаимопонимания сторон.

Стадии ЖЦ ИС:

1. Планирование и анализ требований (предпроектная стадия) ─ системный анализ. Производится исследование и анализ существующей ИС, определяются требования к создаваемой ИС и техническое задание (ТЗ) на разработку ИС;
2. Проектирование (техническое и логическое). Исходя из требований формируются состав функций, подлежащих автоматизации (функциональная архитектура) и состав обеспечивающих подсистем (системная архитектура), производится оформление технического проекта ИС;

3. Реализация (рабочее и физическое проектирование, кодирование). Разработка и конфигурация программ, формирование и заполнение баз данных, формулировка рабочих инструкций и указаний для персонала, оформление рабочего проекта;
4. Внедрение (опытная эксплуатация) - комплексная отладка и тестирование подсистем ИС, обучение персонала, постепенное внедрение ИС в эксплуатацию по подразделениям организации, оформление акта приемо-сдаточных испытаний ИС;
5. Эксплуатация ИС (сопровождение, модернизация) - сбор статистики о функционировании ИС, исправление недочетов и ошибок, определение требований к модернизации ИС и ее выполнение (повторение стадий 2–5).

Ниже рассматривается основное содержание стадий и этапов ЖЦ ИС.

Системный анализ. Основными целями этапа являются:

- формулировка потребностей в новой ИС (определение всех недостатков существующей ИС);
- выбор направления и определение экономической обоснованности проектирования ИС.

Системный анализ ИС начинают с описания и анализа функционирования рассматриваемого объекта в соответствии с предъявляемыми требованиями. В результате этого этапа выявляются недостатки существующей ИС. На основе недостатков формулируется потребность совершенствования системы управления данным объектом и задача экономического обоснования необходимости автоматизации определенных функций (технико-экономическое обоснование проекта ИС).

После определения потребности следует выбор направлений совершенствования объекта на основании подбора программно-технических средств. Результат оформляется в качестве ТЗ проекта, которое отражает

технические требования и условия к ИС, а также перечень ограничений на ресурсы проектирования.

Этап проектирования предполагает:

- проектирование функциональной архитектуры ИС, которая отражает структуру выполняемых функций;
- проектирование системной архитектуры ИС (состав обеспечивающих подсистем);
- реализацию проекта.

Формирование функциональной архитектуры, представляющая собой совокупность функциональных подсистем и связей между, ними с точки зрения качества всей последующей разработки ИС, является наиболее ответственным и важным этапом

Построение системной архитектуры на основе функциональной подразумевает определение элементов и модулей информационного, технического, программного обеспечения, других обеспечивающих подсистем, связей по информации и управлению между выделенными элементами и разработку технологии обработки информации.

Реализация включает разработку программ и инструкций для конечных пользователей, создание информационного обеспечения, включая наполнение баз данных. Внедрение разработанного проекта разделяется на опытное и промышленное.

Этап внедрения подразумевает под собой проверку работоспособности всех модулей и элементов разработанного проекта, а также устранение ошибок на уровне элементов и уровне связей между ними. Этап передачи в промышленную эксплуатацию - организация проверки уровня функций проекта и контроля на соответствие требованиям, определенным на стадии системного анализа.

Важной особенностью жизненного цикла ИС является повторяемость: "системный анализ ─ разработка ─ сопровождение ─ системный анализ". Это полностью соответствует представлению об ИС как о развивающейся динамической системе. При первом выполнении стадии "Разработка" создается проект ИС, а при последующих реализациях данной стадии происходит модификация проекта, направленная на поддержание его в актуальном состоянии.

Модели ЖЦ претерпевали существенные изменения в аспекте реализации технологий проектирования ИС. Среди известных моделей жизненного цикла можно выделить следующие:

- каскадная модель (до 70-х) ─ последовательный строгий переход на следующий этап только по окончании работ на предыдущем;
- итерационная модель (70-80-е) ─ допускает итерационные возвраты на прошлые этапы;
- спиральная модель (80-90-е) ─ модель прототипирования, предполагающая постепенное масштабирование прототипа ИС.

В каскадной модели переход на следующий этап ознаменует полное завершения работ на предыдущем. Главное достоинство каскадной модели заключено в планировании времени осуществления этапов проекта и упорядочении хода конструирования.

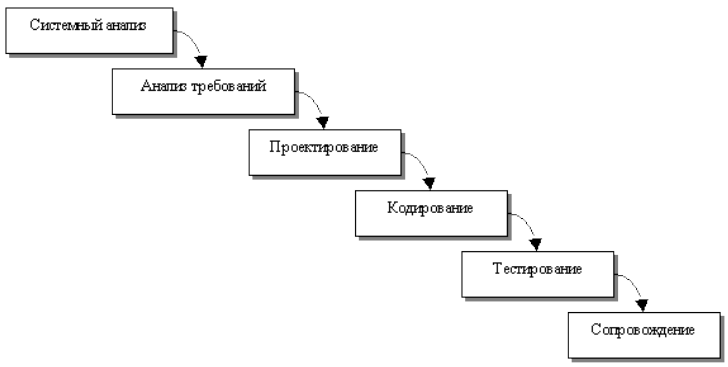


Рисунок 4 - Каскадная модель ЖЦ ИС

Недостатки модели:

- на деле реальные проекты почти всегда требуют отхождения от стандартной последовательности шагов (проявляется недостаточная гибкость модели);
- цикл основан на точных формулировках исходных требований к программному продукту (в реальности требования заказчика в начале проекта определены лишь частично);
- заказчик получает доступ к результатам разработки только в конце работы.

Итерационная модель. Стандартная итерационная модель позволяет осуществлять возвраты на предыдущие этапы разработки. В то же время — это обстоятельство является и существенным недостатком: заранее предвидеть все варианты использования системы, как правило, невозможно. Все подобные методологии по своей сути направлены на минимизацию возвратов. При возврате всегда приходится повторять построение того, что уже считалось готовым. В случае с методологиями, которые реализуют поддержку итерационного развития проектов, происходит отказ от завершенности фаз и этапов, вместо этого предлагается распределение наращивания функциональности и интерфейсных возможностей постепенно, по итерациям. В результате требование переделки старого при возвратах вполне можно ослабить.

Спиральная модель ─ отличный пример эволюционной стратегии конструирования.

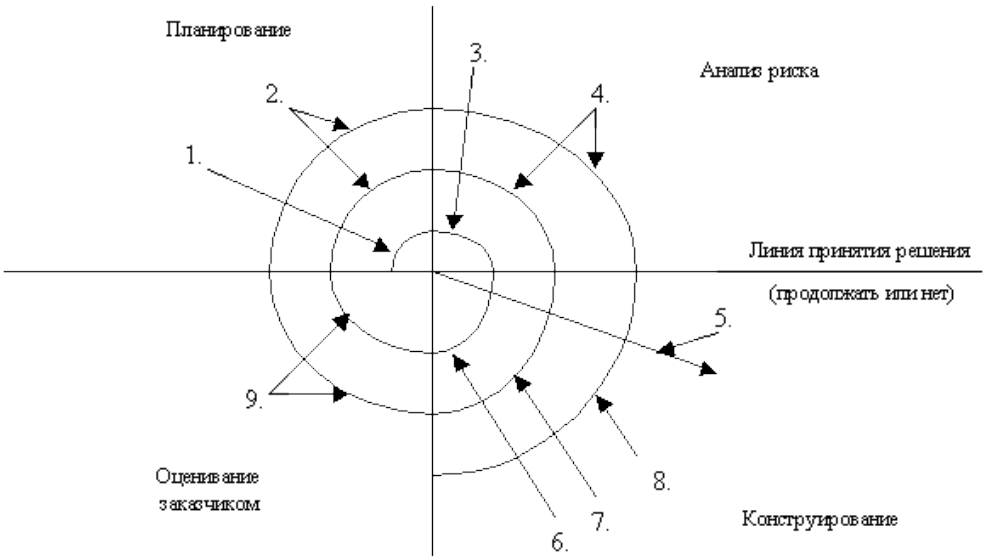


Рисунок 5 - Спиральная модель ЖЦ

1 - начальный сбор требований и планирование проекта;

2 – всё то же, но на основании рекомендаций заказчика;

3 - анализ риска исходя из начальных требований;

4 - анализ риска уже на основе реакции заказчика;

5 - переход к комплексной системе;

6 - начальный макет системы;

7 - следующий уровень макета;

8 - сконструированная система;

9 – оценка заказчиком.

Спиральная модель определяет четыре действия, представляемые четырьмя квадрантами спирали:

- планирование ─ определяются цели, варианты и ограничения;
- анализ риска ─ изучение вариантов и распознавание риска;
- конструирование ─ разработка продукта следующего уровня;

- оценивание — оценка заказчика текущих результатов конструирования.

С каждой итерацией по спирали строятся все более полные версии ПО.

Спиральная модель жизненного цикла ИС реально отображает разработку ПО; позволяет явно учитывать риск на каждом витке разработки; включает шаг системного подхода в итерационную структуру разработки; использует моделирование для уменьшения риска и совершенствования программного продукта.

Недостатками спиральной модели являются:

- повышенные требования к заказчику;
- трудность контроля и управления временем разработки.

Переходя к описанию алгоритма проектирования функциональной модели стоит еще раз вспомнить, что из себя представляет функциональное моделирование - это процесс моделирования функций, выполняемых рассматриваемой ИС/объектом, путем создания описательного структурированного графического изображения, показывающего, что, как и кем делается в рамках функционирования объекта и объектов, связывающих эти функции, с учетом имеющейся информации.

Процесс моделирования системы в нотации IDEF0 начинается с создания контекстной диаграммы — диаграммы наиболее абстрактного уровня описания системы в целом, содержащей определение субъекта моделирования, цели и точки зрения на модель. Под субъектом понимают саму систему, и при этом важно точно определить, что входит в систему и что лежит за пределами. Затем определить, что будет в дальнейшем рассматриваться в качестве компонентов системы, а что как внешнее воздействие.

Обычно в первую очередь строится модель существующей организации работы — AS-IS (как есть). Анализ функциональной модели позволяет понять, где находятся наиболее слабые места, в чем будут состоять преимущества новых бизнес-процессов и насколько глубоким изменениям подвергнется существующая структура организации бизнеса. Найденные в модели AS-IS недостатки можно исправить при создании модели TO-BE (как будет) — модели новой организации бизнес-процессов. Только на основе модели TO-BE строится модель данных, прототип и затем окончательный вариант ИС.

Модель в нотации IDEF0 есть совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Модель может содержать четыре типа диаграмм:

- контекстную диаграмму (в каждой модели может быть только одна контекстная диаграмма);
- диаграммы декомпозиции;
- диаграммы дерева узлов;
- диаграммы только для экспозиции (FEO).

Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой самое общее описание системы и ее взаимодействия с внешней средой. После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции. После декомпозиции контекстной диаграммы осуществляется декомпозиция больших фрагментов системы на более мелкие. Это происходит до момента, пока не будет достигнут нужный уровень подробности описания.

Работы (Activity) обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Работы изображаются в виде прямоугольников. Все работы должны быть названы и определены (например, "Деятельность компании", "Прием заказа" и т. д.).

Рисунок 6 - Пример контекстной диаграммы

Диаграммы декомпозиции содержат в себе дочерние работы, которые имеют общую - родительскую.

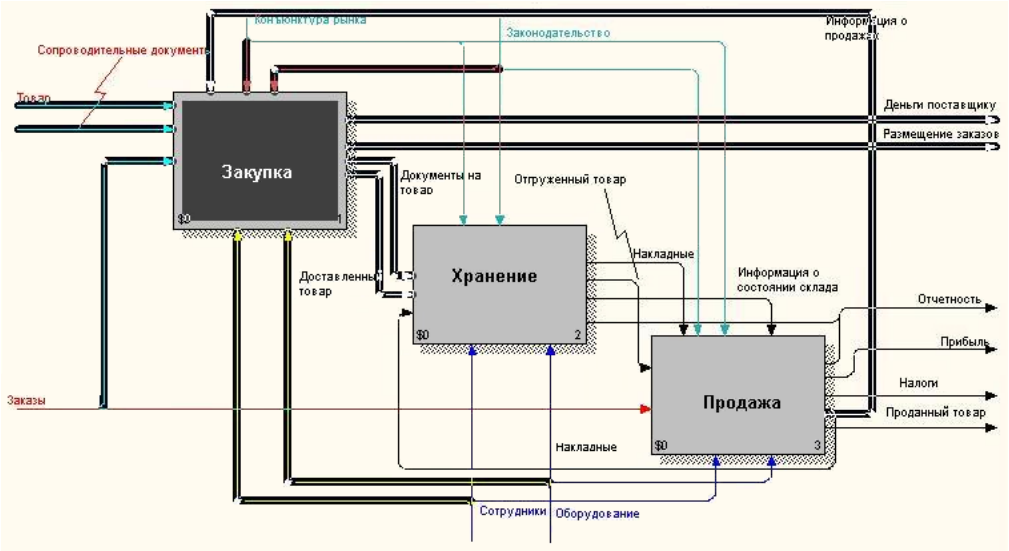


Рисунок 7 - Пример контекстной диаграммы

В IDEF0 различают пять типов стрелок:

- Вход (Input) — материальный объект или информация, которые используются или преобразуются работой для получения результата (выхода). Стрелка входа рисуется входящей в левую грань блока «работа».
- Управление (Control) — правила, стратегии, процедуры или стандарты, которыми руководствуется работа. У каждой работы должна иметься хотя бы одна стрелка управления. Стрелка управления рисуется как входящая в верхнюю грань работы. Управление влияет на работу, но не преобразуется работой. Если цель работы — изменить процедуру или стратегию, то такая процедура или стратегия будет для работы входом.
- Выход (Output) — предмет или информация, которые производятся работой. Каждая работа должна иметь хотя бы одну стрелку выхода. Стрелка выхода изображается исходящей из правой грани работы.
- Механизм (Mechanism) — ресурсы, которые выполняют работу, например, персонал предприятия, станки, устройства и т. д. Стрелка механизма рисуется входящей в нижнюю грань работы.

- Вызов (Call) — специальная стрелка, указывающая на другую модель работы. Стрелка вызова рисуется исходящей из нижней грани работы. Стрелка вызова используется для указания того, что некоторая работа выполняется за пределами моделируемой системы
- Граничные стрелки. Стрелки на контекстной диаграмме служат для описания взаимодействия системы с окружающим миром. Они могут начинаться у границы диаграммы и заканчиваться у работы, или наоборот.
- Обратная связь по управлению (output-control feedback), когда выход нижестоящей работы направляется на управление вышестоящей. Обратная связь по управлению часто свидетельствует об эффективности бизнес-процесса.

Так же в IDEF0 различают пять типов связей работ:

- связь по входу (input-output), когда выход вышестоящей работы направляется на вход следующей работы.
- связь по управлению (output-control), когда выход вышестоящей работы направляется на управление следующей работы. Связь показывает доминирование вышестоящей работы.
- обратная связь по входу (output-input feedback), когда выход нижестоящей работы направляется на вход вышестоящей. Используется для описания циклов.
- обратная связь по управлению (output-control feedback), когда выход нижестоящей работы направляется на управление вышестоящей. Является показателем эффективности бизнес-процесса.
- связь выход-механизм (output-mechanism), когда выход одной работы направляется на механизм другой и показывает, что работа подготавливает ресурсы для проведения другой работы.

Из перечисленных блоков, как из отдельных кирпичиков, строится диаграмма. После описания диаграммы функциональной модели строится диаграмма потоков данных, которая является следующим шагом проектирования ИС и строится уже по нотации другого уровня абстракции.

1.3 Формализация требований и техническое задание на разработку сервиса проектирования функциональной модели ИС.

Исходя из ранее описанных недостатков существующих программных средств проектирования был сформирован перечень требований к разрабатываемому сервису и выбрана модель ЖЦ.

Требования:

- реализация процесса моделирования в нотации IDEF0;
- интуитивно понятный и красивый интерфейс;
- возможность экспорта/импорта файла разрабатываемого проекта;
- кроссплатформенность;
- нетребовательность к ресурсам компьютера и скорость работы.

Определение и анализ требований позволили окончательно определиться с выбором типа будущего программного продукта. Оптимальным вариантом будет разработка веб – сервиса.

Огромное количество популярных программных продуктов реализованы в виде веб – приложений. Часть из них при этом имеет реализацию в виде классических приложений. Наличие веб приложения, если такой вариант реализации возможен в контексте разрабатываемого продукта, позволяет если не решить, то значительно упростить ряд задач как для разработчиков, так и для конечных пользователей, а именно:

- задача кроссплатформенности. При создании веб – приложения особое внимание уделяется интерфейсу приложения и особенностям работы различных браузеров. Интерфейс проектируют и реализуют так, чтобы приложение корректно

отображалось и функционировало в любом актуальном браузере на любом устройстве. В конечном счете получается приложение, которое не зависит от программно-аппаратной платформы конкретного устройства.

- задача распределения нагрузки и требовательности приложения к ресурсам. Данный момент выгоден как для пользователей, так и для разработчиков.

  С точки зрения конечного пользователя: как правило веб приложения не требовательны к аппаратным ресурсам, это позволяет пользоваться сервисами на устройствах со слабыми характеристиками, что понижает порог вхождения для пользователей, также им не нужно заморачиваться с поиском и установкой программного продукта. Это позволяет начать работу без промедлений и заминок.

  С точки зрения разработчиков: это дает возможность делегировать часть нагрузки на сторону на клиента, что позволяет снизить нагрузку на вычислительные мощности компании.
- задача обновления приложения. В случае с веб-приложением обновление происходит единовременно и централизовано для всех пользователей. При обновлении всем пользователям одновременно становится доступна новая версия сервиса.

Что касается ЖЦ ИС, наиболее подходящим и удобным вариантом будет модель прототипирования. Так как итогом разработки на каждом витке является прототип приложения с новым набором уточнений и правок.

Прототипирование позволяет создать минимально жизнеспособный вариант программного продукта до или в течение этапа составления требований к нему. Группа потенциальных пользователей работает с прототипом и определяет его

достоинства и недостатки. Результаты передаются разработчикам программного продукта. Так осуществляется обратная связь пользователей и разработчиков, она используется для корректировки требований к программному продукту. Результатом такой работы будет продукт, отражающий реальные потребности пользователей.

Разработка плана проекта является отправной точкой ЖЦ разработки программного продукта, после чего выполняется экспресс анализ, а далее разрабатываются БД, пользовательский интерфейс и осуществляется разработка требуемого функционала. Результатом проделанной работы служит документ, содержащий частичный набор требований к программному продукту. Данный документ будет являться основой итерационного цикла быстрого прототипирования.

Далее разработчик демонстрирует пользователям готовый прототип – результат процесса прототипирования, а пользователи оценивают его функционирование. После этого пользователи и разработчики совместно работают над устранением выявленных проблем. Этот процесс должен продолжаться до момента, пока программный продукт не начнет соответствовать поставленным перед ним требованиям. После прототип демонстрируют пользователям для получения предложений по его улучшению. Далее следует получение от пользователей одобрения (утверждения) функциональных возможностей представленного прототипа и производится окончательное преобразование в финальный программный продукт.

Модель прототипирования обладает целым рядом преимуществ:

1) Взаимодействие заказчика с разрабатываемой системой начинается на раннем этапе;

2) Благодаря реакции заказчика на прототип сводится к минимуму число неточностей в требованиях;

3) Снижается вероятность возникновения путаницы, искажения информации или недоразумений при определении требований к программному продукту, что приводит к созданию более качественного программного продукта;

4) В процессе разработки всегда можно учесть новые, даже неожиданные требования заказчика;

5) Прототип представляет собой формальную спецификацию, воплощенную в программный продукт;

6) Прототип позволяет очень гибко выполнять проектирование и разработку, включая несколько итераций на всех фазах жизненного цикла разработки;

7) Заказчик всегда видит прогресс в процессе разработки программного продукта;

8) Возможность возникновения противоречий между разработчиками и заказчиками сведена к минимуму;

9) Уменьшается число доработок, что снижает стоимость разработки. Возникающие проблемы решаются на ранних стадиях, что резко сокращает расходы на их устранение. Заказчики принимают участие в процессе разработки на протяжении всего жизненного цикла.

Ничто не совершенно, как и любой модели ей присущ ряд определенных недостатков:

1) Решение сложных задач вполне может затянуться и отодвигаться на будущее;

2) Всегда есть вероятность, что заказчик предпочтет получить именно прототип, а не законченный вариант продукта;

3) Перед началом работы невозможно сказать, как много итераций придется выполнить.

Модель прототипирования хорошо подходит для применения в следующих случаях:

- Требования к программному продукту определены неявно или заранее неизвестны;
- Есть необходимость в уточнении требований;
- Необходима проверка концепции;
- Имеется потребность в пользовательском интерфейсе;
- Неуверенность разработчиков в выборе решения.

Итогом разработки сервиса проектирования функциональной модели ИС будет являться минимально жизнеспособный вариант приложения (прототип).

1.4 Техническое задание:

1. Общие сведения.
   1.1. Название организации-заказчика: заказчиком в данном случае выступает ФГБОУ ВО «УрГПУ»;
   1.2. Название продукта разработки: сервис проектирования функциональной модели ИС;
   1.3. Назначение продукта: моделирование контекста систем в нотации IDEF0;
   1.4. Плановые сроки начала и окончания работ: 01.09.21-21.05.22;
2. Характеристика области применения продукта.
   2.1. Процессы и структуры, в которых предполагается использование продукта разработки: проектирование систем и бизнес-процессов, не ограниченные сферой деятельности организации, использующей сервис;
   2.2. Характеристика персонала: аналитики, проектировщики, разработчики - нижнего, среднего и высшего звена.
3. Требования к продукту разработки.
   3.1. Требования к продукту в целом:
      - реализация процесса моделирования в нотации IDEF0;
      - интуитивно понятный и красивый интерфейс;
      - возможность экспорта/импорта файла разрабатываемого проекта;
      - кроссплатформенность;
      - нетребовательность к ресурсам компьютера и скорость работы.

   3.2. Аппаратные требования: для работы нужен доступ в интернет и любой актуальный браузер. Примерные аппаратные требования к браузеру на сегодняшний день такие:
      - Процессор: Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2;

- Оперативная память: рекомендуется от 2Гб;
- Дисковое пространство: от 1Гб;
- Видеоадаптер: любой с поддержкой OpenGL.

3.3. Указание системного программного обеспечения: в качестве ОС может выступать любая система с графическим интерфейсом, поддерживающая работу современных браузеров: firefox, chrome, opera, safari и другие;

3.4. Указание программного обеспечения, используемого для реализации: сервис для разработки интерфейсов и прототипирования Figma, редактор исходного кода Visual studio code, инспектор браузера Chrome для мониторинга и дальнейшей отладки;

3.5. Форматы входных и выходных данных: входные данные представляют собой текст и графические примитивы для моделирования диаграмм. На выходе: файл проекта в формате json, либо графический файл формата jpeg;

3.6. Источники данных и порядок их ввода в систему, порядок вывода, хранения: графические примитивы для описания модели из панели инструментов вводятся в рабочую область в соответствии с предварительно продуманным алгоритмом процесса проектирования конкретной модели. Проект может быть сохранен на диск в формате jpeg (как конечный файл) или как файл проекта в формате json.

3.7. Меры защиты информации: Все элементы ввода на странице сервиса экранированы от ввода некорректных данных и обрабатываются скриптами. В дальнейшем необходим перенос проекта с локального сервиса на хостинг с поддержкой расширенного протокола https (протокол передачи гипертекста http + криптографический протокол tls), а значит весь трафик надежно защищен от прослушивания. Так же планируется подключение с системы зашиты от DDOS атак;

4. Требования к пользовательскому интерфейсу.

4.1. Общая характеристика пользовательского интерфейса: веб – сервис представлен графическим интуитивно понятным интерфейсом;

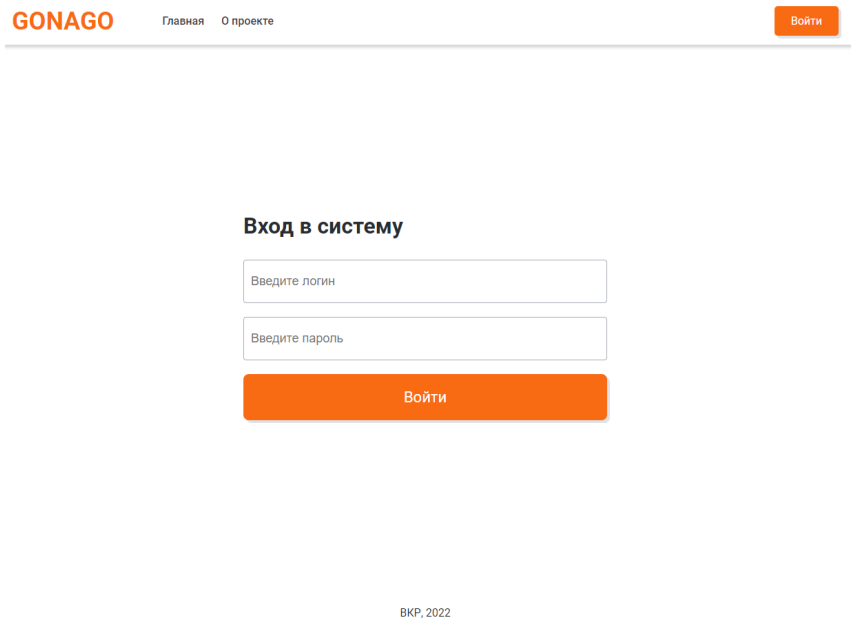


Рисунок 8 - Окно входа в систему

4.2. Размещение информации на экране, дизайн экрана: экран разделен на две области – рабочую и область панели инструментов. Так же на странице сервиса размещено небольшое функциональное меню.

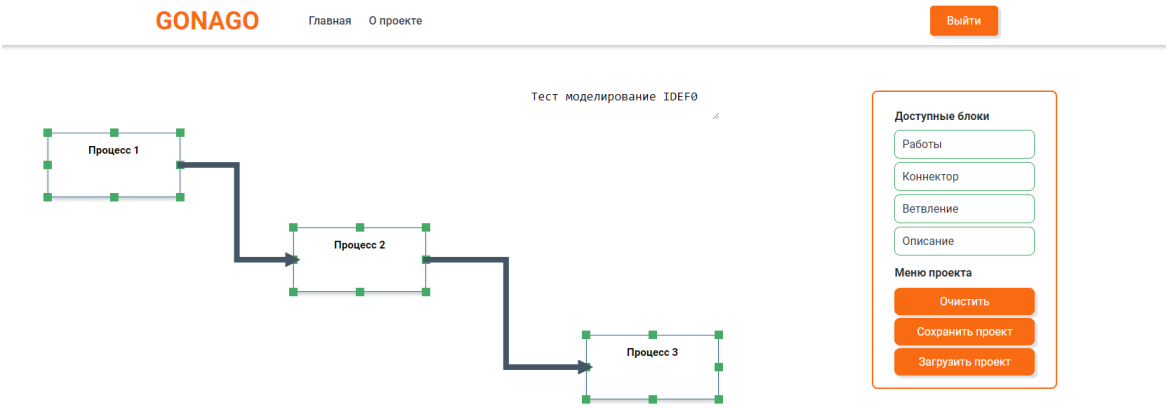


Рисунок 9 - Рабочая область сервиса

5. Требования к документированию: в силу простоты и наглядности реализованного интерфейса сервиса, сопроводительная документация будет представлена кратким руководством пользования.
6. Порядок сдачи-приемки продукта: продукт считается принятым при его положительной оценке экспертами – преподавателями ВУЗа в день защиты ВКР.

Глава 2. Описание проектного решения.

2.1 Модельные представления и описание алгоритма разработки сервиса.

Тип разрабатываемой системы – веб приложение. Любое веб приложение реализовано на основе клиент – серверной архитектуры. Это сетевая архитектура, где процессы обмена данными или файлами распределены «поставщиками» и «заказчиками».

В технологии клиент-сервер есть два главных действующих лица:

- клиент – локальный компьютер на стороне виртуального пользователя, который выполняет отправку запроса к серверу для возможности предоставления данных или выполнения определенной группы системных действий. В данном случае конечный пользователь сервиса.
- сервер – мощный компьютер или специальное системное оборудование, которое предназначается для разрешения определенного круга задач по процессу выполнения программных кодов. Он выполняет работы сервисного обслуживания по клиентским запросам, предоставляет пользователям доступ к определенным системным ресурсам, сохраняет данные или БД.

Особенности такой модели заключаются в том, что пользователь отправляет определенный запрос на сервер, где тот системно обрабатывается и конечный результат отсылается клиенту. В возможности сервера входит одновременное обслуживание сразу нескольких клиентов.

Параметры, которые реализуются на стороне сервера:

1. Хранение, защита и доступ к данным;
2. Работа с поступающими клиентскими запросами;
3. Процесс отправки ответа клиенту.

4. Параметры, которые могут быть реализованы на стороне клиента:

1. Площадка по предоставлению пользовательского графического интерфейса;
2. Формулировка запроса к серверу и его последующая отправка;
3. Получение итогов запроса и отправка дополнительной группы команд (запросы на добавление, обновление информации, удаление группы данных).

Архитектура системы клиент-сервер формулирует принципы виртуального общения между локальными компьютерами, а все правила и принципы взаимодействия находятся внутри протокола. В свою очередь сетевой протокол – это особый набор правил, на основании которого выполняется точное взаимодействие между компьютерами внутри виртуальной сети.

Формализация элементов сервиса (выделение его составных частей и связей):

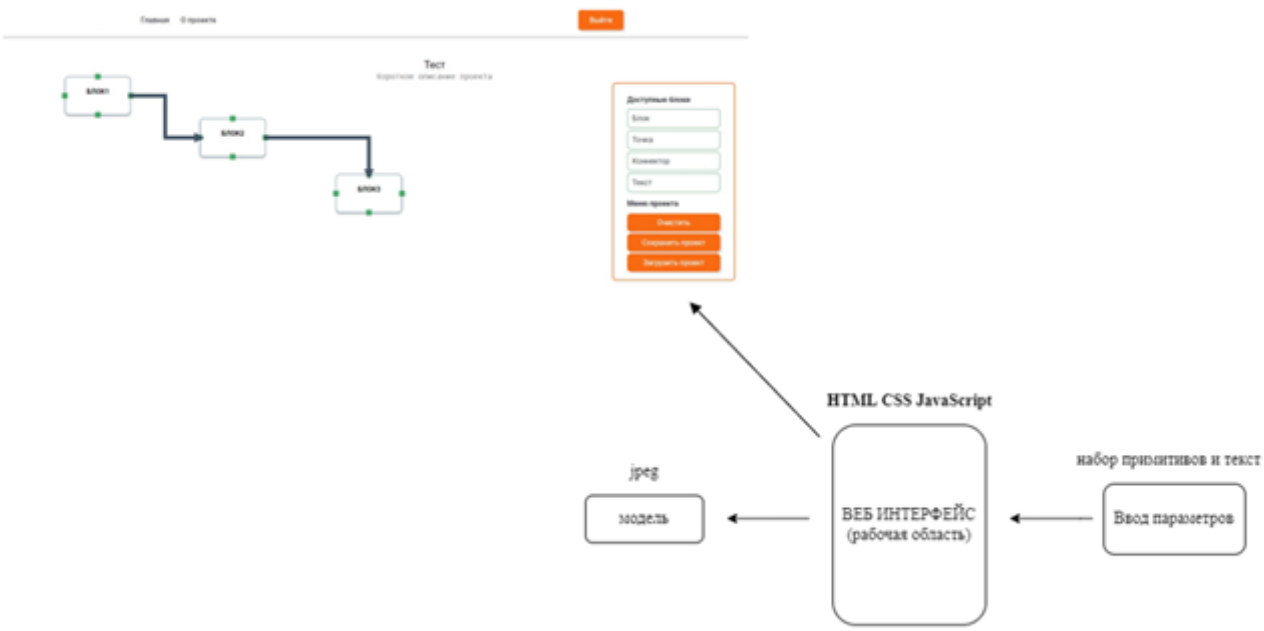
Разделим сервис на части, и он будет представлять стек из:

- фронтенд (html, css, JavaScript);
- бэкенд (JQuery, jsPlumb, JavaScript);

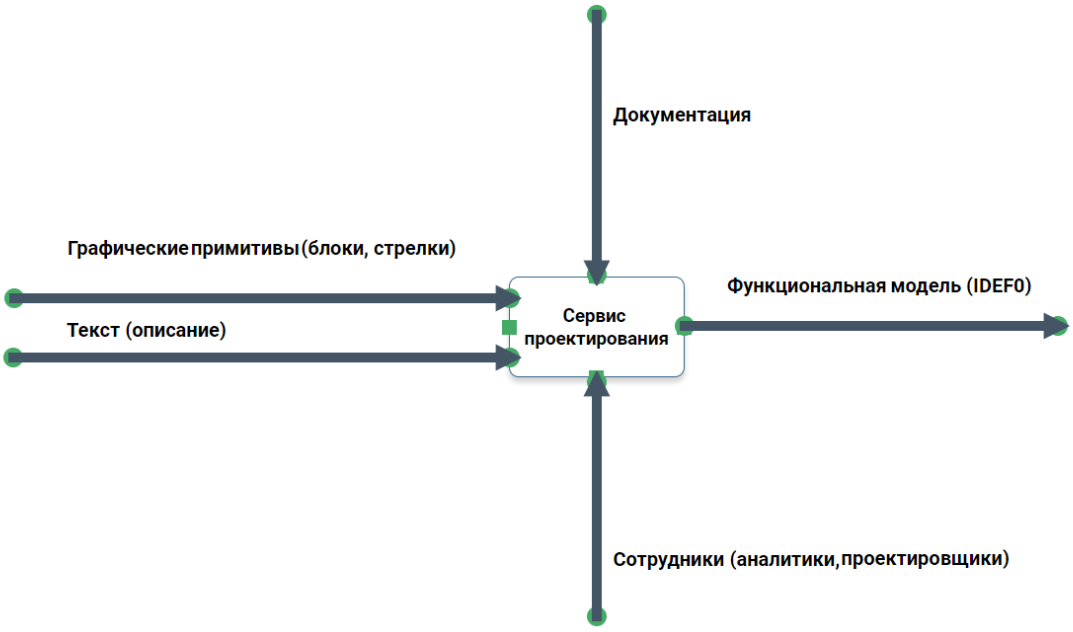
Составные части:

- - главная страница с кратким описанием возможностей;
- - страница авторизации;
- - страница рабочей области;
- - страница о проекте.

Схема информационных потоков сервиса проектирования функциональной модели:



Контекстная диаграмма сервиса проектирования функциональной модели, описанная с помощью разработанного приложения:



Алгоритм разработки сервиса начался с формулировки требований и определения типа разрабатываемого приложения. Дальнейший процесс разработки включает следующие этапы:

- выбор инструментальных средств и путей имплементации проекта;
- развертывание платформы и настройка необходимого программного окружения для разработки;
- разработка макета пользовательского интерфейса;
- реализация интерфейса;
- проектирование функциональной составляющей сервиса;
- реализация функциональной части;
- тестирование и исправление «багов».

Описание средств и инструментов проектирования. После подготовительных работ была начата разработка интерфейса сервиса. С помощью онлайн - сервиса для разработки интерфейсов и прототипирования Figma был разработан макет интерфейса сервиса. Основной упор при разработке был сделан на удобство и дизайн (ux/ui) интерфейса. Для разработки проекта был развернут локальный веб сервер с помощью приложения Open Server. В качестве среды написания и отладки кода был выбран редактор visual studio code.

После окончания настроек окружения начался процесс реализации (вёрстки) разработанного макета. Разработка велась средствами html и css. Листинг приложения изложен в приложении к ВКР. Ниже подробнее описаны основные инструменты, использованные при разработке интерфейса.

HTML — это основа почти любой веб-страницы. В первую очередь выступает как средство для разметки страницы. Состоит из ряда элементов, которые используются для вложения или оборачивания различных частей контента, чтобы заставить его отображаться или действовать определённым образом.

CSS — язык описания внешнего вида документа (каскадные таблицы стилей), написанного с использованием языка разметки. Сетка пользовательского интерфейса построена с помощью технологии css flexbox. Это технология для создания сложных гибких макетов за счёт правильного размещения элементов на странице.

Завершение реализации интерфейса ознаменовало переход к следующему шагу – разработке функциональной части сервиса. Исходя из предъявленных требований и выбранных инструментов для разработки, начался процесс проектирования функциональной части. То есть определение составных частей, зависимостей и требуемого поведения элементов на странице. После был начат процесс реализации, который базируется на основе выбранных средств:

библиотека JQuery, плагин jsPlumb и нативный JavaScript. Ниже так же подробнее описаны основные программные средства, на основе которых реализован функционал.

JavaScript - кроссплатформенный, объектно-ориентированный, скриптовый язык. Создан для легкого встраивания в другие продукты и приложения, например, в веб браузеры.

JQuery — это быстрая лёгкая и многофункциональная JavaScript библиотека, которая позволяет легко:

- выбирать элементы для выполнения различных манипуляций над ними; создавать различные визуальные эффекты (например, плавное отображение и скрытие элементов);
- создавать сложную анимацию, при этом реализовывая это за гораздо меньшее количество строк кода чем на чистом JavaScript; манипулировать DOM элементами и их атрибутами;
- реализовывать AJAX для асинхронного обмена данными между клиентом и сервером; перемещаться от текущего узла к другим по иерархической структуре DOM дерева;
- выполнять несколько действий над элементом посредством одной строчки кода; получать или устанавливать размеры HTML элементам и т. д.

В проекте JQuery совместно с нативным JavaScript служат основой для работы плагина jsPlumb.

jsPlumb составляет ядро, на котором реализован функционал процесса проектирования. Это плагин для JQuery, который позволяет соединить элементы HTML интерфейса динамическими или статическими ссылками.

Ниже описана работа основных элементов jsPlumb, используемых в разрабатываемом сервисе, которые нужны для проектирования функциональной модели.

Соединители: линии, которые фактически соединяют элементы пользовательского интерфейса. В jsPlumb имеются четыре типа соединителей, для построения функциональной модели необходимо использование только двух - прямой соединитель и соединитель «блок-схемы».

Соединители задаются одним из двух способов: либо по имени, которое создает экземпляр соединителя со значениями по умолчанию, либо по имени и значениям конструктора.

Указание коннектора по имени - здесь мы соединим два элемента «Straight» коннектором:

```
import { StraightConnector } from "@jsplumb/core"

instance.connect({
  source:someElement,
  target:someOtherElement,
  connector:StraightConnector.type
})
```

У каждого соединителя есть type доступный статический член, который помогает обеспечить безопасность типов при указании соединителя. Однако вы можете использовать здесь строку, если вы не используете ES6/Typescript:

```
instance.connect({
  source: someElement,
  target: someOtherElement,
  connector: 'Straight'
})
```

Указание коннектора по имени и параметрам:

Интерфейс, определяющий определение соединителя, называется «ConnectorSpec».

```
import { BezierConnector } from "@jsplumb/bezier"

instance.connect({
    source: someElement,
    target: someOtherElement,
    connector:{
        type:BezierConnector.type,
        options:{
            curviness: 50
        }
    }
})
```

Прямой соединитель: импортируется из @jsplumb/core. Рисует прямую линию между двумя конечными точками. Поддерживаются два аргумента конструктора:

- stub - необязательный, по умолчанию равен 0. Любое положительное значение этого параметра приведет к тому, что от конечной точки отходит отрезок такой длины перед прямым сегментом, соединяющимся с другим концом соединения.
- gap - необязательно, по умолчанию 0. Промежуток между конечной точкой и либо началом заглушки, либо сегментом, соединяющим другую конечную точку.

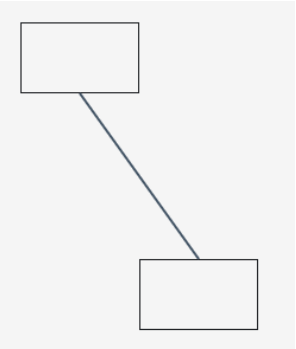


Рисунок 10 - Прямой соединитель

Соединитель блок-схемы: импортируется из @jsplumb/connector-flowchart. Рисует соединение, состоящее из ряда вертикальных или горизонтальных сегментов — классический вид блок-схемы. Поддерживаются четыре аргумента конструктора:

- stub- минимальная длина в пикселях начальной заглушки, исходящей из конечной точки. Это необязательный параметр, который может быть либо целым числом, указывающим заглушку для каждого конца коннектора, либо массивом из двух целых чисел, указывающим заглушку для конечных точек (источник – цель) в соединении. По умолчанию используется целое число со значением 30 пикселей.
- alwaysRespectStubs- необязательный, по умолчанию «false». Этот параметр указывает jsPlumb всегда рисовать линию заданной длины заглушки из каждой конечной точки вместо автоматического уменьшения заглушек, если два элемента ближе, чем сумма двух заглушек.
- gap- необязательный, по умолчанию 0 пикселей. Позволяет указать зазор между концом соединителя и элементами, к которым он прикреплен.
- midpoint- необязательный, по умолчанию 0,5. Это расстояние между двумя элементами, на котором будет нарисована самая длинная часть соединителя блок-схемы. Этот параметр полезен в тех случаях, когда у вас есть программный контроль над рисунком и, возможно, вы хотите избежать какого-либо другого элемента на странице.

- cornerRadius - по умолчанию 0. Положительное значение этого параметра приведет к изогнутым угловым сегментам.

Этот соединитель поддерживает соединения, которые начинаются и заканчиваются на одном и том же элементе ("петлевые" соединения).

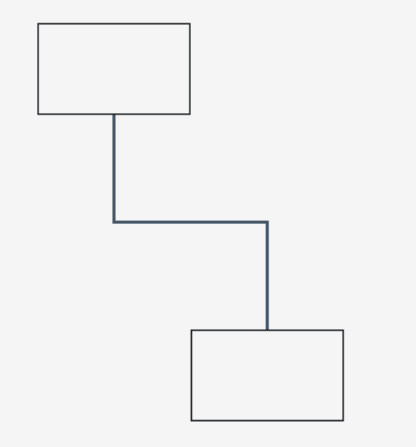


Рисунок 11 - Соединитель блок-схемы

Якоря: моделируют представление о том, где на элементе должен соединяться коннектор — он определяет положение конечной точки. В jsPlumb существуют четыре основных типа якорей, для моделирования был использован только один – статический. Он фиксируется в некоторой точке элемента и не двигается. jsPlumb имеет девять мест привязки по умолчанию, которые вы можете использовать, чтобы указать, где соединители соединяются с элементами: это четыре угла элемента, центр элемента и середина каждого края элемента:

- Top;
- TopRight;
- Right;
- BottomRight;
- Bottom;
- BottomLeft;
- Left;

- TopLeft;
- Center.

Перечисление «AnchorLocations» содержит эти значения. Каждое из этих строковых представлений является просто оболочкой базового синтаксиса на основе массива [x, y, dx, dy], где хи у — координаты в интервале [0,1], определяющем положение якоря, а dx и dy— которые определяют ориентацию кривой, инцидентной якорю, могут иметь значение 0, 1 или -1. Например, [0, 0.5, -1, 0] определяет «Left» привязку с кривой соединителя, которая исходит влево от привязки. Точно так же [0.5, 0, 0, -1] определяет Top привязку с кривой соединителя, идущей вверх.

```
anchor:"Bottom"
```
или же
```
anchor: AnchorLocations.Bottom
```

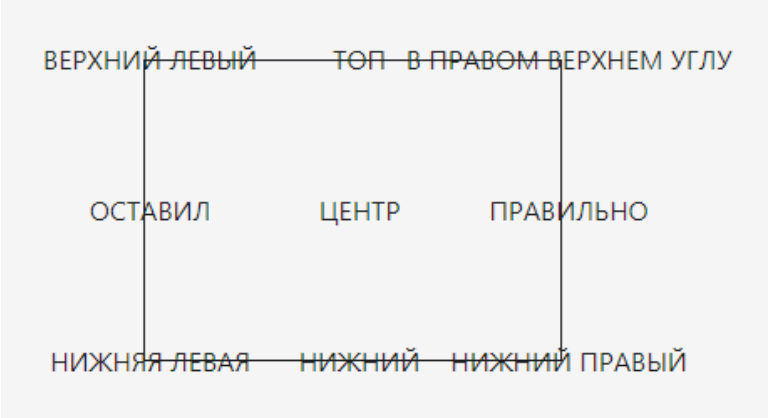


Рисунок 12 - Статические якоря.

Конечные точки: jsPlumb поставляется с тремя реализациями «Endpoint Dot» — «Rectangle» и «Blank». В проекте используются точка и прямоугольник. Конечные точки указываются с использованием того же синтаксиса, который используется для указания соединителей и наложений, либо с использованием своего имени:

```
import { DotEndpoint } from "@jsplumb/core"

instance.addEndpoint(someElement, {
    endpoint: DotEndpoint.type
})
```

```
instance.addEndpoint(someElement, {
    endpoint: "Dot"
})
```

Так можно используя его имя и некоторые параметры конструктора:

```
import { DotEndpoint } from "@jsplumb/core"

instance.addEndpoint(someElement, {
    endpoint: {
        type: DotEndpoint.type,
        options:{
            radius: 5
        }
    }
})
```

Создание конечной точки: конечные точки создаются несколькими способами:

- когда вы вызываете «connect» (..) экземпляр jsPlumb и передаете идентификатор элемента или элемент DOM в качестве источника/цели, создается и назначается новая конечная точка.
- когда вы вызываете «addEndpoint» (...) экземпляр jsPlumb, создается новая конечная точка (и возвращается из вызова)
- когда вы настроили перетаскиваемое соединение с «addSourceSelector» (...) помощью экземпляра jsPlumb и впоследствии перенесли соединение из соответствующего элемента, создается и назначается новая конечная точка.

В каждом из этих различных случаев параметры, которые вы можете использовать для указания конечной точки, которую вы хотите создать, абсолютно одинаковы. Размер конечной точки не ограничен.

Типы конечных точек: тип точка рисует точку на экране. Поддерживает три параметра конструктора:

- radius - необязательно; по умолчанию 5 пикселей. Определяет радиус точки.
- cssClass - необязательно. Класс CSS для присоединения к элементу, который создает конечная точка.
- hoverClass - необязательно. Класс CSS для присоединения к элементу, который Endpoint создает всякий раз, когда указатель мыши находится над элементом или прикрепленным соединением.

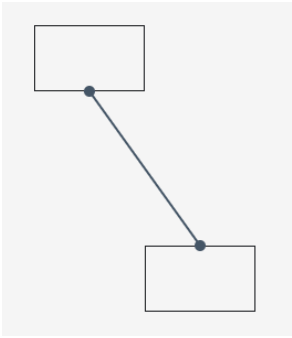


Рисунок 13 - Тип конечной точки «точка».

Следующий тип – прямоугольник: рисует прямоугольник. Поддерживаемые параметры конструктора:

- width - необязательно; по умолчанию 20 пикселей. Определяет ширину прямоугольника.
- height - необязательно; по умолчанию 20 пикселей. Определяет высоту прямоугольника.
- cssClass - необязательно. Класс CSS для присоединения к элементу, который создает конечная точка.

- hoverClass - необязательно. Класс CSS для присоединения к элементу, который Endpoint создает всякий раз, когда указатель мыши находится над элементом или прикрепленным соединением.

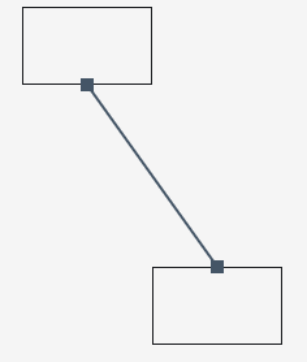


Рисунок 14 - Тип конечной точки «прямоугольник».

2.2 Описание результатов разработки и рекомендации по применению разработанного продукта.

Результатом разработки, осуществленной в прошлом разделе, стал рабочий прототип сервиса проектирования функциональной модели ИС.

Процесс работы в разработанном сервисе начинается со страницы приветствия, на которой расположены:

- меню для навигации;
- кнопка входа в систему;
- блок с кратким описанием сервиса;
- кнопка для начала проектирования.

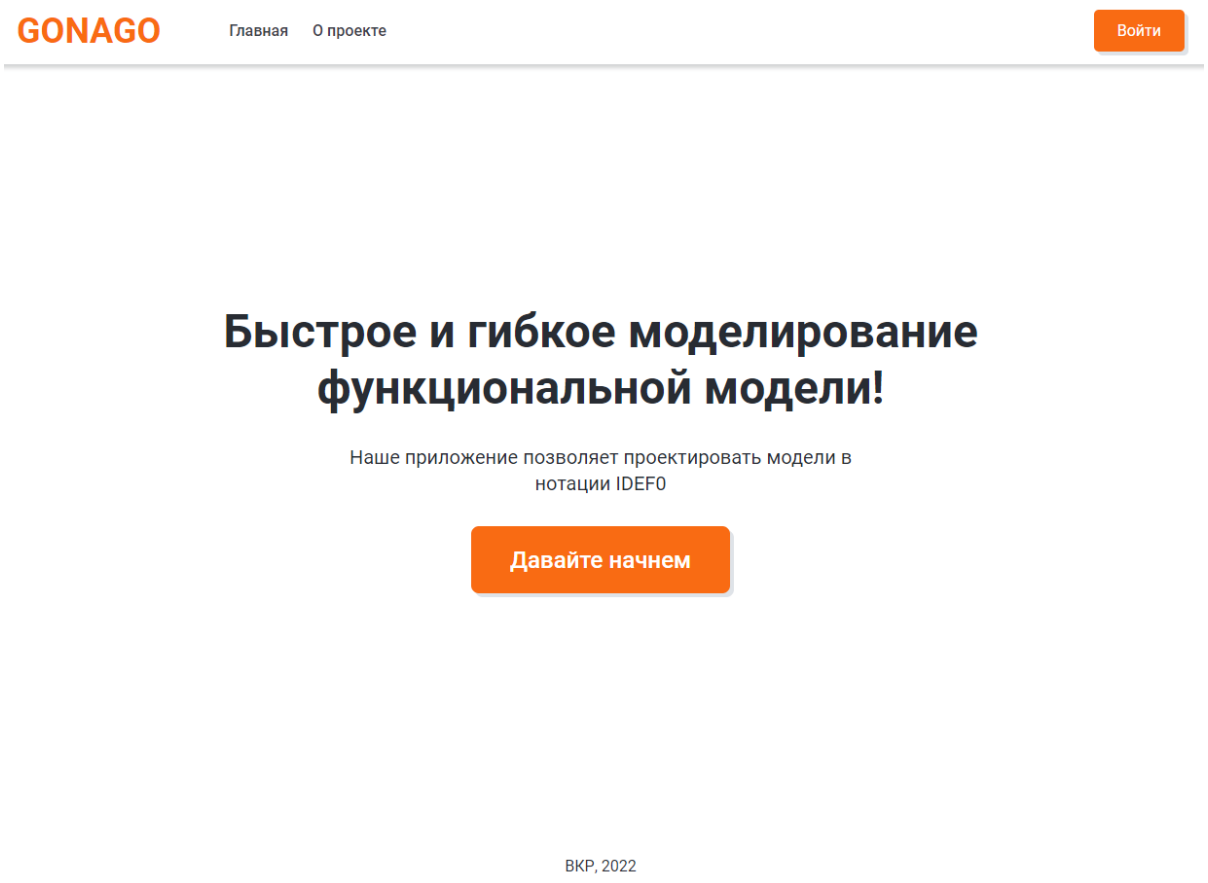


Рисунок 15 - Страница приветствия.

При нажатии кнопки «Войти» откроется страница авторизации.

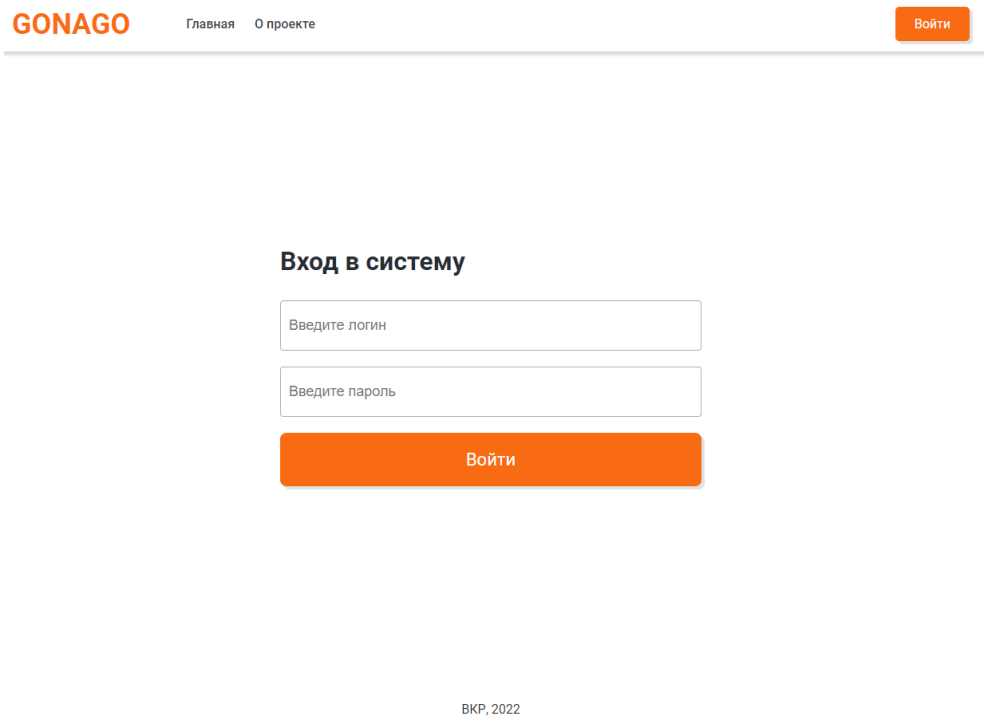


Рисунок 16 - Страница входа в систему.

После введения учетных данных происходит переход на рабочую страницу, на которой непосредственно и происходит процесс моделирования.

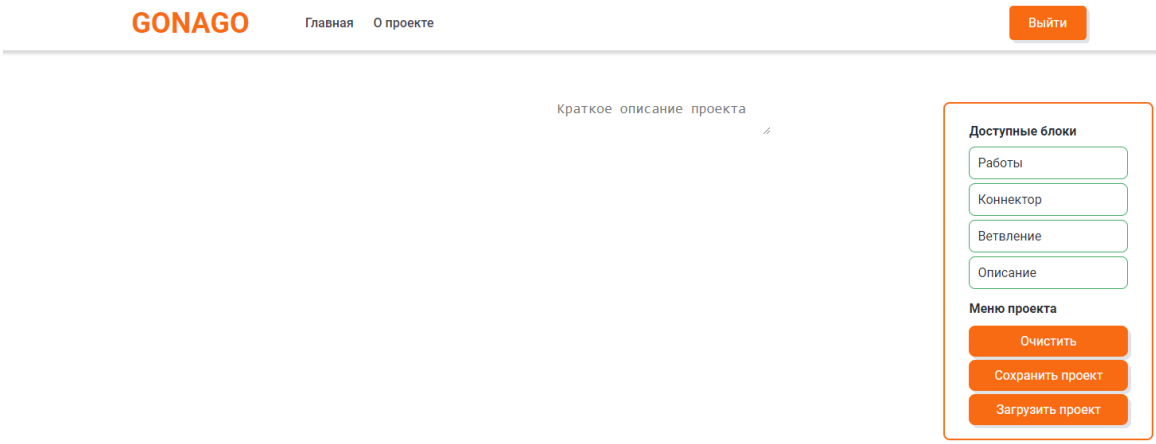


Рисунок 17 - Рабочая страница моделирования.

На рабочей странице располагается меню навигации, рабочая область проектирования, плавающее меню инструментов, совмещенное с меню проекта и плавающий блок описания проекта. Особый интерес здесь представляет блок меню инструментов. Оно реализует весь требуемый от сервиса функционал.

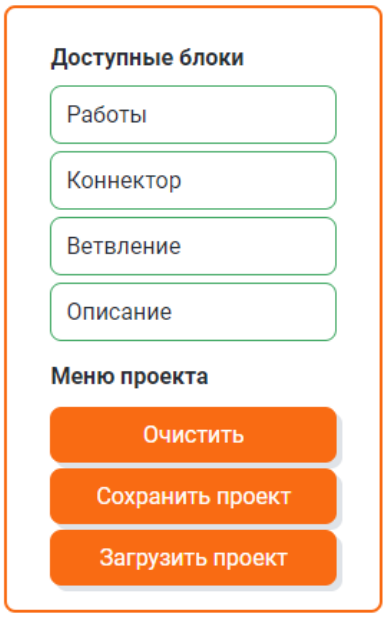


Рисунок 18 - Рабочая страница моделирования.

Из доступных инструментов имеется:

Работы (Activity): объект служит для обозначения поименованных процессов, задач или функций. Работы изображаются в виде прямоугольников. У каждого «activity» есть поле для наименования.

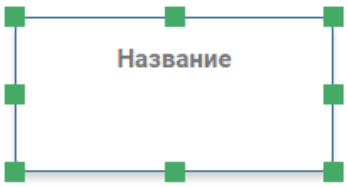


Рисунок 19 - Объект «работы».

Коннектор: объект позволяет создавать очень важный элемент описания моделей – стрелки.

Рисунок 20 - Объект «работы».

Рисунок – объект «коннектор».

Ветвление – объект дает возможность создавать ветвящиеся стрелки:

Рисунок 21 - Объект «работы».

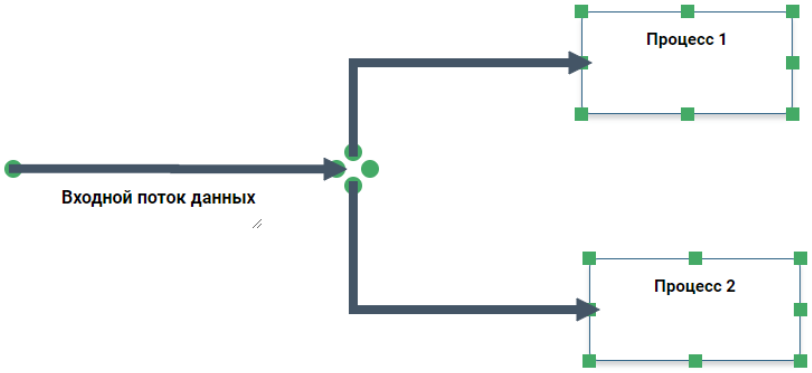


Рисунок 22 - Пример использования объекта «ветвление».

Описание – объект служит в качестве блока описание, позволяет именовать объекты в процессе моделирования.

Так же у каждого элемента есть кнопка удаления, которая появляется при наведении:

Рисунок 23 - Кнопки удаления элементов.

Меню проекта содержит три кнопки:

- кнопка «очистить» позволяет очистить рабочую область;
- кнопка «Сохранить проект» сохранит проект в файле формата json;
- кнопка «Загрузить проект» позволит открыть ранее сохраненный проект для доработки.

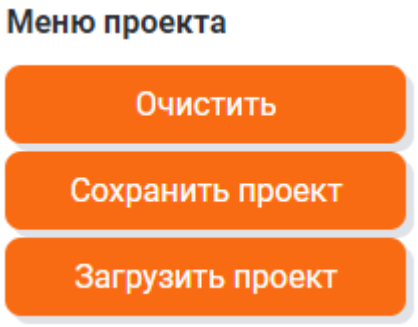


Рисунок 24 - Меню проекта.

Пример построения контекстной диаграммы в разработанном сервисе:

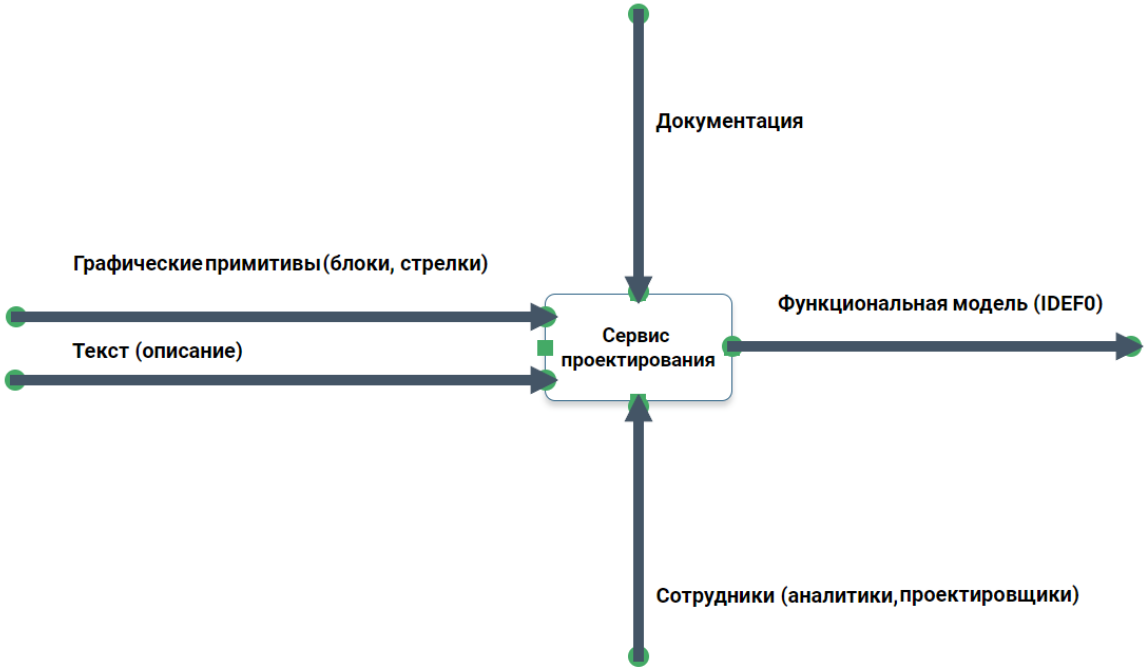


Рисунок 25 - Контекстная диаграмма, описанная средствами разработанного сервиса.

2.3 Результаты апробации продукта.

По результатам апробации можно резюмировать:

- представленный прототип веб сервиса проектирования функциональной модели ИС соответствует описанию и отвечает заявленным требованиям;
- сервис выполняет свою главную функцию – проектирование функциональной модели ИС в заданной нотации IDEF0;
- сервис выполнен в соответствии с техническим заданием и предоставляет минимальный, но достаточный для работы набор возможностей.

Заключение

Результатом выпускной квалификационной работы стал не только подробный разбор предметной области, но и рабочий прототип сервиса проектирования функциональной модели ИС, удовлетворяющий всем предъявленным требованиям.

Описание методологий и алгоритмов проектирования, на ряду с разработкой собственного сервиса, позволили углубиться в вопрос. Благодаря этому были получены новые знания в области моделирования процессов, и что еще более важно - серьезный опыт разработки программного продукта. Помимо этого, был получен опыт составления функциональных требований, требований к дизайну и «юзабилити» интерфейса, а также технического задания на разработку

На данном этапе своего развития сервис, в соответствии с техническим заданием, обладает минимально достаточным набором функций. Дальнейшая разработка по модели прототипирования позволит сервису обзавестись новым функционалом и планомерно масштабироваться.

Приложения

Приложение 1.

Листинг программного кода

Файл style.css:

```
* {
    padding: 0;
    margin: 0;
}
body {
    font-family: 'Roboto', sans-serif;
    height: 100vh;
    color: #282c33;
}
.container {
    width: 1200px;
    margin: 0 auto;
}
.main-header {
    display: flex;
    align-items: center;
    height: 70px;
    box-shadow: rgb(40 44 51 / 20%) 0px 4px 4px;
}
.main-header-content {
    display: flex;
    align-items: center;
    justify-content: space-between;
}
.logo {
    font-size: 36px;
    font-weight: 600;
    color: #f96b13;
}
.main-menu {
    display: flex;
    align-items: center;
    list-style: none;
    margin-right: auto;
    margin-left: 70px;
}
.main-menu li {
    margin-right: 25px;
}
.main-menu a {
    font-size: 16px;
    text-decoration: none;
    color: #3A414A;
    font-weight: 500;
}
.main-menu a:hover {
    color: #f96b13;
}
.btn {
    background: #f96b13;
    color: white;
    text-decoration: none;
    box-shadow: rgb(223 227 232) 4px 4px 0px 0px;
    text-align: center;
    border-radius: 8px;
}
.header-btn {
    padding: 12px 24px;
    border-radius: 5px;
    box-shadow: rgb(223 227 232) 4px 4px 0px 0px;
}
.btn:hover {
    background: #c3520d;
}
.main {
    height: calc(100% - 140px);
}
.main .container {
    height: 100%;
}
.main-content {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100%;
}
.main-title {
    font-size: 48px;
    color: #282C33;
```

```
    font-weight: 700;
    text-align: center;
    width: 70%;
}
.main-sub {
    color: #282C33;
    font-size: 20px;
    font-weight: 400;
    line-height: 1.35;
    margin-top: 30px;
    margin-bottom: 30px;
    max-width: 574px;
    text-align: center;
}
.main-btn {
    font-size: 24px;
    font-weight: 500;
    padding: 20px 40px;
    border-radius: 8px;
    box-shadow: rgb(223 227
232) 4px 4px 0px 0px;
}
.main-img {
    width: 60%;
    margin-top: 50px;
}


.footer-content {
    display: flex;
    align-items: center;
    justify-content: center;
    height: 70px;
}
.login-form {
    width: 528px;
    display: flex;
    flex-direction: column;
}
.login-title {
    font-size: 32px;
    margin-bottom: 30px
}
.login-form input {
    border-radius: 4px;
    border: 1px solid #a9afb8;
    color: #282c33;
    background: #fff;
    overflow: hidden;
    font-size: 18px;
    padding: 20px 10px;
    margin-bottom: 20px
}


.login-error,
.pass-error {
    margin-bottom: 5px;
    color: #f91313;
    font-size: 14px;
    display: none;
}
.login-btn {
    text-align: center;
    padding: 20px;
    border-radius: 8px;
    font-size: 22px;
}
/*app----------------------------
----------------------------------
----*/
.app {
    position: relative;
    height: 100%;
    width: 100%;
}
.canvas {
    width: 100%;
    height: 100%;
}
.menu {
    background: white;
    padding: 15px 30px;
    width: 200px;
    border: 2px solid #f96b13;
    position: absolute;
    top: 100px;
    left: 80%;
    cursor: pointer;
    border-radius: 8px;
    visibility: visible
!important;
}
.window {
    cursor: pointer;
}
.app-title {
    position: absolute;
    left: 40%;
    top: 1%;
   padding: 10px;
    display: flex;
    flex-direction: column;
}
.app-title:hover {
    outline: 1px solid
#346789;
```

```
      box-shadow: rgb(40 44 51 / 20%) 0px 4px 4px;
}
.app-title textarea {
    border: none;
    font-size: 22px;
    font-weight: 500;
    min-width: 300px;
    text-align: center;
}
.app-title textarea {
    text-align: center;
    font-size: 18px;
    border: none;
}
.menu-head {
    display: flex;
}
.menu-foot {
    display: flex;
    flex-direction: column;
}
.menu-title {
    width: 100%;
    margin: 10px 0
}
.save-btn {
    padding: 10px 15px;
    margin: 2px 0;
}
.browse-file {
    display: none;
}
.menu_button {
    border: 1px solid #4A6;
    padding: 10px;
    margin-bottom: 5px;
    border-radius: 8px;
}
.menu_button:hover {
    background: #4A6;
    color: white;
}
.task {
    padding: 5px;
    background-color: white;
    border: 1px solid #346789;
    min-height: 80px;

    position: absolute;
    min-width: 160px;
    display: flex;
    align-items: center;
    justify-content: center;
    box-shadow: rgb(40 44 51 / 20%) 0px 4px 4px;
    border-radius: 8px;
    cursor: grab;
}
.task-input {
     font-weight: 600;
    font-family: 'Roboto', sans-serif;
}
.label {
    max-width: 600px;
    width: fit-content;
    min-width: 200px;
    min-height: 25px;
    background: none;
    position: absolute;
    cursor: grab;
    z-index: 999;
    display: flex;
    align-items: center;
    padding: 5px;
    padding-left: 10px;
}
.label-input {
     font-weight: 600;
    font-family: 'Roboto', sans-serif;
    font-size: 16px;
    border: none;
}
.label:hover {
      cursor: grabbing;
     outline: 1px solid #346789;
     box-shadow: rgb(40 44 51 / 20%) 0px 4px 4px;
}
.task:active {
    cursor: grabbing;
}
.task-input {
    width: 90%;
    border: none;
    text-align: center;
    font-size: 15px;
    resize: none;
    min-height: 60px;
}
textarea:active,
textarea:focus,
```

```
textarea:focus-within {
    outline: none;
}
.cross {
    width: 30px;
    height: 30px;
    position: absolute;
    border-radius: 50%;
}
.pointF  {
    width: 65px;
    height: 65px;
    position: absolute;
    border-radius: 50%;
}
.cross:hover,
.pointF:hover {
    outline: 1px solid
#346789;
    cursor: grab;
}
.button_remove {
    font-size: 16px;
    color: red;
    background:
url(../img/close.svg);
    width: 10px;
    height: 10px;
    background-size: contain;
    background-repeat: no-
repeat;
    opacity: .7;
}
.ctrl_container {
    position: absolute;
    display: flex;
    align-items: center;
    justify-content: center;
    width: 20px;
    height: 20px;
    right: 5px;
    top: 5px;
    cursor: pointer;
    visibility:hidden;
}
.ctrl_container:hover
.button_remove  {
    opacity: 1;
}
.window:hover
.ctrl_container {
    visibility: visible;
}
#jsonOutput {
    width: 98%;
    height: 50px;
    margin-left: 1%;
    display: none;
}
```

Файл app.js:

```
$(function(){
$("#file").change(function(){
let file =
document.getElementById('file').files[0];
let reader = new FileReader();
    reader.readAsText(file);
    reader.onload = function() {
    document.getElementById('jsonOutput').value
= reader.result;
    }
        reader.onerror = function() {
            alert(reader.error);
    }
//        setTimeout(loadFlowchart, 100);
        setTimeout(load, 100);
    });
});
var numberOfElements = 0;
var htmlBase = 'drawingArea';
jsPlumb.ready(function () {

            $(("#taskcontainer0"))[0].innerHTML =
$(("#taskcontainer0"))[0].innerHTML;
    $(("#pointFcontainer0"))[0].innerHTML =
$(("#pointFcontainer0"))[0].innerHTML;
```

```
    $(("#crosscontainer0"))[0].innerHTML =
$(("#crosscontainer0"))[0].innerHTML;
    $(("#labelcontainer0"))[0].innerHTML =
$(("#labelcontainer0"))[0].innerHTML;
        jsPlumb.draggable($(".window"));
        jsPlumb.importDefaults({
            Endpoint : ["Rectangle",
{radius:8}],
            EndpointStyle : { fillStyle :
"#4A6" },
            Connector:[ "Flowchart" ],
            ConnectionOverlays : [
                [ "Arrow", {
                    location:-1,
                    id:"arrow",
                    length:20,
                    foldback:1
                } ]
            ]
        });
        $('#'+htmlBase).on("click",
".button_remove", function () {
            var parentnode =
$(this)[0].parentNode.parentNode;
        jsPlumb.detachAllConnections(parentno
de);

        jsPlumb.removeAllEndpoints(parentnod
e);
            $(parentnode).remove();
        });
        $(".button_add_task").click(function () {
            addTask();
        });
    $(".button_add_pointF").click(function () {
            addPointF();
        });
     $(".button_add_cross").click(function () {
            addCross();
        });
      $(".button_add_label").click(function () {
            addLabel();
        });
        $('#saveButton').click(function(){
            saveFlowchart();
        });
        $('#loadButton').click(function(){
            loadFlowchart();
        });
    $('#cleanBtn').click(function(){
            clean();
        });
});
function addTask(id, label = false){
        if(typeof id === "undefined"){
            numberOfElements++;
            id = "taskcontainer" +
numberOfElements;
        }
        $('<div class="window task node" id="'
+ id + '" data-nodetype="task" style="left:30px;
top:30px;">').appendTo('#'+htmlBase).html($("
#taskcontainer0"))[0].innerHTML);
        if (label) {
            $('#' +
id).find('textarea').val(label);
        }
        var taskTargetConnectorEndpoint1 = {
            isSource: true,
            isTarget: true,
            maxConnections: 4,
            anchor: "Right",
      endpoint: ["Rectangle", {width:12,
height:12}]
        };
```

```
        var taskTargetConnectorEndpoint2 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "Left",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
        var taskTargetConnectorEndpoint3 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "Top",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
    var taskTargetConnectorEndpoint4 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "Bottom",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
    var taskTargetConnectorEndpoint5 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "TopRight",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
    var taskTargetConnectorEndpoint6 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "BottomRight",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
    var taskTargetConnectorEndpoint7 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "TopLeft",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
    var taskTargetConnectorEndpoint8 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor: "BottomLeft",
                endpoint: ["Rectangle", {width:12, height:12}]
        };
        jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint1
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint2
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint3
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint4
```

```
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint5
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint6
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint7
        );
    jsPlumb.addEndpoint(
                $('#'+id),
                taskTargetConnectorEndpoint8
        );
        jsPlumb.draggable($('#' + id));
        return id;
}
function clean() {
document.querySelector('#drawingArea').style.visibility = "hidden";
document.querySelector('#drawingArea').style.opacity = "0";
    document.querySelector('#jsonOutput').value = "";
}
function load() {
document.querySelector('#drawingArea').style.visibility = "visible";

document.querySelector('#drawingArea').style.opacity = "1";
}
function addLabel(id, label = false){
        if(typeof id === "undefined"){
                numberOfElements++;
                id = "labelcontainer" + numberOfElements;
        }
        $('<div class="window label node" id="' + id + '" data-nodetype="label" style="left:30px; top:30px;">').appendTo('#'+htmlBase).html($("#labelcontainer0"))[0].innerHTML);
        if (label) {
                $('#' + id).find('textarea').val(label);
        }
        jsPlumb.draggable($('#' + id));
        return id;
}
function addPointF(id){
        if(typeof id === "undefined"){
                numberOfElements++;
                id = "pointFcontainer" + numberOfElements;
        }
        $('<div class="window pointF node" id="' + id + '" data-nodetype="pointF" style="left:30px; top:30px;">').appendTo('#'+htmlBase).html($("#pointFcontainer0"))[0].innerHTML);
        var taskSourceConnectorEndpoint = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor:"Center",
      endpoint: ["Dot", {radius:8}]
        };
        jsPlumb.addEndpoint(
                $('#'+id),
                taskSourceConnectorEndpoint
        );
        jsPlumb.draggable($('#' + id));
        return id;
```

```
}
function addCross(id){
        if(typeof id === "undefined"){
                numberOfElements++;
                id = "crosscontainer" + numberOfElements;
        }
        $('<div class="window cross node" id="' + id + '" data-nodetype="cross" style="left:30px; top:30px;">').appendTo('#'+htmlBase).html($("#crosscontainer0"))[0].innerHTML);

        var taskSourceConnectorEndpoint1 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor:"Left",
      endpoint: ["Dot", {radius:8}]
        };
   var taskSourceConnectorEndpoint2 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor:"Right",
     endpoint: ["Dot", {radius:8}]
        };
   var taskSourceConnectorEndpoint3 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor:"Top",
     endpoint: ["Dot", {radius:8}]
        };
   var taskSourceConnectorEndpoint4 = {
                isSource: true,
                isTarget: true,
                maxConnections: 4,
                anchor:"Bottom",
     endpoint: ["Dot", {radius:8}]
        };
        jsPlumb.addEndpoint(
                $('#'+id),
                taskSourceConnectorEndpoint1
        );
   jsPlumb.addEndpoint(
                $('#'+id),
                taskSourceConnectorEndpoint2
        );
   jsPlumb.addEndpoint(
                $('#'+id),
                taskSourceConnectorEndpoint3
        );
   jsPlumb.addEndpoint(
                $('#'+id),
                taskSourceConnectorEndpoint4
        );
        jsPlumb.draggable($('#' + id));
        return id;
}
function saveFlowchart(){
        var nodes = []
        $(".node").each(function (idx, elem) {
        var $elem = $(elem);
        var endpoints = jsPlumb.getEndpoints($elem.attr('id'));
                nodes.push({
                        blockId: $elem.attr('id'),
                        nodetype: $elem.attr('data-nodetype'),
                        positionX: parseInt($elem.css("left"), 10),
```

```
                positionY:
parseInt($elem.css("top"), 10),
                label:
$elem.find('textarea').val()
            });
    });
    var connections = [];
    $.each(jsPlumb.getConnections(),
function (idx, connection) {
        connections.push({
                connectionId:
connection.id,
                pageSourceId:
connection.sourceId,
                pageTargetId:
connection.targetId
        });
    });
    var flowChart = {};
    flowChart.nodes = nodes;
    flowChart.connections = connections;
    flowChart.numberOfElements =
numberOfElements;
    var flowChartJson =
JSON.stringify(flowChart);
    $('#jsonOutput').val(flowChartJson);
function textToFile (text, name) {
    const b = new Blob([text], { type:
'text/plain' });
    const url =
window.URL.createObjectURL(b);
    const a = document.createElement('a');
    a.href = url;
    a.download = name || 'text.txt';
    a.type = 'text/plain';
    a.addEventListener('click', () => {
        setTimeout(() =>
window.URL.revokeObjectURL(url), 10000);
    })
    a.click()
}
textToFile
(document.getElementById('jsonOutput').value,
`Project.json`);
}
function loadFlowchart(){
   var flowChartJson = $('#jsonOutput').val();
    var flowChart =
JSON.parse(flowChartJson);
    var nodes = flowChart.nodes;
    $.each(nodes, function( index, elem ) {
        if(elem.nodetype === 'task'){
            var id =
addTask(elem.blockId, elem.label);
            repositionElement(id,
elem.positionX, elem.positionY);
        }else if(elem.nodetype ===
'pointF'){
            var id =
addPointF(elem.blockId);
            repositionElement(id,
elem.positionX, elem.positionY);
        } else if(elem.nodetype ===
'cross'){
            var id =
addCross(elem.blockId);
            repositionElement(id,
elem.positionX, elem.positionY);
        }else  if(elem.nodetype ===
'label'){
            var id =
addLabel(elem.blockId, elem.label);
            repositionElement(id,
elem.positionX, elem.positionY);
        } else {}
    });
```

```
        var connections =
flowChart.connections;
        $.each(connections, function( index,
elem ) {
                var connection1 =
jsPlumb.connect({
                        source:
elem.pageSourceId,
                        target:
elem.pageTargetId,
                        anchors:
["BottomCenter", [0.75, 0, 0, -1]]
                });
        });
        numberOfElements =
flowChart.numberOfElements;
}
function repositionElement(id, posX, posY){
        $('#'+id).css('left', posX);
        $('#'+id).css('top', posY);
        jsPlumb.repaint(id);
}
```

Файл index.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="UTF-8">
   <title>Дипломная работа</title>
   <link rel="stylesheet" href="css/style.css">
   <link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;
1,100;1,300;1,400;1,500;1,700;1,900&display=swap" rel="stylesheet">
</head>
<body>
<section class="main-header">
   <div class="container">
      <div class="main-header-content">
         <div class="logo" onclick="location.href='/'">GONAGO</div>
         <ul class="main-menu">
            <li><a href="">Главная</a></li>
            <li><a href="">О проекте</a></li>
         </ul>
         <a href="login.html" class="btn header-btn">Войти</a>
      </div><!-- main-header-content -->
   </div><!-- container -->
```

```
</section><!-- main-header -->
<section class="main">
    <div class="container">
        <div class="main-content">
            <h1 class="main-title">Быстрое и гибкое моделирование функциональной модели!</h1>
            <p class="main-sub">Наше приложение позволяет проектировать модели в нотации IDEF0</p>
            <a href="login.html" class="btn main-btn">Давайте начнем</a>
        </div><!-- main-content -->
    </div><!-- container -->
</section><!-- main -->
<section class="main-footer">
    <div class="container">
        <div class="footer-content">
            <p>ВКР, 2022 </p>
        </div>
    </div>
</section>
<script src="vendor/jq.min.js"></script>
<script src="vendor/jq-ui.js"></script>
<script src="vendor/jq-jsplumb.min.js"></script>
</body>
</html>
```

Файл app.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Дипломная работа</title>
    <link rel="stylesheet" href="css/style.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap" rel="stylesheet">
```

```
</head>
<body style="min-height: 3000px;">
   <section class="main-header">
      <div class="container">
         <div class="main-header-content">
            <div class="logo" onclick="location.href='/'">GONAGO</div>
            <ul class="main-menu">
               <li><a href="">Главная</a></li>
               <li><a href="">О проекте</a></li>
            </ul>
            <a href="/login.html" class="btn header-btn">Выйти</a>
         </div><!-- main-header-content -->
      </div><!-- container -->
   </section><!-- main-header -->
   <section class="main">
         <div class="app">
           <div class="window menu" id="menuContainer">
                <div class="menu-head">
                   <b class="menu-title">Доступные блоки</b>
                </div>
                 <div class="menu_button_container">
                    <div class="button_add_task button menu_button">Работы</div>
                    <div class="button_add_pointF button menu_button">Коннектор</div>
                    <div class="button_add_cross button menu_button">Ветвление</div>
                    <div class="button_add_label button menu_button">Описание</div>
                 </div>
                 <div class="menu-foot">
                     <b class="menu-title">Меню проекта</b>
                     <span id="cleanBtn" class="btn save-btn">Очистить</span>
                     <span id="saveButton" class="btn save-btn">Сохранить проект</span>
                     <span class="btn save-btn" onclick="$('#file').trigger('click');">Загрузить
проект</span>
                    <form id="form" method="post" class="browse-file"> <input type="file" id="file"
></form>
                 </div>
```

```
            </div>
          <div id="drawingArea" class="canvas">
            <div  class="window app-title">
              <textarea placeholder="Краткое описание проекта"></textarea>
            </div>
             <div class="window task" style="left: 120px; top:200px; display:none;" data-nodetype="task" id="taskcontainer0">
               <div class="ctrl_container">
                 <div class="button_remove"></div>
               </div>
               <textarea class="task-input" placeholder="Название"></textarea>
             </div>
             <div class="window ponitF" style="left: 250px; top:300px; display:none;" data-nodetype="pointF" id="pointFcontainer0">
               <div class="ctrl_container">
                 <div class="button_remove"></div>
               </div>
             </div>
             <div class="window cross" style="left: 250px; top:300px; display:none;" data-nodetype="pointF" id="crosscontainer0">
               <div class="ctrl_container">
                 <div class="button_remove"></div>
               </div>
             </div>
               <div class="window label" style="left: 120px; top:200px; display:none;" data-nodetype="text" id="labelcontainer0">
               <div class="ctrl_container">
                 <div class="button_remove"></div>
               </div>
               <textarea  class="label-input" placeholder="Название"></textarea>
             </div>
          </div>
          <textarea id="jsonOutput"></textarea>
        </div><!-- app -->
    </section><!-- main -->
<!--
```

```
    <section class="main-footer">
        <div class="container">
            <div class="footer-content">
                <p>Дипломный проект 2022 </p>
            </div>
        </div>
    </section>
-->
    <script src="vendor/jq.min.js"></script>
    <script src="vendor/jq-ui.js"></script>
    <script src="vendor/jq-jsplumb.min.js"></script>
    <script src="js/app.js"></script>
</body>
</html>
```

Файл login.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Дипломная работа</title>
    <link rel="stylesheet" href="css/style.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;
1,100;1,300;1,400;1,500;1,700;1,900&display=swap" rel="stylesheet">
</head>
<body>
<section class="main-header">
    <div class="container">
        <div class="main-header-content">
            <div class="logo" onclick="location.href='/'">GONAGO</div>
            <ul class="main-menu">
                <li><a href="#">Главная</a></li>
                <li><a href="#">О проекте</a></li>
```

```
            </ul>
            <a href="login.html" class="btn header-btn">Войти</a>
        </div><!-- main-header-content -->
    </div><!-- container -->
</section><!-- main-header -->
<section class="main">
    <div class="container">
        <div class="main-content">
            <div class="login-form">
                <h1 class="login-title">Вход в систему</h1>
                <p class="login-error">Логин не верен!</p>
                <input id="login" type="text" placeholder="Введите логин">
                <p class="pass-error">Пароль не верен!</p>
                <input id="pass" type="password" placeholder="Введите пароль">
                <a href="#" onclick="auth();" class="btn login-btn">Войти</a>
            </div><!-- login-form -->
        </div><!-- main-content -->
    </div><!-- container -->
</section><!-- main -->
<section class="main-footer">
    <div class="container">
        <div class="footer-content">
            <p>ВКР, 2022 </p>
        </div>
    </div><!-- container -->
</section><!-- main-footer -->
<script>
function auth() {
        let login = document.querySelector('#login');
        let pass = document.querySelector('#pass');
         let errorLogin = document.querySelector('.login-error');
         let errorPass = document.querySelector('.pass-error');
        if (login.value == "ivan" && pass.value == "pass") {
            window.location.href = 'app.html';
```

```
    } else {
        login.style.borderColor = "red";
        pass.style.borderColor = "red";
        errorLogin.style.display = "flex";
        errorPass.style.display = "flex";
    }
  }
</script>
</body>
</html>
```