**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Архитектура программного модуля**

Архитектура программного обеспечения (software architecture) – это представление, которое даёт информацию о компонентах ПО, обязанностях отдельных компонентов и правилах организации связей между компонентами. Необходимость архитектуры обоснована сложностью программного обеспечения. Продуманная архитектура облегчает разработку и дальнейшее развитие ПО. Она служит базисом, каркасом создаваемой системы, интегрируя отдельные компоненты и создавая высокоуровневую модель системы.

Набор принципов, используемых в архитектуре, формирует архитектурный стиль (software architecture style). Применение архитектурного стиля сродни употреблению шаблона проектирования, но не на уровне компонента (модуля или класса), а на уровне всей создаваемой системы ПО. Как и шаблоны проектирования, архитектурные стили упрощают коммуникацию разработчиков и предлагают готовые решения целого класса абстрактных проблем. В таблице 3.1 представлено короткое описание основных архитектурных стилей.

Многоуровневая архитектура (multilayered architecture) сосредоточена на иерархическом распределении отдельных частей системы при помощи эффективного разделения отношений.

Таблица 3.1 – Основные архитектурные стили

|  |  |
| --- | --- |
| Архитектурный стиль | Описание |
| Клиент-серверная модель | Разделение системы на два приложения – клиент и сервер. При работе клиент посылает запросы на обслуживание серверу |
| Компонентная архитектура | Деление системы на компоненты, которые могут быть повторно использованы и не зависят друг от друга. Каждый компонент снабжается известным интерфейсом для  коммуникаций |
| Многоуровневая архитектура | Разделение функций приложения на группы (уровни), которые организованы в виде стекового набора |
| Шина сообщений | Система, которая может посылать и передавать информационные сообщения в определённом формате по общему  коммуникационному каналу. Благодаря этому организуется взаимодействие систем без указаний конкретных получателей сообщений |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Многозвенная архитектура | Разделение функций подобно многоуровневой архитектуре, но группировка происходит не только на логическом, а и на физическом уровне – отдельным группам соответствует отдельный компьютер (сервер, кластер) |
| Объектно-ориентированная  архитектура | Представление системы в виде набора взаимодействующих объектов |
| Выделенное представление | Выделение в системе отдельных групп функций для взаимодействия с пользователями и обработки данных |
| Архитектура, ориентированная на сервисы | Каждый компонент системы представлен в виде независимого сервиса, предоставляющего свои функции по  стандартному протоколу |

Каждая часть соотносится с определённым уровнем (layer), для каждого уровня заданы выполняемые им функции, уровни выстроены в стековую структуру (то есть находятся один поверх другого). Например, типичная многоуровневая архитектура веб-приложения включает уровень представления (компоненты пользовательского интерфейса), уровень бизнес-логики (обработка данных) и уровень доступа к данным [4]. При этом уровень представления считается высшим, за ним идёт уровень бизнес-логики, а за уровнем бизнес-логики – уровень доступа к данным.

Основные принципы многоуровневой архитектуры:

* проектирование чётко устанавливает разграничение функций между уровнями;
* нижние уровни независимы от верхних уровней;
* верхние уровни вызывают функции нижних уровней, но при этом взаимодействуют только соседние уровни иерархии.

Использование многоуровневой архитектуры обеспечивает следующие преимущества:

1. Изоляция. Разработка и обновление ПО могут быть изолированы рамками одного уровня.
2. Производительность. Распределение уровней на отдельные физические компьютеры повышает производительность и отказоустойчивость.
3. Тестируемость. Уровни допускают независимое тестирование.

Многоуровневая архитектура активно применяется при создании бизнес-приложений и сайтов, особенно приложений масштаба предприятия. При этом обычно используется следующий набор уровней (см. рисунок 3.1):

1. Уровень представления (presentation layer) ответственен за взаимодействие с пользователем, ввод и вывод информации.
2. Бизнес-уровень или уровень бизнес-логики (business logic layer) обрабатывает информацию, реализуя конкретные бизнес-правила.
3. Уровень доступа к данным (data access layer) обеспечивает загрузку и сохранение информации, используя источник данных (файл, база данных) или внешний сервис.

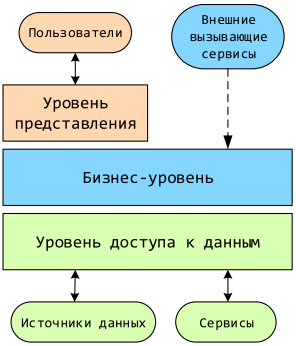


Рисунок 3.1 – Типичные уровни бизнес-приложения

**3.2 Структура программного модуля**

В процессе проектирования при решении всех требуемых задач проект был разделен на относительно большие части, взаимодействующие между собой и выполняющие свои особые функции (см. рисунок 3.2).

Рассмотрим структурные модули:

1. Модуль организации пользовательского интерфейса. При помощи этого модуля пользователь может осуществлять базовую навигацию на сайте: выбирать интересующие аукционы, знакомиться с правилами его проведения, просматривать данные своего аккаунта. Область ответственности этого модуля - обработка ввода информации от пользователя, а также формирование нужных запросов сетевым модулям.



Рисунок 3.2 – Модули приложения

Обобщенная структурная схема программного средства изображена на рисунке 3.3.



Рисунок 2.3 – Обобщенная структурная схема

1. Препроцессор Sass используется для компиляции css-кода ответственного за отображение элементов пользовательского интерфейса (кнопки, списки, текст и т.д.).
2. Модуль сервисных интерфейсов является прослойкой между внутренней логикой приложения и внешней стороной, которая видна пользователю. Обеспечивает возможность гибкой перестройки внутренностей модуля, не затрагивая пользовательский интерфейс.
3. Модели уровня бизнес-логики являются абстракцией первого уровня над моделями базы данных. Они содержат в себе первичные правила и способы хранения и обработки записей более низкого уровня абстракции. Служат вспомогательным инструментом, не допускающим рост сложности работы с БД при развитии приложения.
4. Сервисы уровня бизнес-логики отвечают за передачу данных от БД к пользовательскому интерфейсу и обратно. В дополнении происходит проверка последних. При добавлении функциональных возможностей архитектура легко расширяется с учетом новых требований.
5. Уровень доступа к базе данных. Отвечает за хранение, создание, редактирование информации. Реализован с помощью MSSQL Server с применением *объектно-реляционного отображателя* (object relational mapper – ORM) Entity Framework. Адаптеры являются посредниками между уровнями области определения и распределения данных (domain and data mapping layers). Объекты-клиенты создают описание запроса декларативно и направляют их к объекту-репозиторию (Repository) для обработки. Объекты могут быть добавлены или удалены из репозитория, как будто они формируют простую коллекцию объектов. А код распределения данных, скрытый в объекте Repository, позаботится о соответствующих операциях незаметно для разработчика. В двух словах, паттерн Repository инкапсулирует объекты, представленные в хранилище данных и операции, производимые над ними, предоставляя более объектно-ориентированное представление реальных данных. Repository также преследует цель достижения полного разделения и односторонней зависимости между уровнями области определения и распределения данных.

**3.3 Алгоритмы, положенные в основу программного средства**

**3.3.1** Алгоритм регистрации пользователя

После ввода в адресной строке URL-адреса веб-портала пользователю предоставляется ограниченная функциональность. Для получения расширенных возможностей необходимо создать учётную запись.

Учётная запись — хранимая в [компьютерной системе](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) совокупность [данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5) о пользователе, необходимая для его опознавания ([аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F)) и [предоставления доступа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B0%D0%B2%D0%B0_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%B0) к его личным данным и настройкам. Для использования учётной записи (другими словами, для входа в систему под чьим-то именем) обычно требуется ввод имени и [пароля](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D0%BB%D1%8C). Также может требоваться другая дополнительная информация. Пользователи [портала](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) могут воспринимать учётную запись как личную страничку, профиль, кабинет, место хранения личных и других сведений на интернет-ресурсе.

Для создания учётной записи пользователю необходимо зарегистрироваться. Регистрация производится путём корректного ввода: имени, почты и пароля.

Выполнение алгоритма начинается с перехода пользователя на страницу регистрации и ввода данных в доступные поля. Алгоритм начинается с получения пользовательских данных, таких как электронная почта, имя, пароль. После получения данных их нужно провалидировать. Если в пользовательском браузере присутствует JavaScript, то появляется возможность валидации на стороне клиента, это означает, что проверка правильности ввода будет осуществляться непосредственно при вводе значений, без перезагрузки страницы. Данные прошедшие валидацию на стороне клиента отправляются на сервер, там они должны пройти серверную валидацию, которая по логике не должна отличаться от клиентской.

После прохода блока валидации данные должны быть приведены к пользовательской сущности и записаны в базу данных. На этапе приведения происходит шифрования пароля, запоминание текущей даты для пользовательского поля «Дата регистрации», создаётся авторизованная пользовательская сущность, которая определяет права и уникальность пользователя в системе.

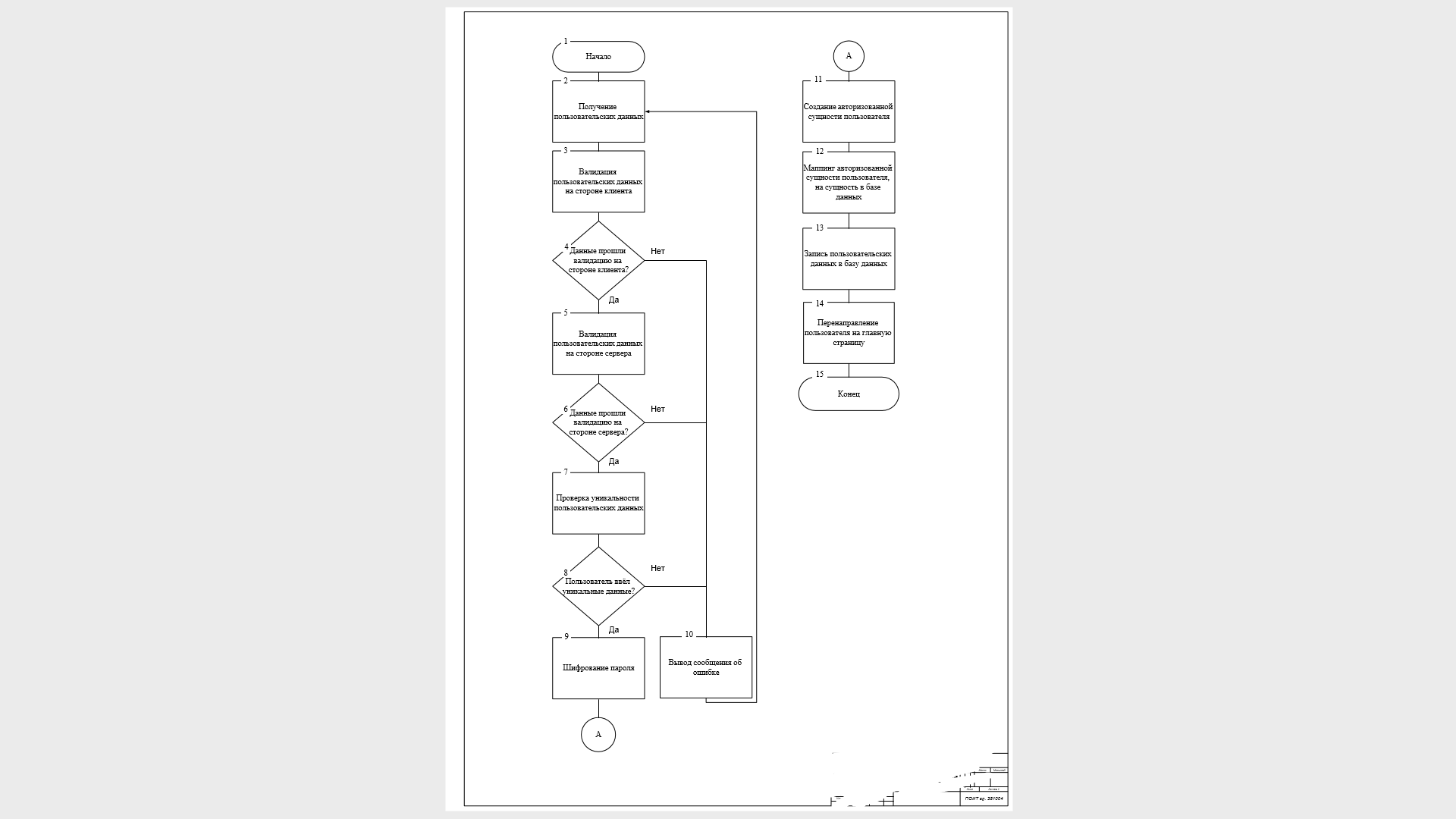


Рисунок 3.2 - Схема алгоритма регистрации пользователя

Для избежание пробелов в безопасности пользовательские данные подвергаются валидации. В разрабатываемом программном средстве используется двухэтапная валидация. Такая валидация подразумевает валидацию при непосредственном вводе информации в поля, а также валидацию подтверждении регистрации. В таблице 3.5 предоставлены требования к регистрационным полям.

Таблица 3.5 – Валидационные требования к регистрационным полям

|  |  |  |
| --- | --- | --- |
| Название | Описание | Требование |
| Email | Электронная почта пользователя | * уникальное; * длина от 5 до 30 символов; * более одного символа до знака @; * знак @; * доменное имя. |
| Name | Имя пользователя. Это имя будет видно другим пользователям | * уникальное; * длина от 3 до 15 символов; * только буквенные символы; * не более двух повторяющихся символов подряд. |
| Password | Пользовательский пароль | * длина от 5 до 15 символов; * более одного цифрового символа. |
| Confirm password | Подтверждения пользовательского пароля | * длина от 5 до 15 символов; * более одного цифрового символа; * совпадение с полем Password. |

**3.3.2** Алгоритм сохранения тегов

Данный алгоритм начинает выполняться при создании статьи пользователем. После получения сервером объекта, содержащим информацию о статье, необходимо поля, содержащие текстовую информацию, проанализировать на наличие тегов. Каждая статья состоит из двух основных текстовых частей: заголовок и основная часть. В соответствии с требованиями теги должны быть выделены в обоих частях. Для этого будет произведено слияние заголовка и основной части текста.

Чтобы ускорить процесс анализирования тегов в тексте, из текста должны быть удалены все ненужные части, то есть должны остаться только полезные для анализа слова. После удаления ненужных частей из текста формируется список слов. В этом списке циклически ищутся слова, начинающиеся со знака #, что означает начало тега.

Все найденные теги должны быть сохранены в базу данных. Могут возникнуть случаи, когда тег из созданной статьи уже имеется в базе данных, чтобы не хранить дубликаты каждый тег проверяется на уникальность, вследствие в базу данных запишутся только новые теги.

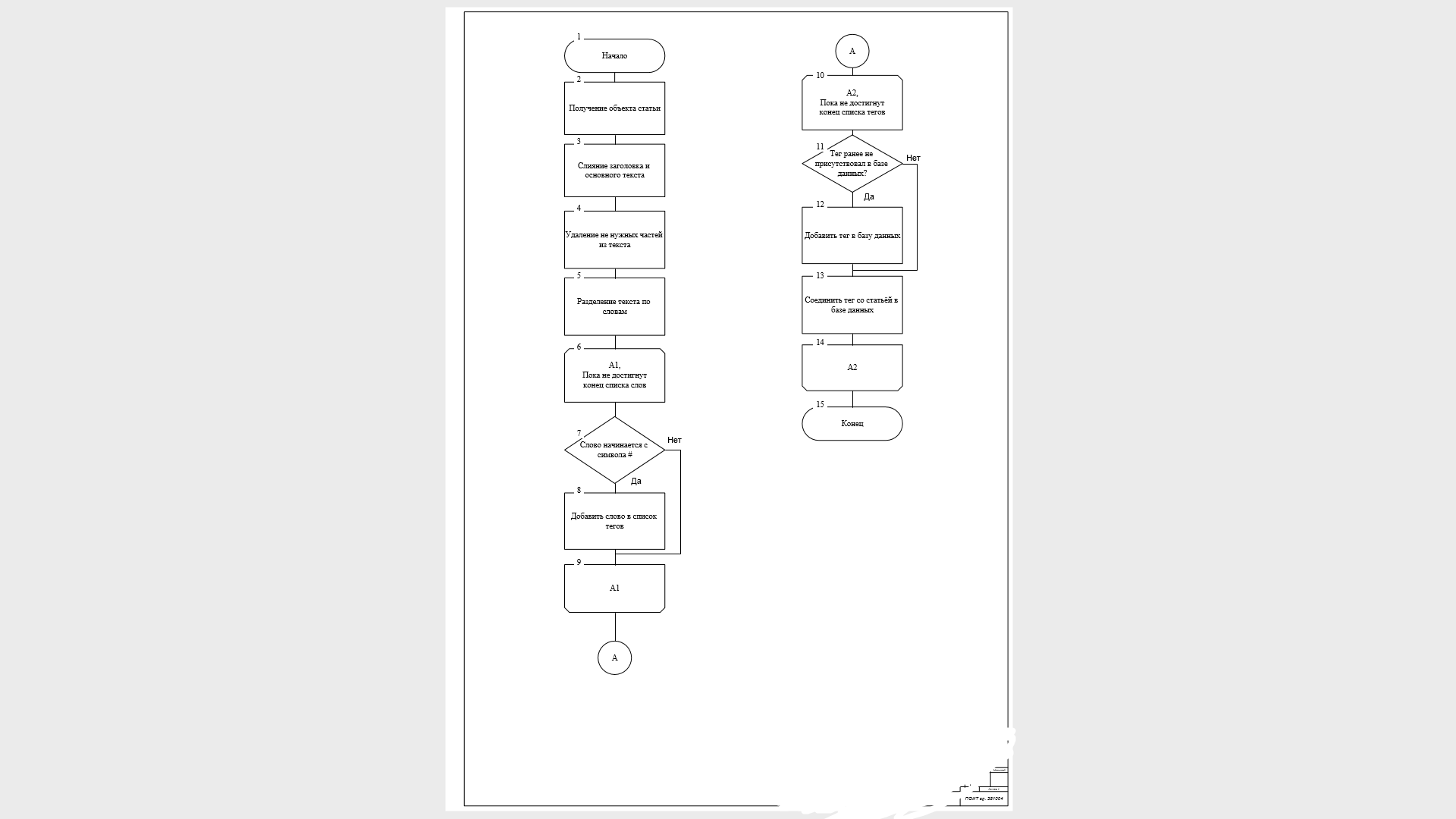


Рисунок 3.3 – Схема алгоритма сохранения тегов