МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

WEB-приложение «TipaHabr»

Выполнил студент Храмых Владислав Олегович

(Ф.И.О.)

Руководитель проекта преп.-стаж. Сенюк В.К.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Сенюк В.К.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Сенюк В.К.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[Введение 3](#_Toc41676336)

[1. Постановка задачи 4](#_Toc41676337)

[1.1. Алгоритмы решения 4](#_Toc41676338)

[1.2. Обзор прототипа 5](#_Toc41676339)

[2. Разработка архитектуры проекта 7](#_Toc41676340)

[2.1. Обобщенная структура 7](#_Toc41676341)

[2.2. Взаимосвязь всех компонентов 8](#_Toc41676342)

[2.3. Диаграмма UML 9](#_Toc41676343)

[3. Разработка функциональной модели и модели данных ПС 11](#_Toc41676344)

[3.1. Модель базы 12](#_Toc41676345)

[3.2. Выполняемые функции 13](#_Toc41676346)

[3.3. Регистрация и авторизация 15](#_Toc41676347)

[4. Руководство пользователя 16](#_Toc41676348)

[4.1. Регистрация пользователя 16](#_Toc41676349)

[4.2. Аутентификация и авторизация пользователя 17](#_Toc41676350)

[4.3. Просмотр постов 19](#_Toc41676351)

[4.4. Админская часть 21](#_Toc41676352)

[5. Тестирование 23](#_Toc41676353)

[Заключение 25](#_Toc41676354)

[Список используемых источников 26](#_Toc41676355)

[Приложение А 27](#_Toc41676356)

[Приложение Б 30](#_Toc41676357)

[Приложение В 31](#_Toc41676358)

# **Введение**

В данном курсовом проекте разработано WEB-приложение «TipaHabr». Логически оно разделено на две части: серверную, написанную на Node.js (Express), и клиентскую, в которой использовался язык JavaScript с внедрением библиотеки Vue.js.

В наше время очень сильно возрастает интерес к ИТ сфере, поэтому многие будут встречаться с проблемами на пути становления Senior. Поэтому я решил создать такой сервис, на котором люди смогут делиться своими решениями некоторых проблем. Также здесь можно выкладывать посты и не только на IT тематику, но основная масса постов будет именно про IT.

Целью курсового проекта является разработка WEB-приложения, которое поможет вам осуществить как размещение своего поста, так и просмотр других. Я выбрал эту тему, потому что она очень актуальна в наше время. Также одной из причин выбора, является то, что каждый программист посещает такие ресурсы и появление ещё одного это несомненно плюс.

Пояснительная записка курсового проекта состоит из 30 страниц, 22 рисунков, 3 приложений, 9 источников литературы.

В первом разделе рассматриваются основные технологии, которые использовались в разработке данного приложения, а также его прототипы и актуальность задачи.

Во втором разделе описана архитектура курсового проекта.

В третьем разделе предоставлена информация о разработанных объектах базы данных.

Четвертый раздел содержит руководство пользователя для разработанного клиентского приложения.

В пятом разделе представлены результаты тестирования приложения.

В заключении описывается результат курсового проектирования и задачи, которые были решены в ходе разработки приложения.

1. **Постановка задачи**

Главная задача курсового проекта является разработка WEB- приложения, которое позволяет пользователю просматривать посты на разные тематики. Также пользователь может зарегистрироваться и опубликовать свой пост. Также присутствует сортировка по категориям. В данном курсовом проекте требовалось реализовать следующие задачи:

* сохранение информации в базе данных;
* организация регистрации и входа в систему;
* реализация архитектуры MVC;
* обмен данными между клиентом и сервером должен производиться по протоколу HTTPS;
* возможность просмотра, добавления постов;

## **Алгоритмы решения**

При реализации курсового проекта использовались технологии Node.js[1], фреймворк Express[2], ORM библиотека sequelize[3], WebSoket[4] для реализации back-end, для front-end части Bootstrap [5], JavaScript[6], [Vue](http://handlebarsjs.com/).js[7], CSS. В качестве централизованного хранилища на стороне клиента использовалась Vuex[8] В качестве базы данных использовалась СУБД MSSQL.

Node.js –  программная платформа, основанная на движке [V8](https://ru.wikipedia.org/wiki/V8_(%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA_JavaScript)) (транслирующем [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript) в [машинный код](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript) взаимодействовать с устройствами [ввода-вывода](https://ru.wikipedia.org/wiki/%D0%92%D0%B2%D0%BE%D0%B4-%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4) через свой [API](https://ru.wikipedia.org/wiki/API) (написанный на [C++](https://ru.wikipedia.org/wiki/C%2B%2B)), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль [веб-сервера](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80). В основе Node.js лежит [событийно-ориентированное](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [асинхронное (или реактивное)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программирование с [неблокирующим вводом/выводом](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B0%D1%8F_%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F).

Express – [фреймворк web-приложений](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для Node.js, реализованный как [свободное и открытое программное обеспечение](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%B8_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) под [лицензией MIT](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_MIT). Он спроектирован для создания [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и [API](https://ru.wikipedia.org/wiki/API).

Sequelize– это ORM-библиотека для приложений на Node.js, которая осуществляет сопоставление таблиц в базе данных и отношений между ними с классами. При использовании Sequelize мы можем не писать SQL-запросы, а работать с данными как с обычными объектами.

WebSoket – это технология, которая позволяет создавать интерактивное соединение между клиентом и сервером для обмена сообщениями в режиме реального времени. Веб-сокеты, в отличие от HTTP, позволяют работать с двунаправленным потоком данных, что делает эту технологию совершенно уникальной.

Bootstrap – свободный набор инструментов для создания сайтов и веб-приложений, который включает в себя HTML и CSS – шаблоны оформления для веб-форм, кнопок, меток, слайд шоу, блоков навигации и прочих компонентов веб-интерфейса.

JavaScript – язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Vue.js ­­– JavaScript-фреймворк с открытым исходным кодом для создания пользовательских интерфейсов. Легко интегрируется в проекты с использованием других JavaScript-библиотек. Может функционировать как веб-фреймворк для разработки одностраничных приложений в реактивном стиле.

Vuex – паттерн управления состоянием + библиотека для приложений на Vue.js. Он служит централизованным хранилищем данных для всех компонентов приложения с правилами, гарантирующими, что состояние может быть изменено только предсказуемым образом. Vuex интегрируется с официальным расширением [vue-devtools](https://github.com/vuejs/vue-devtools" \t "_blank), предоставляя «из коробки» такие продвинутые возможности, как «машину времени» для отладки и экспорт/импорт слепков состояния данных.

## **Обзор прототипа**

Найти прототипы, подходящие под наш проект в интернете – не проблема.

В качестве прототипа был выбран сервис «Habr», который представлен на рисунке 1.1. Это приложение предназначено для осуществления поиска решений проблем, связанных с различными тематиками, а также не только решений проблем: там публикуются достаточно интересные посты на другие темы.

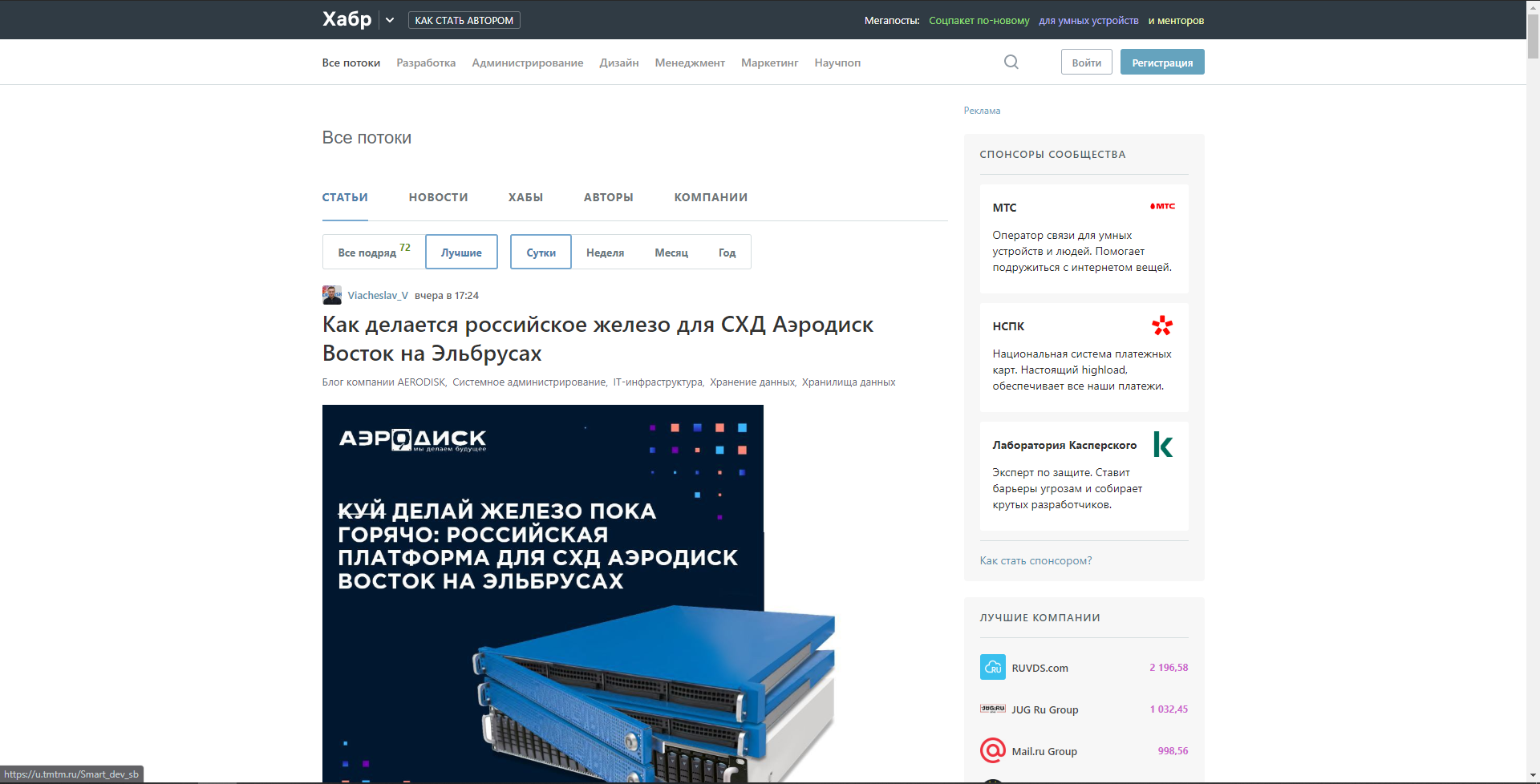


Рисунок 1.1 – Интернет-ресурс «Habr»

Еще один интернет-ресурс помог мне определиться: что именно я буду делать. WEB-приложение – это ресурс «Stackoverflow» (рисунок 1.2).

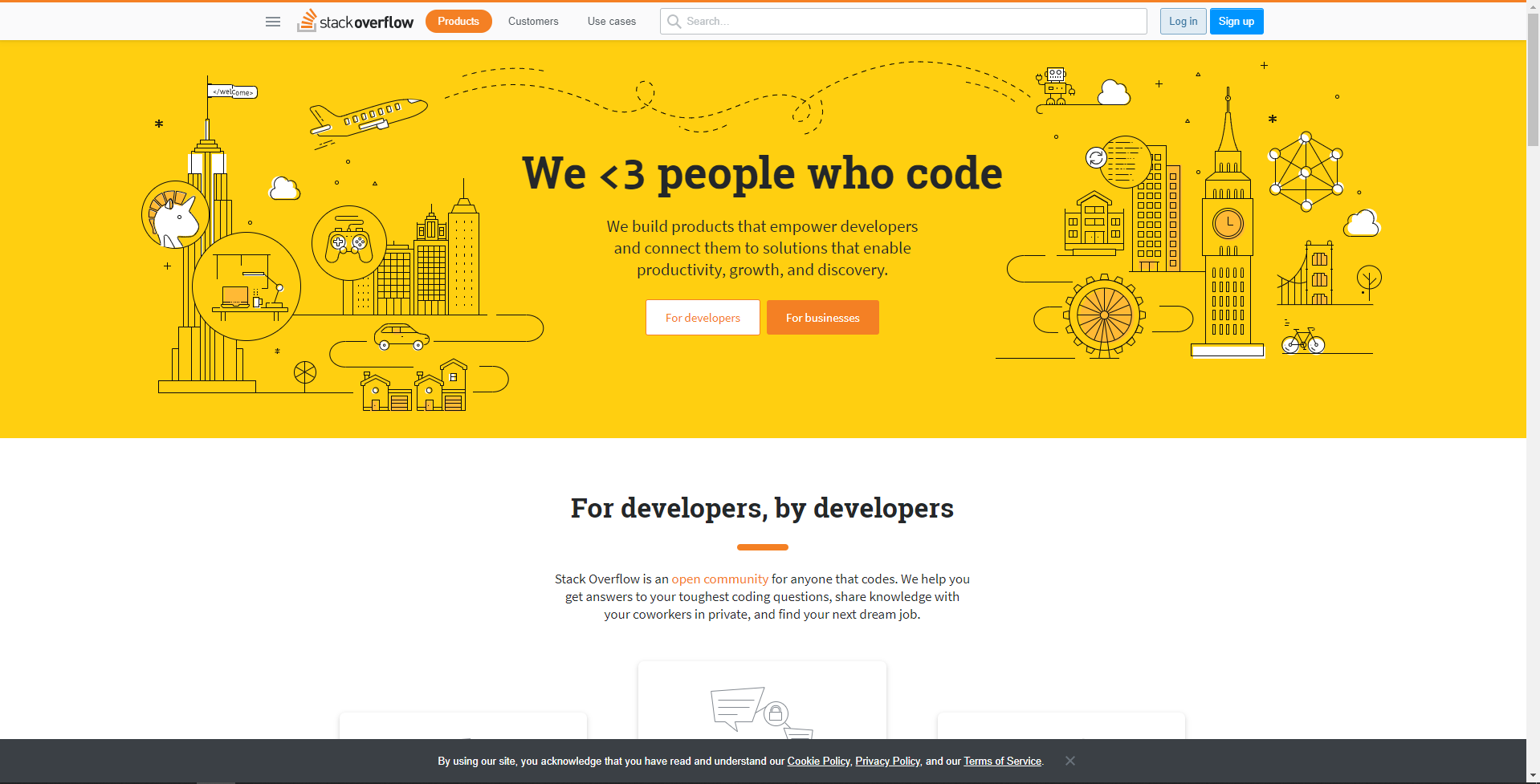


Рисунок 1.2 – Интернет-ресурс «Stackoverflow»

# **Разработка архитектуры проекта**

Разработка архитектуры проекта – одна из важнейших задач в процессе работы над приложением, так как в зависимости от неё определяется уровень зависимости компонентами приложения, и насколько легко расширяемы её составные части. Хорошая архитектура – залог хорошего и быстрого приложения.

Архитектура проекта – это его строение приложения, это то, как должны быть определены компоненты приложения, и как оны должны быть зависимы, чтобы приложение работало хорошо, т.е. представление программного средства как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью программного средства, на котором реализуется принцип выделения относительно независимых компонент.

Основные задачи разработки архитектуры проекта:

* Выделение программных компонентов и обеспечение «слабосвязного» кода;
* Определение способов взаимодействия между выделенными программными компонентами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

## **Обобщенная структура**

Курсовой проект построен на многоуровневой архитектуре [9]. Данная архитектура основана на следующих принципах:

* 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.

Для данного курсового проекта были разработаны следующие уровни:

* Уровень «Модель» – описание таблиц в базе данных;
* Уровень «Представление» – отвечает за отображение данных модели пользователю, реагируя на изменения модели;
* Уровень «Контроллер» – описание классов, методы которых предназначены для создания ответа пользователю по его запросу.
* Уровень «Роутер» – направление запроса по URL туда, где происходит обработка запросов именно такого типа.
* Уровень «Авторизация» – назначение пользователю прав и привилегий для работы с объектами базы данных.
* Уровень «Аутентификация» – проверка подлинности пользователя.

Каждый запрос клиента проходит как минимум через 3 уровня («Аутентификация», «Авторизация» и «Контроллер»), после чего клиент получает ответ. В случае успешной аутентификации и авторизации клиента сервер также может задействовать другие уровни приложения.

В приложении используется паттерн MVC:

Паттерн MVC является одним из распространенных паттернов, применяемых в веб-приложениях. В том числе он применяется и в приложениях на Node.js.

Паттерн MVC включает ряд компонентов:

* Модели определяют структуру и логику используемых данных
* Представления (views) определяет визуальную часть, как данные будут отображаться
* Контроллеры обрабатывают входящие http-запросы, используя для обработки модели и представления, и отправляет в ответ клиенту некоторый результат обработки, нередко в виде html-кода, но также и виде JSON.
* Система маршрутизация (router) как дополнительный компонент сопоставляет запросы с маршрутами и выбирает для обработки запросов определенный контроллер.

В общем случае, когда к приложению приходит запрос, система маршрутизации выбирает нужный контроллер для обработки запроса. Контроллер обрабатывает запрос. В процессе обработки он может обращаться к данным через модели и для рендеринга ответа использовать представления. Результат обработки контроллера отправляется в ответ клиенту. Нередко ответ представляет html-страницу, которую пользователь видит в своем браузере.

## **Взаимосвязь всех компонентов**

В решении курсового проекта была использована n-layer архитектура. Представлена на рисунке 2.1. Эта архитектура выбрана, так как она идеально подходит для приложений с длительным жизненным циклом и сложной бизнес логикой. Для реализации серверной части была использована N-Layer архитектура. Ее схематическое представление продемонстрировано на рисунке 2.1. Данный курсовой проект построен по примеру ниже приведенной схемы классической трехуровневой системы.

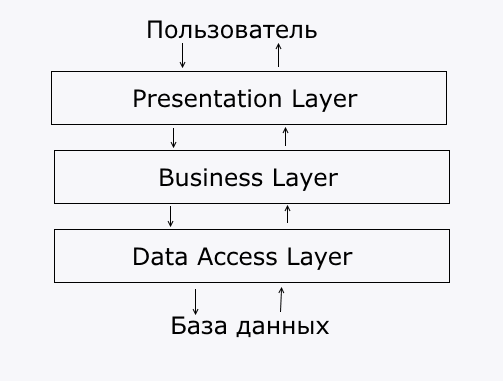


Рисунок 2.1 – Общая схема N-Layer архитектуры

Data Access Layer (уровень доступа к данным) содержит сущности, соответствующие таблицам базы данных, а также схемы их взаимодействия.

Business Layer (уровень бизнес-логики) при тесном взаимодействии с Service Layer (уровень сервисов) являются посредниками между сущностями базы данных и моделями для UI, сочетая в себе сервисы и классы, реализующий основной функционал всего разрабатываемого приложения.

Presentation Layer (уровень представления), поддерживаемый UI Components (User Interface Components – компоненты пользовательского интерфейса), представляет собой генерирующуюся на сервере разметку или получаемые с уровня бизнес-логики данные для REST API в формате JSON, преобразующиеся впоследствии в разметку на стороне клиента. Уровень представления состоит состоящее из контроллеров, координирующих пользовательские запросы, и представлений**.**

## **Диаграмма UML**

ДиаграммаUML–это графическое представление набора элементов,изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграмма использования представлена на рисунке 2.2.

В языке UML вариант использования изображается в виде овала, помеченного именем представляемого варианта. Варианты использования могут быть связаны с участвующими в них действующими лицами (actors), изображаемыми в виде человечков и представляющими различные роли пользователей системы или внешние системы, взаимодействующие с ней.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship). Действующие лица также могут быть связаны друг с другом с помощью связей обобщения (generalization).

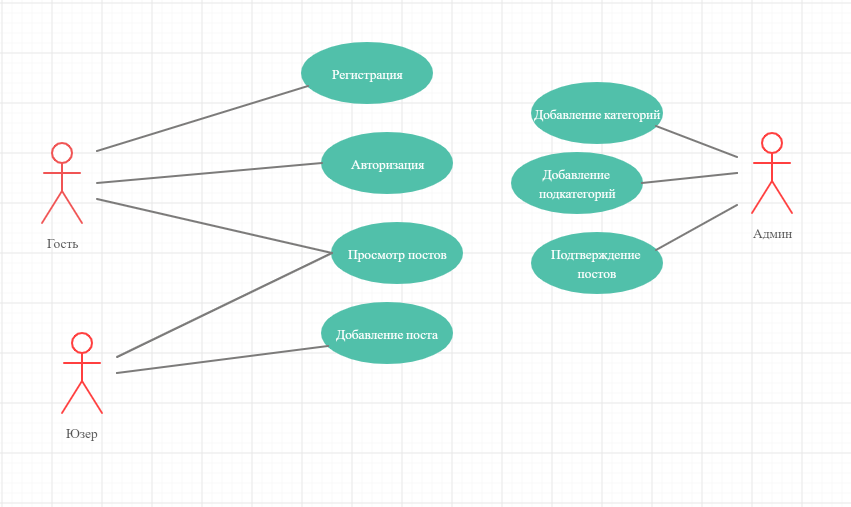


Рисунок 2.2 – Диаграмма использования

1. **Разработка** **функциональной модели и модели данных ПС**

Для разработки и управления базой данных курсового проекта использовалась объектно-реляционная система управления базами данных «MS SQL Server».

Выбор данного системного продукта произошел по нескольким очевидным причинам: легкая установка, дизайн оболочки IDE SQL Management Studio, бесплатна для разработки, раннее полученные знания по использованию данной базы данных.

База данных — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных.

Microsoft SQL Server – система управления реляционными базами данных, разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Основные функции СУБД:

* определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;
* предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);
* обеспечение логической и физической независимости данных;
* защита логической целостности базы данных;
* защита физической целостности;
* управление полномочиями пользователей на доступ к базе данных;
* синхронизация работы нескольких пользователей;
* управление ресурсами среды хранения;
* поддержка деятельности системного персонала.

Примеры СУБД: Oracle, MS SQL Server, Microsoft Access, MySQL, PostgreSQL и так далее.

## **Модель базы**

Для базы данных HABR было разработано 4 таблиц. На рисунке 3.1 предоставлена структура связей внешних ключей и первичных ключей.

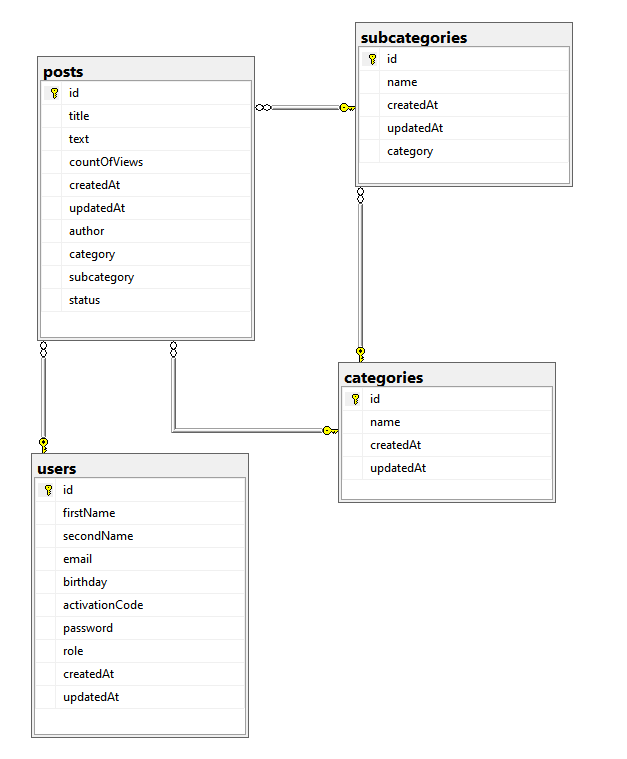


Рисунок 3.1 – Структура базы данных

## **Выполняемые функции**

На рисунке 3.2 представлена структура таблицы *users*, которая содержит информацию о пользователях системы.

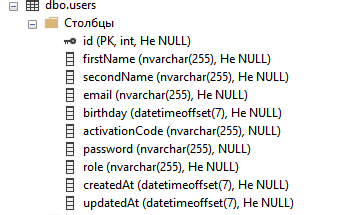


Рисунок 3.2 – Таблица users

Таблица включает поля:

* id – первичный ключ, идентификатор пользователя;
* firstName– имя пользователя в системе;
* lastName – фамилия пользователя;
* email–адрес электронной почты пользователя;
* birthday – дата рождения пользователя;
* acticationCode – код активации email;
* password – хеш пароля пользователя;
* role– роль пользователя;
* createdAt –дата регистрации пользователя;
* updatedAt– дата последнего обновления профиля.

На рисунке 3.3 представлена структура таблицы posts, которая содержит информацию о посте.

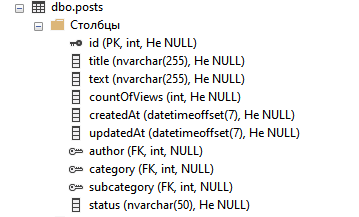


Рисунок 3.3 – Таблица posts

Таблица включает поля:

* id – первичный ключ, идентификатор поста;
* title – название поста;
* text – основное содержимое поста;
* countOfViews– количество просмотров поста;
* createdAt – дата создания;
* updatedAt – дата последнего обновления;
* category – вторичный ключ. Id категории поста;
* subcategory – вторичный ключ. Id подкатегории поста;
* Status – статус поста.

На рисунке 3.4 представлена структура таблицы categories, которая содержит информацию о продукте, а именно добавлен ли продукт в корзину, добавлен ли продукт в избранное.

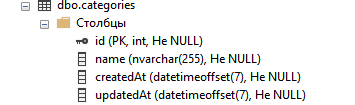


Рисунок 3.4 – Таблица categories

Таблица включает поля:

* id – первичный ключ, идентификатор таблицы categories;
* name – название категории;
* createdAt – дата создания;
* updatedAt – дата последнего обновления.

На рисунке 3.5 представлена структура таблицы subcategories, которая содержит информацию о товарах.

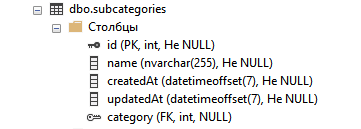


Рисунок 3.5 – Таблица subcategories

Таблица включает поля:

* id – первичный ключ, идентификатор подкатегории;
* name – название подкатегории;
* createdAt– дата добавления;
* updatedAt– дата последнего обновления;
* category – вторичный ключ. id категории.

## **Регистрация и авторизация**

Регистрация пользователя происходит в разделе Registration, листинг которого находится в приложении Б. При регистрации предусмотрена валидация данных. Логин и пароль пользователя уникальны. Каждый пользователь сначала должен зарегистрироваться, после чего он сможет авторизоваться.

Авторизация пользователя происходит на странице Login, листинг приведен в приложение В. Также предусмотрена валидация данных.

1. **Руководство пользователя**

Данная глава содержит описание некоторых функций приложения для более легкого восприятия конечного пользователя.

* 1. **Регистрация пользователя**

Для регистрации необходимо заполнить форму, которая продемонстрирована на рисунке 4.1. Код регистрации со стороны backend есть в Приложении Б, а frontend части в приложении А.

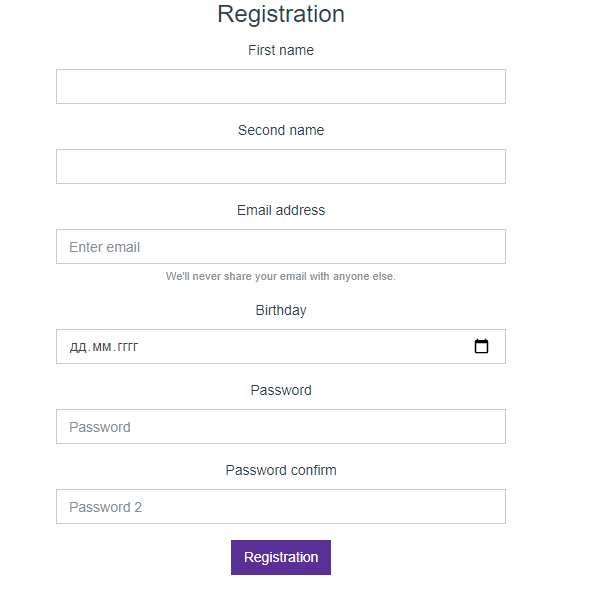


Рисунок 4.1 – Форма регистрации

После отправки пользователем запроса на регистрацию, происходит валидация всех полей формы. Если валидация будет неуспешной, пользователь увидит сообщения об ошибках в соответствующих полях для ввода информации. В случае успешной регистрации пользователь переадресовывается на страницу входа в приложение.

* 1. **Аутентификация и авторизация пользователя**

Для аутентификации необходимо ввести пароль, логин и нажать на кнопку «Войти», что позволит проверить, существует ли пользователь с введенными данными в системе. Все поля являются обязательными для заполнения.

Если аутентификация прошла успешно и соблюдены все выше описанные правила, то происходит перенаправление на главную страницу, что означает, что пользователь был успешно авторизован. Скриншот формы авторизации представлен на рисунке 4.2.

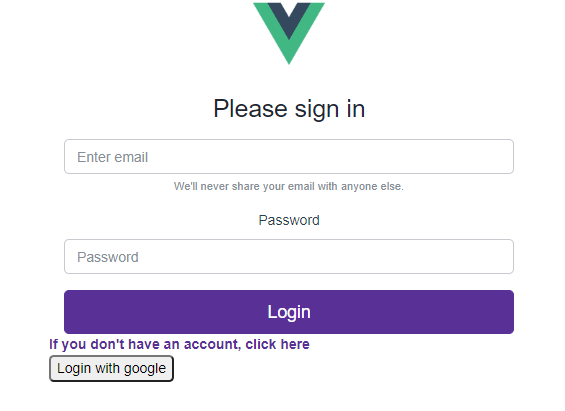


Рисунок 4.2 – Фрагмент экрана с формой для авторизации пользователя

При входе в систему авторизированному пользователю выдается токен, который записывается в localstorage, тем самым позволяя разграничить возможности пользователей на сайте.

В данном курсовом проекте логика приложения задумана так, что пользователь, который не вошел в систему имеет доступ только к просмотру постов. Если пользователь авторизировался, то у него появляется возможность добавления поста.

* 1. **Просмотр постов**

Для того, чтобы пользователь смог увидеть сообщения, ему достаточно зайти на сайт (рисунок 4.3).

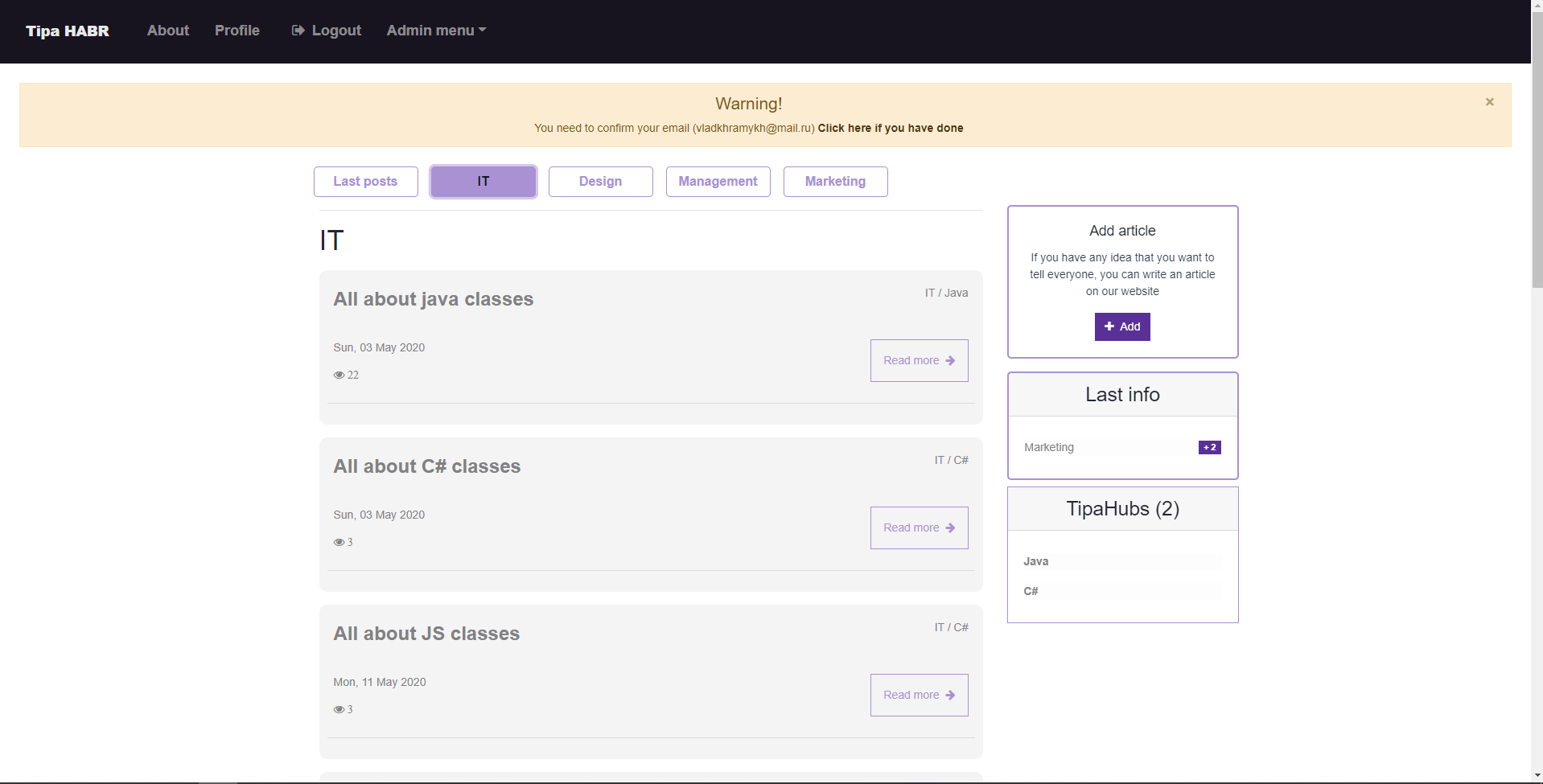


Рисунок 4.3 – Просмотр постов

После того как пользователь авторизовался, он имеет возможность добавлять посты. Кнопка добавления поста находится справа в верхнем углу. После перехода на форму добавления поста, вам нужно заполнить форму. (рисунок 4.4).

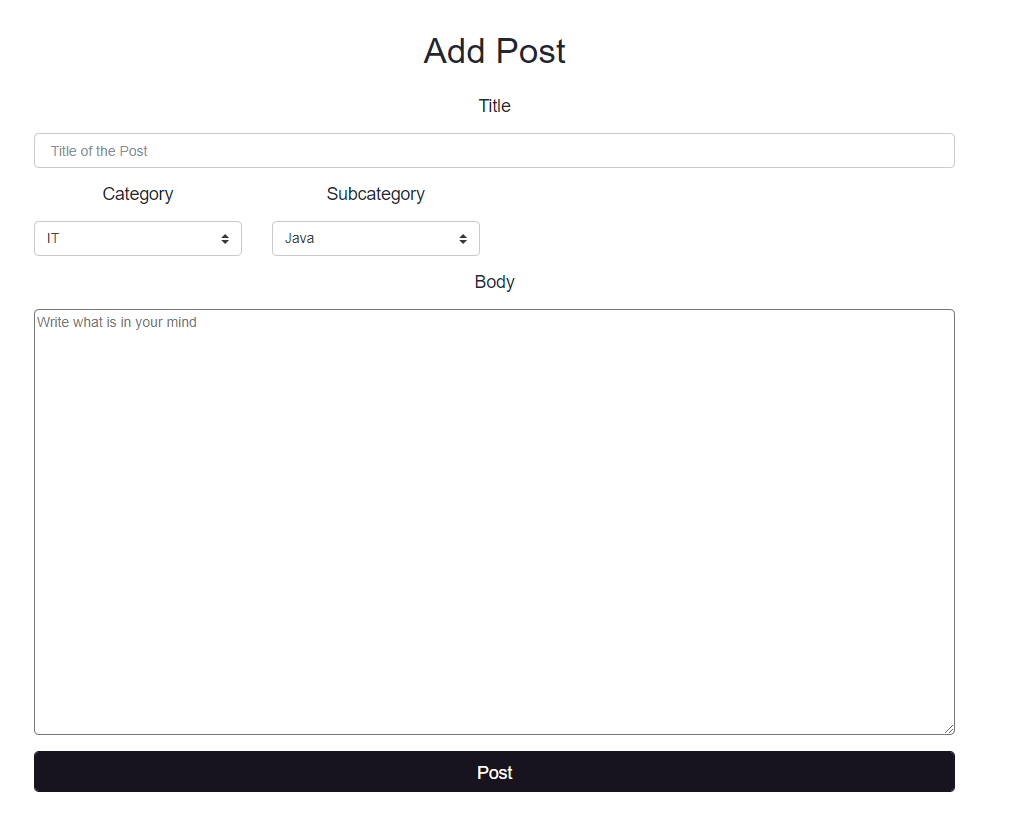


Рисунок 4.4 – Форма для добавления поста

При добавлении можно выделять участки кода, которые нужно отформатировать модератору. 4.5.

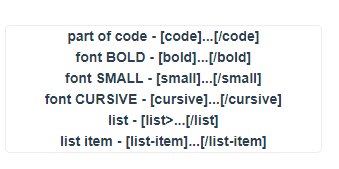


Рисунок 4.5 – Некоторые способы форматирования

Когда все поля буду корректно заполнены пользователем, необходимо нажать на кнопку «Send». Будет показано окно, которое укажет на умпешнюу отправку поста на модерацию (Рисунок 4.6).

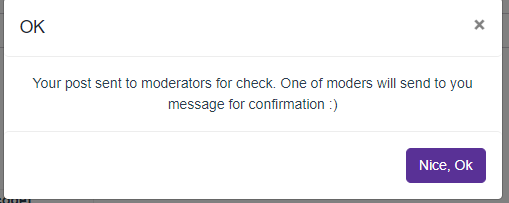


Рисунок 4.6 – Уведомление о том, что пост отправлен

## **Администрирование**

В данном курсовом проекте была разработана возможность администрирования. Админ может добавлять новые категории и подкатегории, а также удалять их. Также он может проверять новые посты на корректность и выполнять форматирования, согласно тому, что было указано выше.

Чтобы зайти под админом, нужно знать учётные данные админа, и после того, как вы войдёте «под» админом, у вас будет открыта панель Admin Menu (рисунок 4.7).

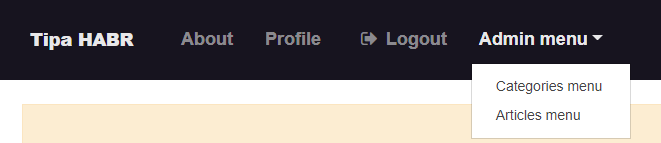


Рисунок 4.7 – Панель админа

После этого вы можете выбрать то, что хотите сделать. Первый элемент выпадающего меню переведёт вас на страницу, на которой вы сможете выполнять манипуляции с категориями и подкатегориями. (рисунок 4.8).

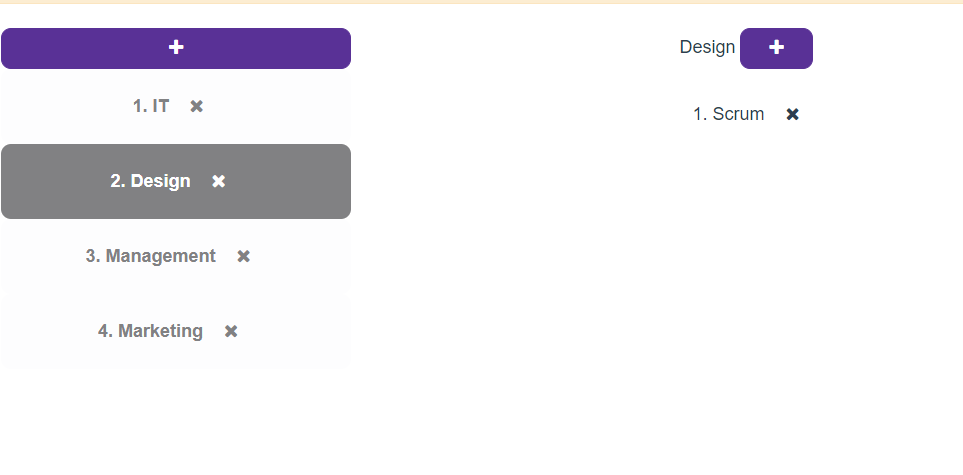


Рисунок 4.8 – Меню категорий (Админ меню)

Также есть возможность проверки и модерации постов. Это второй пункт выпадающего меню (Рисунок 4.9)

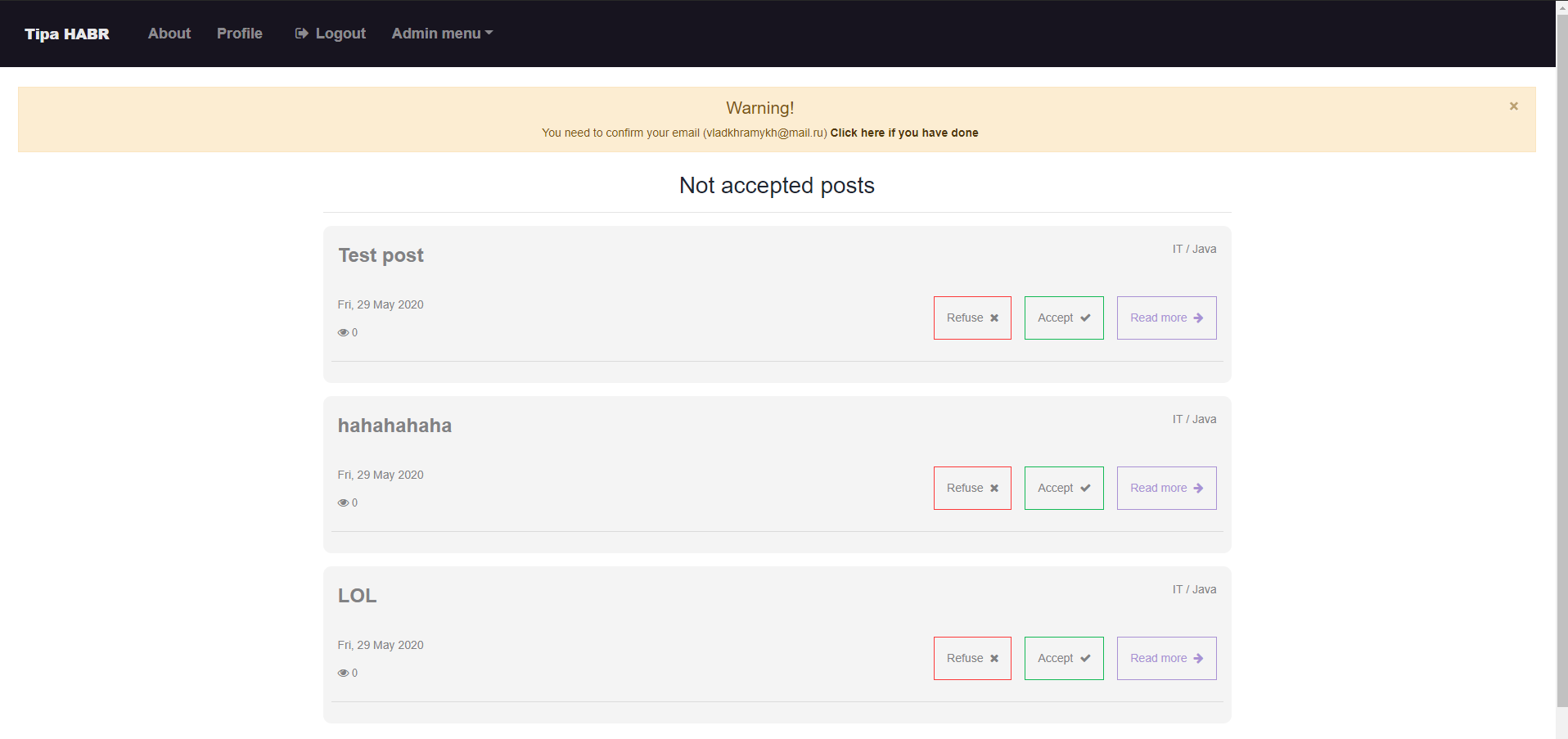


Рисунок 4.9 – Меню постов (Админ меню)

# **Тестирование**

Для обеспечения корректности работы программы, обрабатываются различные ошибки, возникающие в процессе работы. ПС работает с базой данных, поэтому допускать ввод некорректных данных недопустимо, т.к. это может привести в неработоспособности приложения.

В курсовом проекте организована валидация полей на случай неочевидного или неправильного поведения пользователя.

При регистрации обрабатываются вводимые данные пользователя. Так как поле «email» является уникальным, в базе данных не могут храниться пользователи с одинаковыми email, если в базе данных уже существует пользователь с таким же email, то клиент не будет зарегистрирован в системе (Рисунок 5.1.)

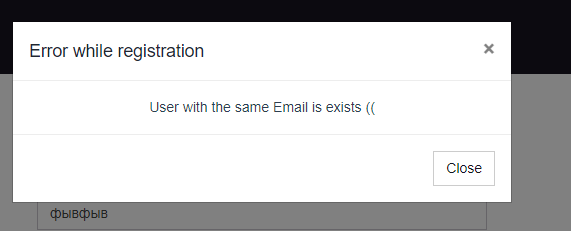


Рисунок 5.1 – Повторение email

Также возникают ошибки, если пользователь пытается ввести неверные данные в поле «Password confirm» (рисунок 5.2). Введенный пароль в поле «Password» должен полностью совпадать с введенным паролем в поле «Confirm password».

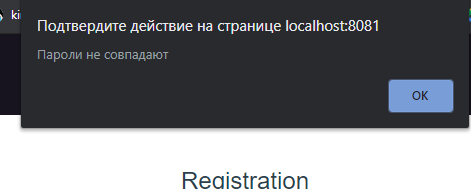


Рисунок 5.2 – Несовпадение паролей

При входе в систему также обрабатываются вводимые данные пользователя. Если пользователь неверно ввел данные в поля для входа, его уведомят об этом, пример уведомления об ошибке предоставлен на рисунке 5.3.

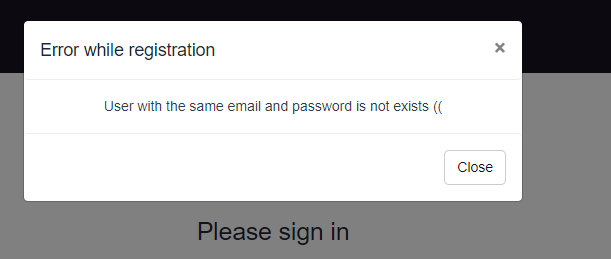
****

Рисунок 5.3 – Неверно введены данные

# **Заключение**

В данном курсовом проекте была разработана база данных и WEB-приложение «TipaHabr», которое позволяет пользователю просматривать посты на различную тематику, есть возможность регистрации, авторизации, и если вы авторизованы, то и добавления поста. Были выполнены следующие задачи:

* добавление, изменение данных в базе данных;
* организация регистрации и входа в систему со стороны пользователей;
* реализация архитектуры MVC;
* возможность просмотра, добавления постов;
* взаимодействие между клиентом и сервером происходит по протоколу https.

Данное программное средство использует технологии Node.js, фреймворк Express, ORM библиотека sequelize, WebSoket для реализации счётчика онлайн, для front-end части Bootstrap, JavaScript, Vue.js, CSS. MS SQL в качестве базы данных.

В соответствии с полученным результатом работы программы, можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# **Список используемых источников**

1. Что такое Node.js [Электронный ресурс]. — Режим доступа: <https://metanit.com/web/nodejs/1.1.php> Дата доступа: 12.04.2020.
2. Начало работы с Express [Электронный ресурс]. — Режим доступа: <https://metanit.com/web/nodejs/4.1.php> Дата доступа: 07.03.2020.
3. Что такое Sequelize [Электронный ресурс]. — Режим доступа: <https://metanit.com/web/nodejs/9.1.php> Дата доступа: 22.03.2020.
4. Асинхронный веб, или что такое веб-сокеты [Электронный ресурс]. — Режим доступа: <https://tproger.ru/translations/what-are-web-sockets/> Дата доступа: 17.03.2020.
5. Bootstrap [Электронный ресурс]. — Режим доступа: <https://getbootstrap.com/> Дата доступа: 27.03.2020.
6. JavaScript – [Электронный ресурс]. — Режим доступа:  <https://ru.wikipedia.org/wiki/JavaScript> Дата доступа: 24.03.2020.
7. Vue.js – The progressive JS Framework [Электронный ресурс]. — Режим доступа: <https://vuejs.org/> Дата доступа: 15.04.2020.
8. Vuex [Электронный ресурс]. — Режим доступа: <https://vuex.vuejs.org/ru/guide/> Дата доступа: 25.04.2020.
9. N-layer архитектура [Электронный ресурс]. — Режим доступа: <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier> Дата доступа: 07.03.2020.

# **Приложение А**

<template>  
 <div class="container col-sm-3">  
 <form class="form-signin" @submit.prevent="login">  
 <img class="mb-4" src="../assets/logo.png" alt="" width="72" height="72">  
 <h1 class="h3 mb-3 font-weight-normal">Please sign in</h1>  
 <div class="form-group">  
 <label for="exampleInputEmail1" class="sr-only">Email address</label>  
 <input v-model="email" type="email" class="form-control" id="exampleInputEmail1"  
 aria-describedby="emailHelp" placeholder="Enter email" required autofocus>  
 <small id="emailHelp" class="form-text text-muted">  
 We'll never share your email with anyone else.  
 </small>  
 </div>  
 <div class="form-group">  
 <label for="exampleInputPassword1">Password</label>  
 <input v-model="pwd" type="password" class="form-control" id="exampleInputPassword1"  
 placeholder="Password" required>  
 </div>  
 <button type="submit" class="btn btn-lg btn-primary btn-block">Login</button>  
 </form>  
 <div class="row">  
 <router-link to="/registration">If you don't have an account, click here</router-link>  
 </div>  
 <!-- <a @click.prevent="loginGoogle" href='http://localhost:3000/login/google'>google</a>-->  
 <div class="row">  
 <button onclick="location.href='http://localhost:3000/login/google'">Login with google</button>  
 </div>  
  
 <button id="wrong" class="btn" data-toggle="modal" data-target="#modal" hidden>Modal</button>  
  
 <div class="modal fade" id="modal" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel"  
 aria-hidden="true">  
 <div class="modal-dialog">  
 <div class="modal-content">  
 <div class="modal-header">  
 <h5 class="modal-title" id="exampleModalLabel">Error while registration</h5>  
 <button type="button" class="close" data-dismiss="modal" aria-label="Close">  
 <span aria-hidden="true">&times;</span>  
 </button>  
 </div>  
 <div class="modal-body">  
 User with the same email and password is not exists ((  
 </div>  
 <div class="modal-footer">  
 <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>  
 </div>  
 </div>  
 </div>  
 </div>  
  
 <!-- <p class="mt-5 mb-3 text-muted">&copy; 2017-2018</p>-->  
 </div>  
</template>  
  
<script>  
 export default {  
 name: "LoginForm",  
 date() {  
 return {  
 email: '',  
 pwd: ''  
 };  
 },  
 methods: {  
 login: function () {  
 let email = this.email;  
 let password = this.pwd;  
 this.$store  
 .dispatch("login", {email, password})  
 .then(() => {  
 this.$router.push("/");  
 window.location.reload();  
 })  
 .catch(err => {  
 document.getElementById("wrong").click();  
 });  
 },  
 loginGoogle: function () {  
 this.$store  
 .dispatch("loginGoogle")  
 .then(() => {  
 this.$router.push("/");  
 window.location.reload();  
 })  
 .catch(err => console.log(err));  
 }  
 }  
  
 }  
</script>  
  
<style scoped>  
  
 \* {  
 border-radius: 5px;  
 }  
</style>

# **Приложение Б**

exports.registration = *function* (req, res) {  
 *let* user = req.body;  
 *if*(user) {  
 userRepo.findOneByEmail(user.email)  
 .then((foundedUser) => {  
 *if* (!foundedUser) {  
 user.password = bcrypt.hashSync(user.password, 8);  
 user.activationCode = uuid.v4();  
 user.role = 'user';  
 mailer(user);  
 userRepo.add(user)  
 .then((created) => {  
 *let* token = jwt.sign({id: created.id}, config.appSession.secret, {  
 expiresIn: 86400  
 });  
 res.status(201);  
 res.json({auth: *true*, token: token, user: created});  
 })  
 } *else* {  
 res.status(404).send('User with the same email exists');  
 }  
 });  
 } *else* {  
 res.status(404).send('Bad request');  
 }  
};

# **Приложение В**

exports.login = *function* (req, res) {  
 *let* user = req.body;  
  
 userRepo.findOneByEmail(user.email)  
 .then((foundedUser) => {  
 *if* (foundedUser) {  
 *let* passwordIsValid = bcrypt.compareSync(user.password, foundedUser.password);  
 *if* (!passwordIsValid) {  
 *return* res.status(401).send({auth: *false*, token: *null*});  
 }  
 *let* token = jwt.sign({id: foundedUser.id}, config.appSession.secret, {  
 expiresIn: 86400  
 });  
 res.status(200);  
 res.json({auth: *true*, token: token, user: foundedUser});  
 } *else* {  
 res.status(404);  
 res.json('No user found');  
 }  
 });  
};