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

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Дисциплина «Разработка интернет-приложений»

Расчетно-пояснительная записка

Тема: «Шифрование кодом для коррекции ошибок»

Студент: Попов С. Д.

Группа ИУ5-53Б

Преподаватель: Канев А.И.

2023г.

# **ВВЕДЕНИЕ**

В современном мире информация имеет наибольшую ценность. Нельзя недооценить важность передачи данных в неискаженном виде для обеспечения точности и достоверности информации. При передаче данных в искаженном виде возникает риск их неправильной интерпретации, что может привести к ошибкам в принятии решений и негативным последствиям. Поэтому поддержание целостности и точности данных в процессе передачи имеет ключевое значение для обеспечения эффективности и надежности бизнес-процессов, коммуникаций и принятия решений. Коды коррекции ошибок являются одним из способов проверки достоверности передаваемых данных и исправления ошибок.

Идеи и принципы их использования существуют уже много лет. Например, код Паритет был впервые предложен в 19 веке. Код Хэмминга был придуман Ричардом Хэммингом в 1950 году. Он работал в Bell Labs в конце 1940-х годов над компьютером Bell Model V, электромеханической машиной на основе реле, которая часто давала сбои. За несколько следующих лет он работал над проблемой исправления ошибок и опубликовал то, что сейчас известно как код Хэмминга. Код CRC (Cyclic Redundancy Check) был впервые предложен Уилсоном Петтерсоном в 1961 году. Однако технологии и методы, используемые для создания и использования кодов коррекции ошибок, постоянно развиваются и улучшаются.

Кодированию информации с избыточностью подвергается хранимая информации и практически вся передаваемая информация по сети на разных уровнях модели OSI: на втором, третьем, четвертом и седьмом. Например, в области телекоммуникаций объемы передачи закодированных данных могут достигать нескольких гигабайт и даже терабайт. В Web разработке и связанной с ней инфраструктуре в связи с актуальностью объемы больше на несколько порядков.

Целью работы является реализация системы для автоматизации проведения операций кодирования кодами для коррекции ошибок, включающую в себя веб-сервис, веб-приложение, мобильное приложение и выделенный сервис.

Система предназначена для шифровальщиков и модераторов заявок на кодирование кодом коррекции ошибок или исправления ошибок. Предусмотрен доступ к списку двоичных данных, представленных для кодирования. Шифровальщики могут создавать заявки, выбирая нужные двоичные данные и указывая подходящий тип кодов коррекции ошибок. Система предоставляет автоматизированный способ создания, учета и ведения заявок кодирования, обеспечивая эффективное управление этими заявками. Модераторы имеют возможность подтверждать или отклонять заявки на кодирование кодом коррекции ошибок или исправления ошибок, а также редактировать существующие двоичные данные для кодирование.

Нефункциональные требования к разрабатываемой системе:

1. Должна поддерживаться кроссплатформенность.
2. Интерфейс системы и текст ошибок должны быть русифицируемы.

В ходе работы необходимо выполнить следующие задачи:

* 1. Разработать дизайн приложения.
  2. Создать базу данных в PostgreSQL.
  3. Создать веб-сервис на языке GoLang 1.20.
  4. Реализовать интерфейс гостя на технологии React.
  5. Развернуть веб-приложение React на Github Pages.
  6. Добавить авторизацию и аутентификацию в веб-сервис.
  7. Реализовать интерфейс пользователя в React.
  8. Реализовать интерфейс модератора React.
  9. Создать мобильное приложение на React-Native.
  10. Создать асинхронный сервис для расчета кода коррекции ошибок на Python.
  11. Подготовить набор документации, включающий РПЗ, ТЗ и набор диаграмм.

# **БИЗНЕС-ПРОЦЕСС**

Коды коррекции ошибок (Или ECC) - это специальные методы, используемые для обнаружения и исправления ошибок, возникающих при передаче данных. Они играют важную роль в различных аспектах информационных технологий, включая хранение данных, передачу информации по сетям связи и запись на носителях.

Основная цель кодов коррекции ошибок заключается в том, чтобы обеспечить надёжность передачи и хранения данных. При передаче данных по сети или записи их на носитель, могут возникать ошибки из-за различных факторов, таких как электромагнитные помехи, физические дефекты носителя, или человеческие ошибки. Коды ECC позволяют обнаруживать и исправлять эти ошибки, что существенно повышает надёжность систем информационного хранения и обмена.

Применение кодов коррекции ошибок распространено в различных сферах, включая жесткие диски, оперативную память, оптические диски, сетевые коммуникации и даже космические аппараты. Они играют важную роль в обеспечении целостности данных и минимизации возможных проблем, связанных с ошибками передачи.

В моем приложении представлены два вида корректирующих кодов: код Хэмминга и CRC.

Код Хэмминга - специальный вид блочных кодов с добавлением избыточности. Они имеют широкое применение в компьютерной памяти типа ECC, для исправления ошибок, происходящих из за космических лучей и в технологии RAID 2. В этой конфигурации каждый блок данных разбивается на отдельные биты и записывается на отдельные диски с использованием кодов Хэмминга для обнаружения и исправления ошибок. По сути, каждый диск в массиве хранит различные биты от всех остальных дисков, а коды Хэмминга позволяют восстанавливать потерянные данные.

Циклический избыточный код (CRC) - это метод проверки целостности данных, который используется для обнаружения ошибок при передаче данных. CRC генерируется путем деления данных на заданный полином и добавления полученного остатка к исходным данным. При получении данных приемное устройство также вычисляет CRC и сравнивает его с принятыми данными. Если CRC не совпадает, это указывает на возможные ошибки в переданных данных. CRC широко применяется в цифровых коммуникационных системах, сетях передачи данных, хранении данных и других областях, где важна целостность и надежность передаваемой информации. Он используется в протоколах передачи данных, таких как Ethernet, Wi-Fi, Bluetooth, а также в файловых форматах, сжатии данных, и других областях, где требуется обеспечить точность и целостность передаваемых данных.

Для применения корректирующих кодов для больших сообщений на практике требуется произвести определенный набор вычислений. Моя система занимается автоматизацией этих вычислений.

Каждая заявка на шифрование/коррекцию ошибок вначале создается как черновик, в который можно добавлять и из которого можно удалять данные для выше названных операций. В заявке можно выбрать каким кодом для коррекции ошибок будет происходить кодирование/декодирование: код Хэмминга или же CRC(циклический код).

Когда пользователь удостоверится, что выбраны все нужные данные, он формирует заявку и оно отправляется на модерирование. После этого его уже нельзя редактировать. Можно также посмотреть историю своих прошлых заявок. Для того, чтобы минимизировать вероятность появления неопределенных значений от сервиса, модераторы отклоняют и подтверждают заявки таким образом, чтобы удостовериться в правильности работы алгоритма. Само шифрование производит сторонний асинхронный сервис.

В описании данных, пригодных для обработки корректирующими кодами могут присутствовать какие-то ошибки, и в этом случае модератор может либо изменить эти данные, либо полностью удалить их из списка. Он может отредактировать какие то двоичные данные, обновить относящееся к ним фото, или исправить ошибки. Также модератор может создать новые данные. Функции пользователей с разными ролями описаны на диаграммах прецедентов (рис. 1).

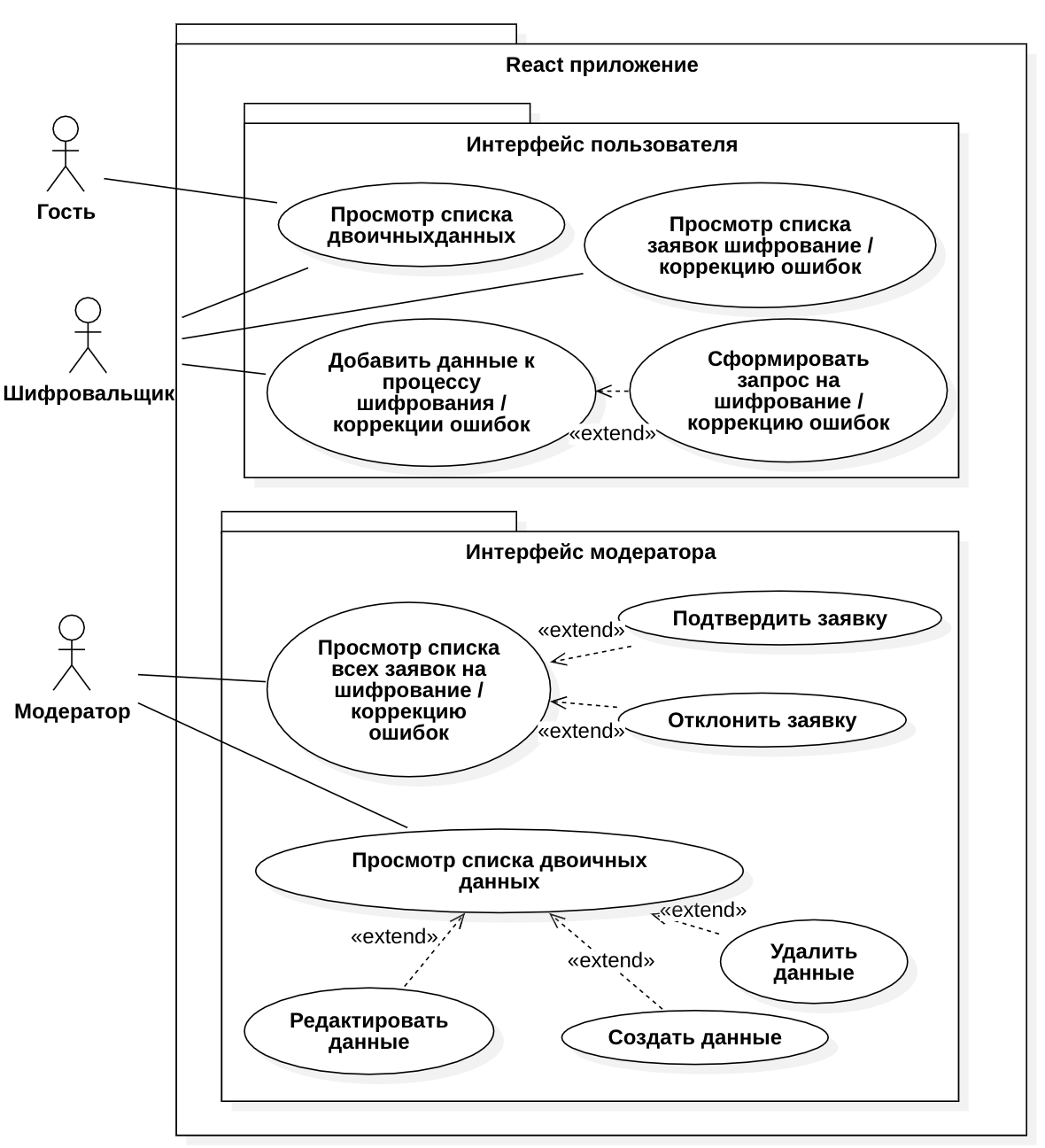


Рисунок 1 - Диаграмма прецедентов

Гостям доступен просмотр данных для кодирования. Зарегистрированные гости – шифровальщики. Они могут добавлять двоичные данные в заявку, просматривать список своих заявок и отправлять заявку на вычисление необходимых операций кодирования / декодирования. Заявки обрабатываются модераторами. В результате обработки заявки, ее либо подтверждают, либо отклоняют. Также модератору доступны уникальные функции для работы с двоичными данными, а именно: создание и удаление двоичных данных, а также просмотр всех данных для кодирования в табличном виде. Процесс формирования заявки отражен на диаграмме деятельности (рис. 2).

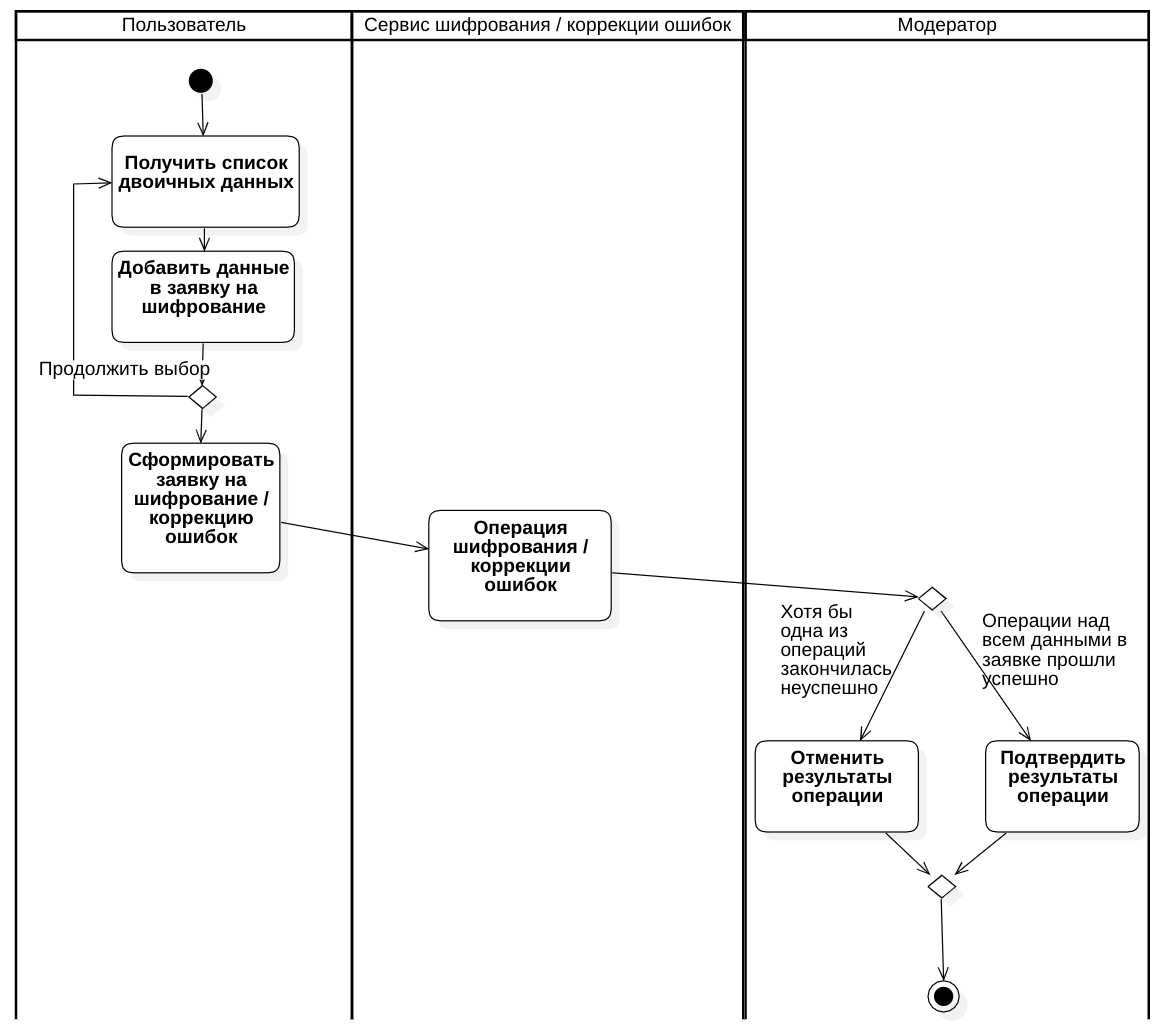


Рисунок 2 - Диаграмма деятельности

Шифровальщик выбирает нужные двоичные данные, затем формирует на основе выбранных данных заявку на кодирование либо исправление ошибок в зависимости от исходных данных. Эту заявку обрабатывает асинхронный сервис, а затем и модератор. В случае одобрения шифровальщику показываются результаты кодирования или исправленные исходные данные. Возможные состояния заявки на кодирование/исправление ошибок отражены на диаграмме состояний (рис. 3).

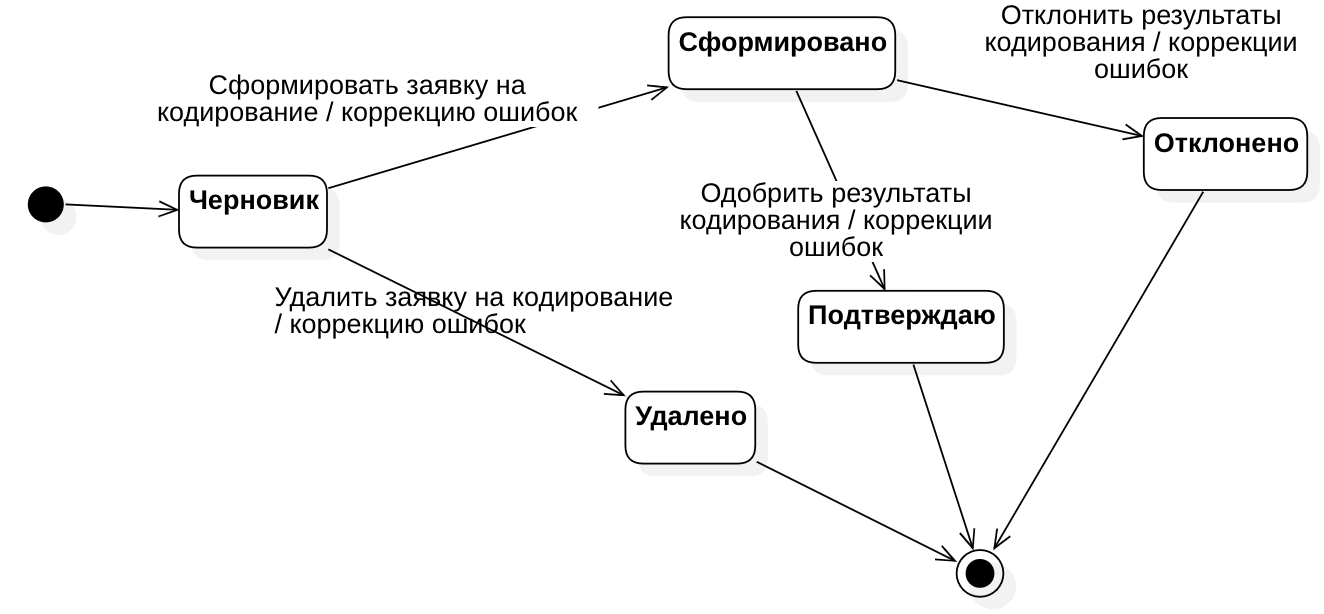


Рисунок 3 - Диаграмма состояний заявки

При выборе первых исходных данных формируется черновик. Последующие выбранные данные добавляются в этот черновик. Шифровальщик затем формирует заявку, удаляет её или выходит из приложения. Сформированную заявку обрабатывает модератор. Он может одобрить или отклонить её.

# **АРХИТЕКТУРА**

Архитектура системы отображена на диаграмме развертывания (рис. 4). Бэкенд и база данных разворачивается на самом устройстве, а s3 хранилище изображений, Redis в докер контейнерах [6]. Такое решение было принято в связи с тем, что технология докер контейнеров позволяет быстро и удобно разворачивать целые системы на любых устройствах, поддерживающих докер. Еще одним неоспоримым преимуществом докер контейнеров является то, что их контейнеры представляют собой виртуальные машины, а значит их работа не зависит от платформы устройствах. Сервис вычислений кода коррекции ошибок [3] разворачиваются отдельно и необязательно на том же компьютере, что и основной сервис.

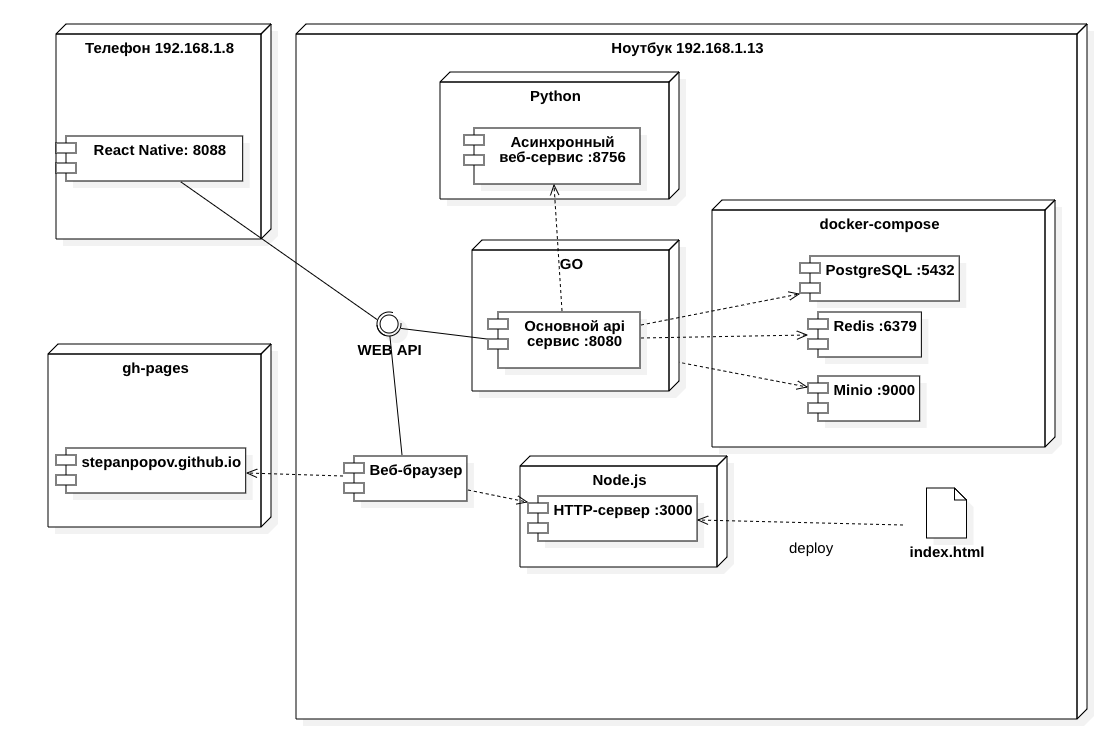


Рис. 4 - Диаграмма развертывания

Мобильное [10] и браузерное приложения [7, 8, 9] обращаются к веб-сервису на базе языке Golang 1.20 [3, 4, 5] через REST Web-API. Использование Golang обосновано его проверенной эффективностью, признанием в индустрии и современностью языка.

Данные хранятся в СУБД PostgreSQL [11], их структура отражена на ER диаграмме (рис. 5). СУБД PostgreSQL является одним из стандартов индустрии, поэтому было решено использовать её. Структура данных довольна проста. Помимо базовых полей, заявка на кодирование кодом коррекции ошибок также имеет поля ResultCounter для отображения количество успешных операций вычислений асинхронного сервиса, а также EncodingType отражающее тип кода коррекции ошибок (Хэмминг / CRC). Для хранения в одной заявке нескольких данных для шифрования используется промежуточная таблица «EncryptDecryptToData», реализующая связь М-М.

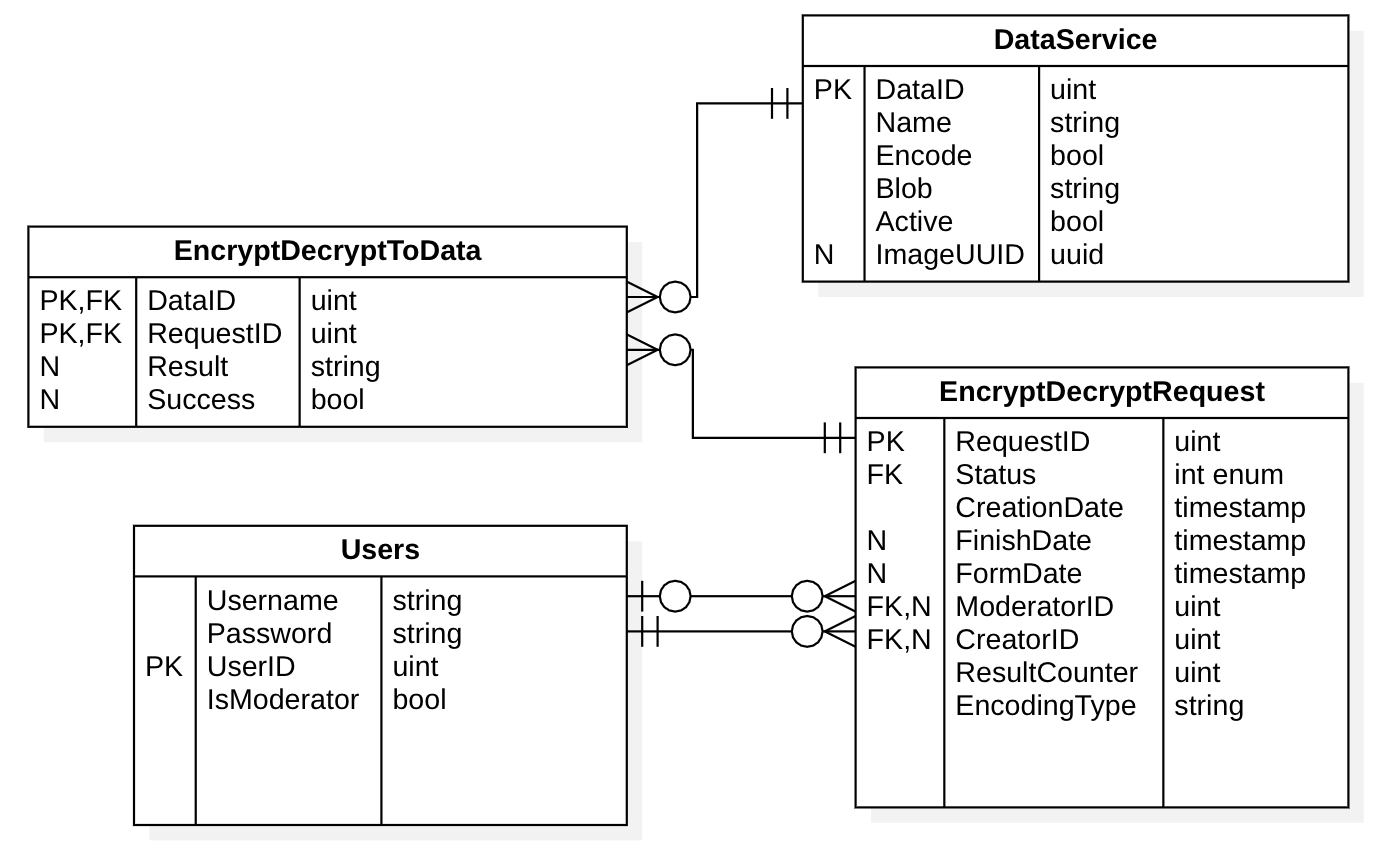


Рисунок 5 - ER диаграмма

Устройство бекенда приложения изображено на диаграмме классов бекенда (рис 6.). Модели имеют связи с таблицами в базе данных. Также некоторые модели имеют связи с внешними сервисами. В частности, исходные данные имеют связь с объектным хранилищем S3, т.к. в исходных хранится ссылка на их изображение, хранимое на сервере статических файлов.

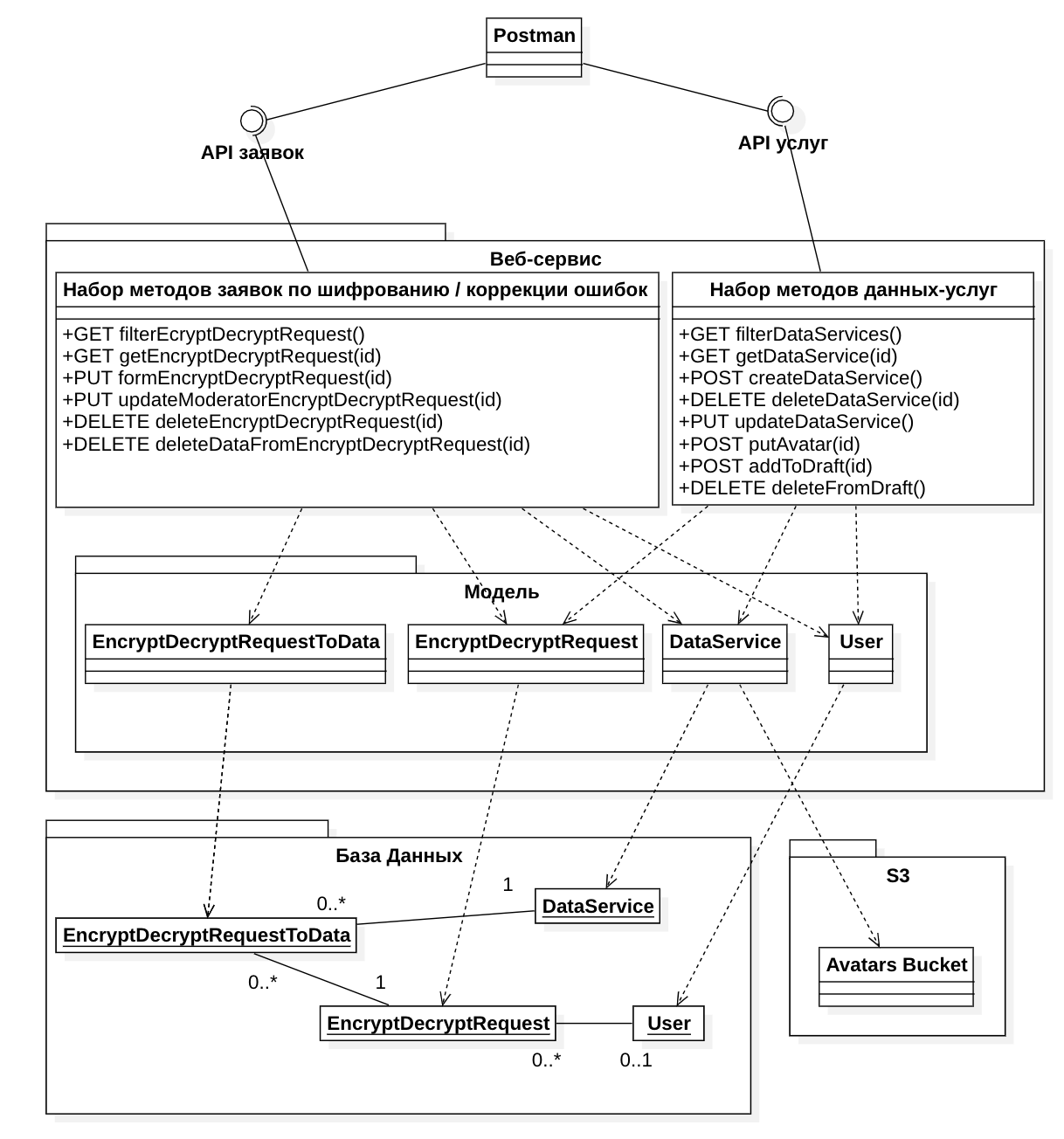


Рисунок 6 *-* Диаграмма классов бекенда

Связь фронтенда и бекенда отражена на диаграмме классов фронтенда (рис. 7). Ключевые страницы имеют связь с API аутентификации, т.к. доступ к ним осуществляется только для авторизированных пользователей с определенными правами (ролями).

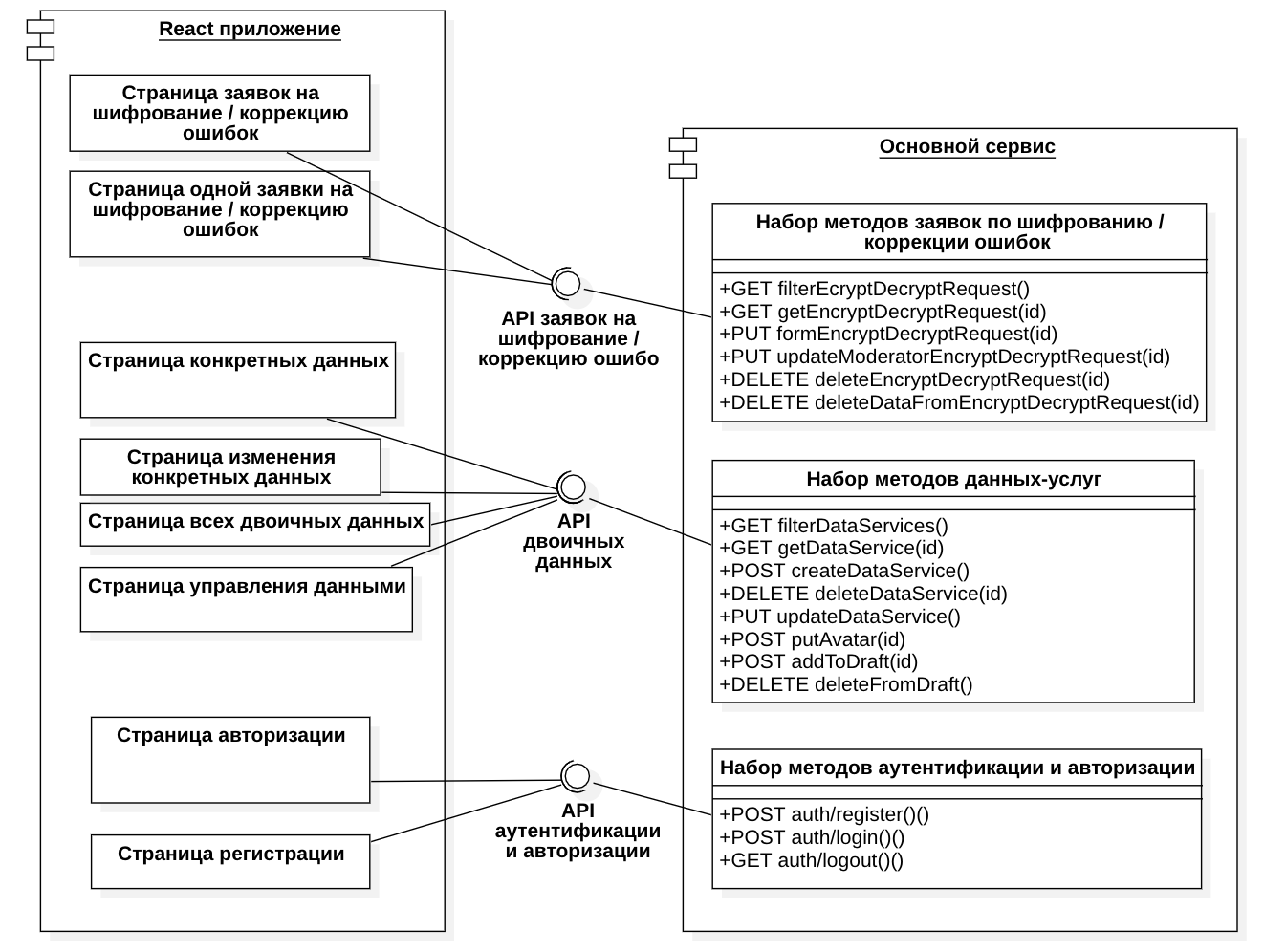


Рисунок 7 *-* Диаграмма классов фронтенда

# **АЛГОРИТМЫ**

Алгоритм работы системы отображен на диаграмме последовательности (рис. 8). В основе системы лежит веб-сервис, реализующий внутри себя всю бизнес-логику. Он предоставляет доступ к методам из следующих доменов: данные-услуги, заявки на шифрование/расшифрование и аутентификация. Методы следуют правилам REST API.

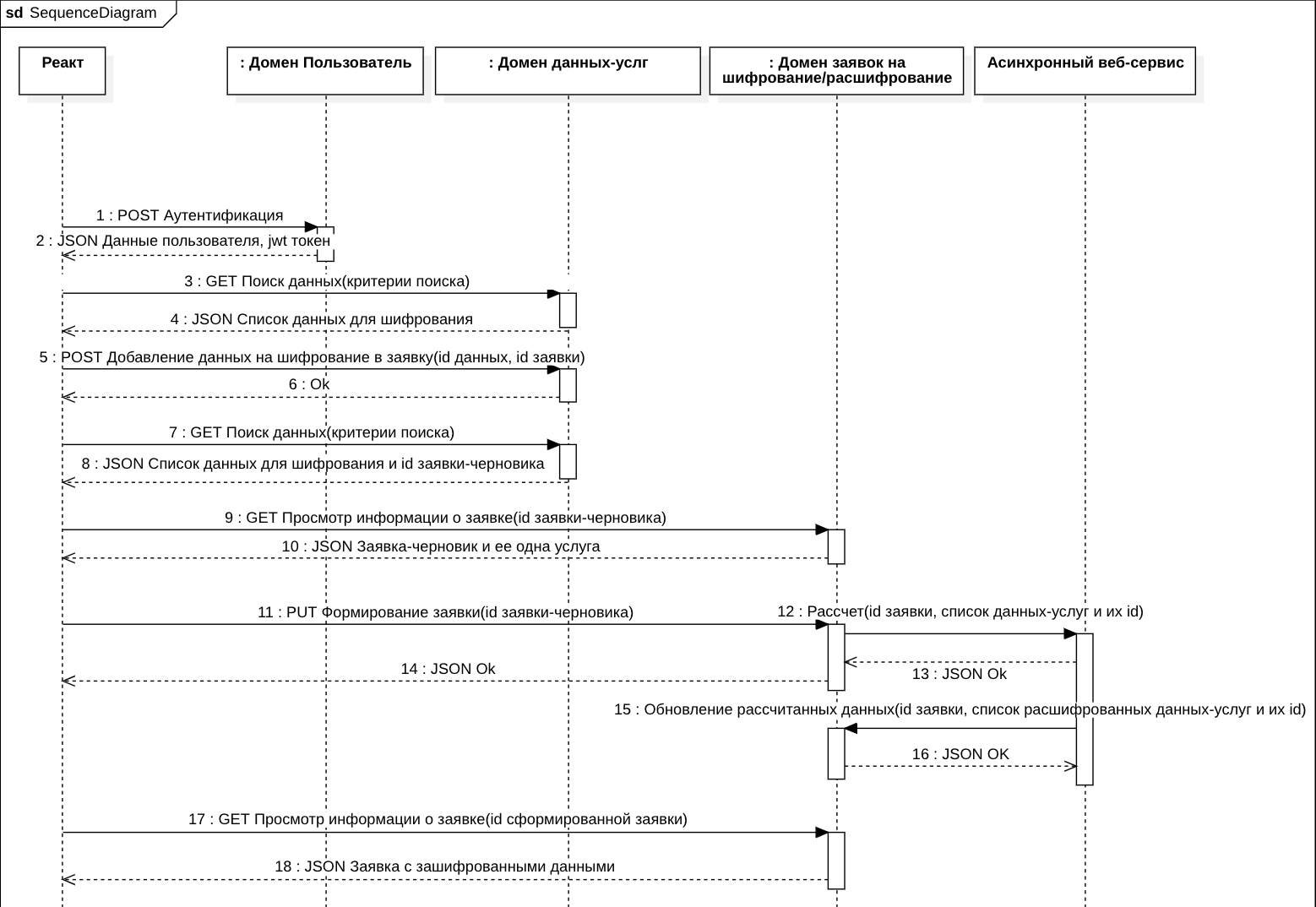


Рисунок 8 - Диаграмма последовательности

В начале бизнес-процесса происходит аутентификация пользователя. Для этого он отправляет через графический интерфейс запрос, передавая в нем логин и пароль. Если аккаунт с такими данными существует, то пользователь получает JWT в ответном запросе. Если же такого аккаунта не существует, или пароль введен неверно, пользователь получит ошибку. В таком случае ему надо либо пройти регистрацию, либо ввести пароль верно. Затем графический интерфейс пользователя запрашивает у веб-сервиса список исходных данных, которые возвращаются в JSON формате. Шифровальщик выбирает нужные исходные данные, которые хочет добавить в заявку, и, нажимая на кнопку «Добавить в корзину» в графическом интерфейсе, отправляет запрос на добавление двоичных данных в свою черновую заявку. Этот процесс может продолжаться несколько раз.

Когда шифровальщик определится с выбором, он нажимает на кнопку «сформировать» в графическом интерфейсе. После этого приложение запрашивает id черновой заявки шифровальщика и затем отправляет запрос на формирование этой заявки. В этот момент основной веб-сервис выполняет асинхронный запрос к отдельному сервису, чтобы тот произвел вычисления необходимых операций кодирования либо исправления ошибок. Через некоторое количество времени шифровальщик может увидеть, что статус его заявки изменился.

Процесс рассмотрения заявок происходит также через графический интерфейс. Модераторы могут просматривать списки всех заявок и, нажимая на соответствующие кнопки, отправлять запросы на подтверждение или отклонение заявок в основной веб-сервис. В эти запросы также можно включить фильтры по имени создателя заявки, одному из статусов заявок и диапазону дат, в которых эти заявки были сформированы. Также модераторы могут управлять двоичными данными через графический интерфейс. Им доступны такие функции, как создание и редактирование исходных данных, просмотр списка двоичных данных в виде таблицы и удаление их. Для каждой из этих функций присутствует свой метод, отправляемый на основной веб-сервис.

# **ОПИСАНИЕ ИНТЕРФЕЙСА**

Главное меню приложения включает пункты, которые доступны в зависимости от роли пользователя (рис. 9, 10).

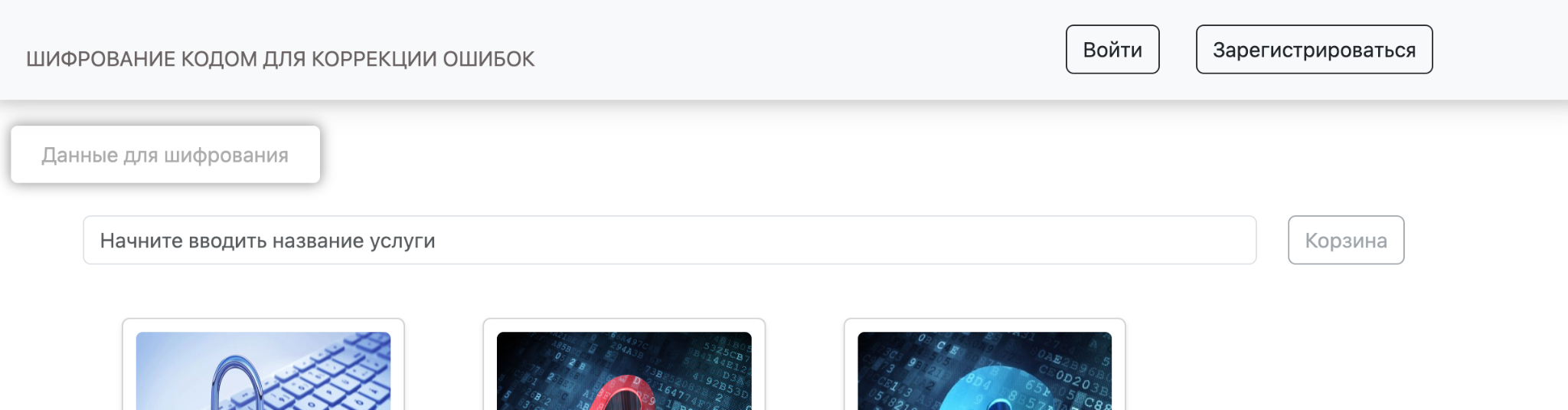


Рисунок 9 - Главное меню (неавторизированный пользователь)

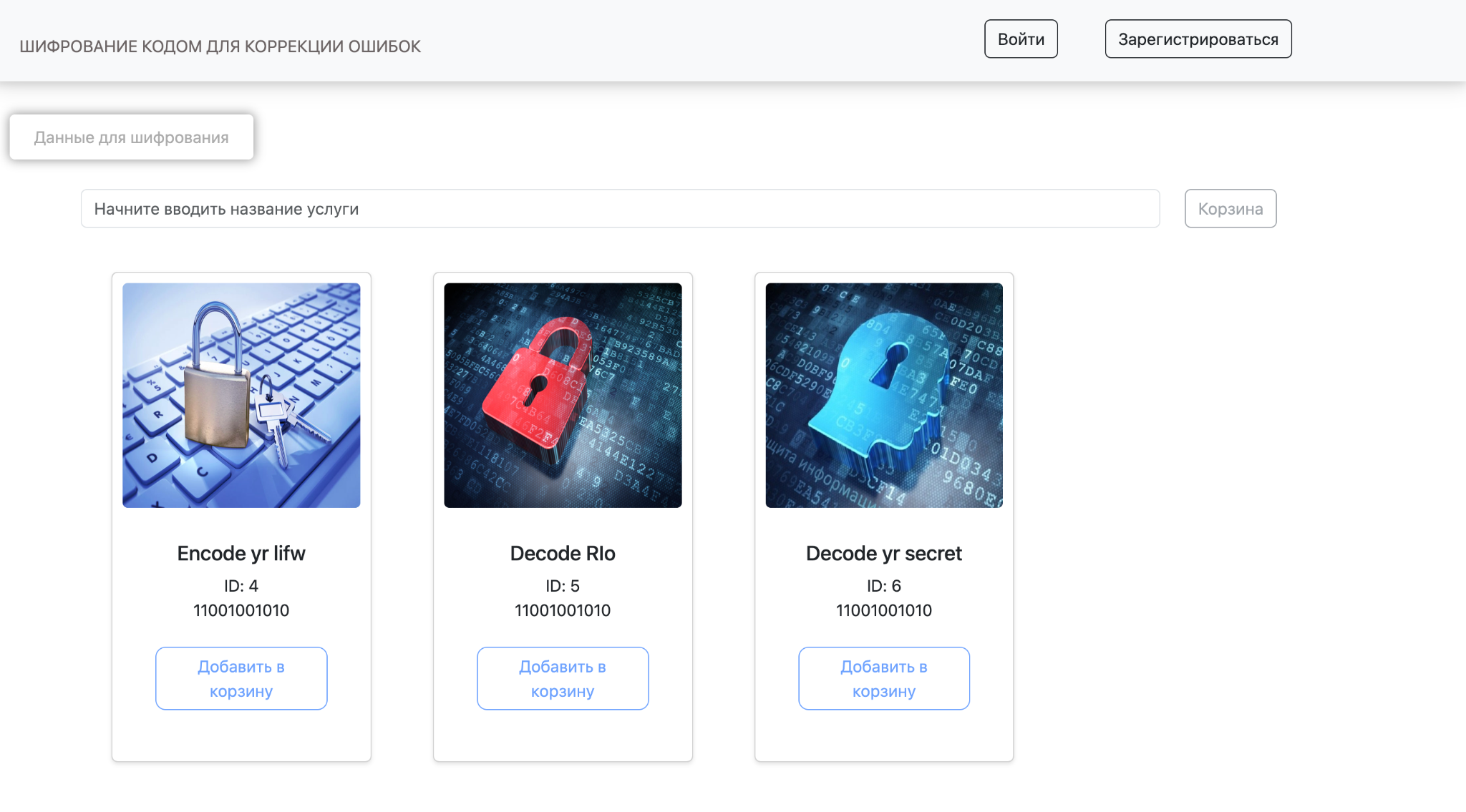


Рисунок 10 - Главное меню (модератор)

Первоначальная страница для всех пользователей и гостей. В зависимости от типа пользователя её содержимое меняется. Гости, например, могут посмотреть список двоичных данных для кодирования и подробную информацию о них, а также нажать на кнопку «Войти», либо «Регистрация». На страницу со списком исходных данных можно перейти, нажав на «Шифрование кодом для коррекции ошибок» в навбаре. А у шифровальщиков уже есть кнопки «Управление шифрованием», «Корзина» и «Выход».

На странице с формой авторизации (рис. 11) отображается форма, через которую гость входит в свой аккаунт. При успешном вводе логина и пароля гость получает JWT, который сохраняется в localstorage и используется при отправлении запросов.

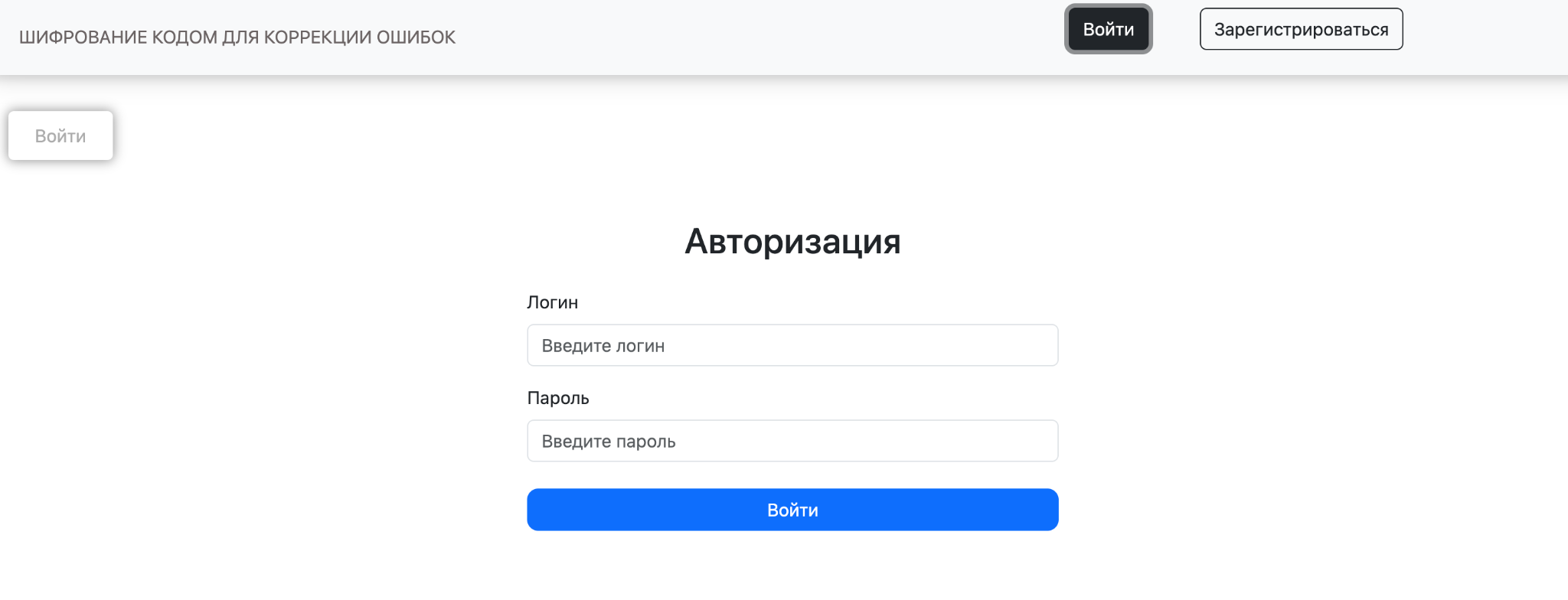


Рисунок 11 - Страница авторизации

На странице с формой регистрации (рис. 12) гости могут завести аккаунт. Для этого нужно указать логин, и пароль. Если введенный логин уже занят система попросит пользователя сменить его.

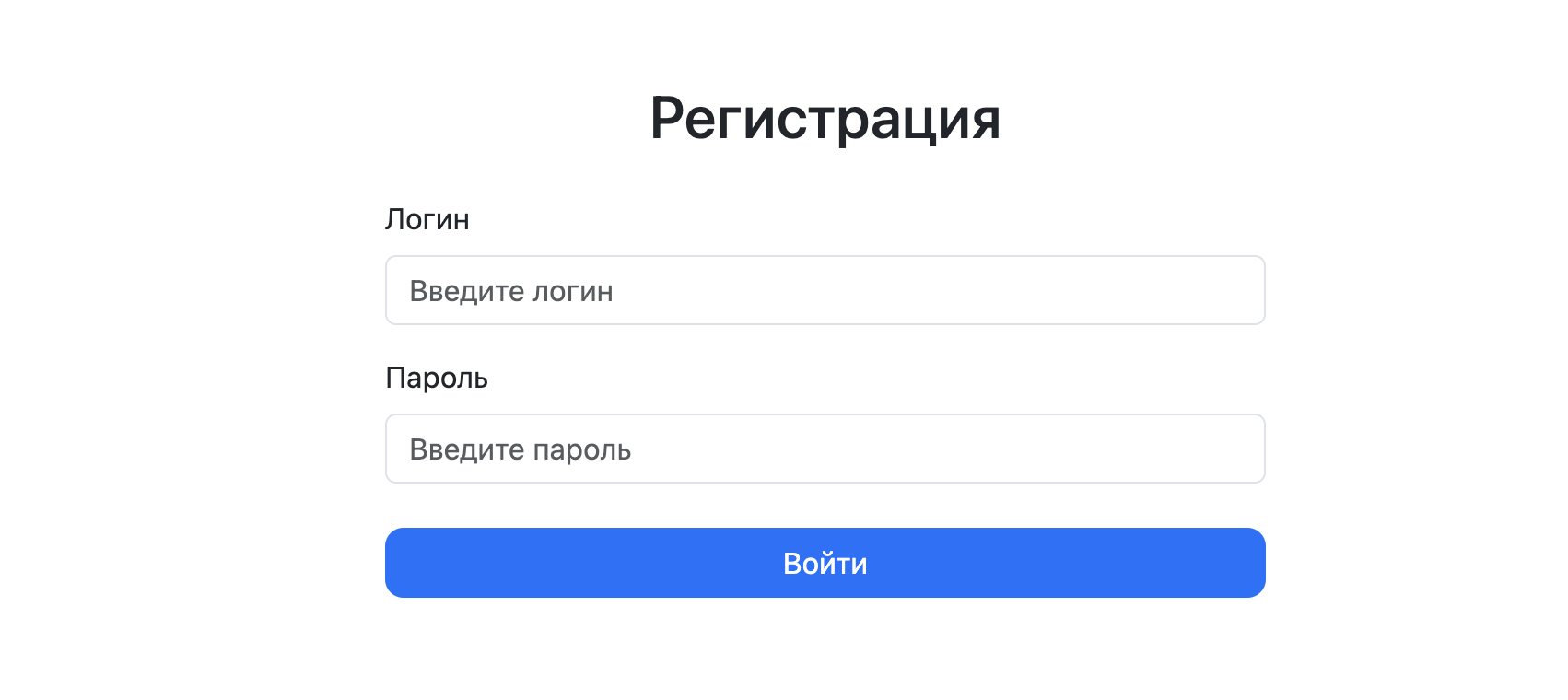


Рисунок 12 - Страница регистрации

На странице со списком исходных данных (рис. 13) отображается список исходных данных в виде карточек. При нажатии на карточку, осуществляется переход на страницу с подробной информацией, и кнопка «добавить в корзину» для добавления исходных данных в черновик. Сверху находится фильтр, где можно ввести название исходных данных для отображения нужных пользователю.

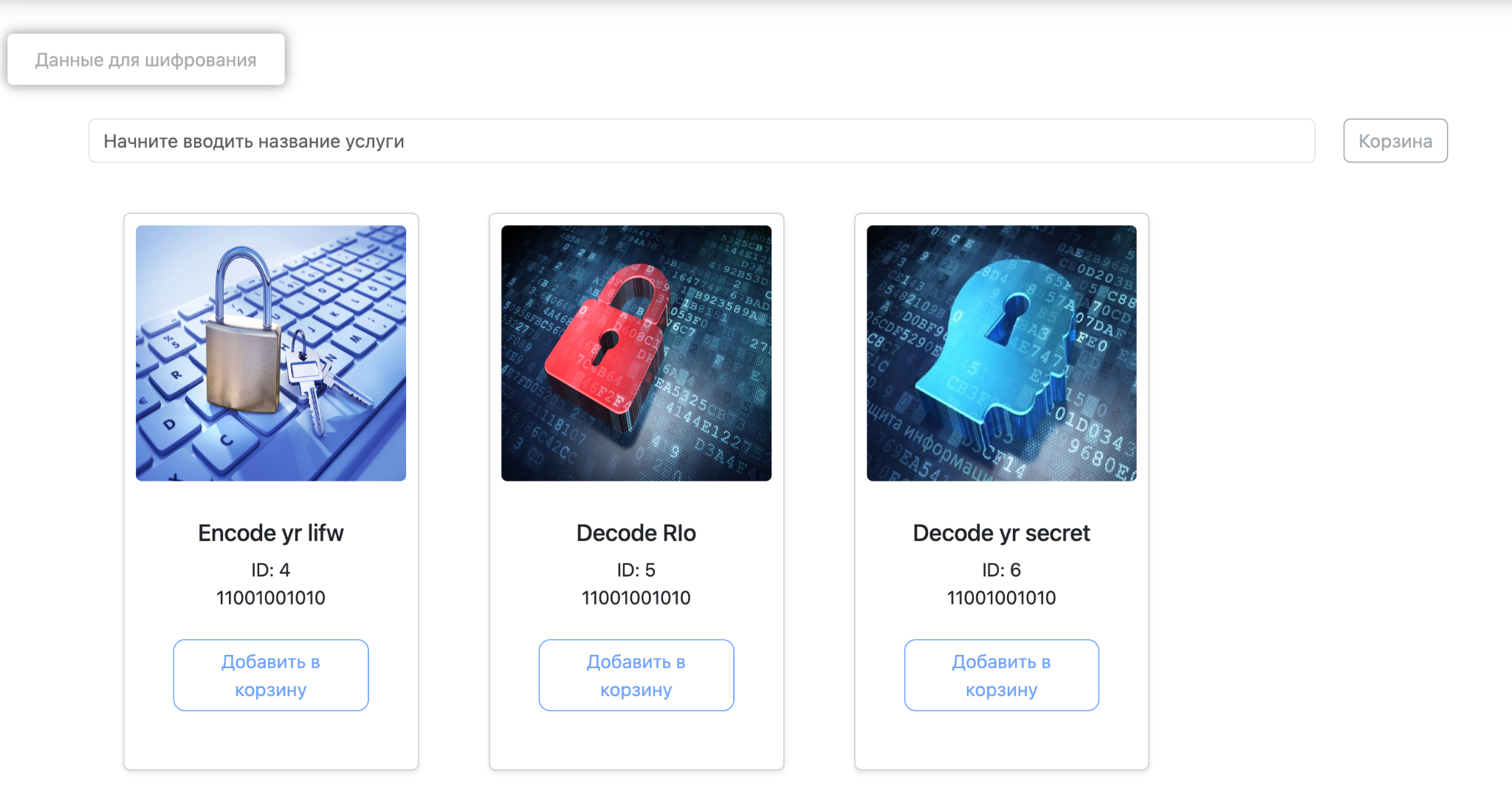


Рисунок 13 - Страница исходных данных

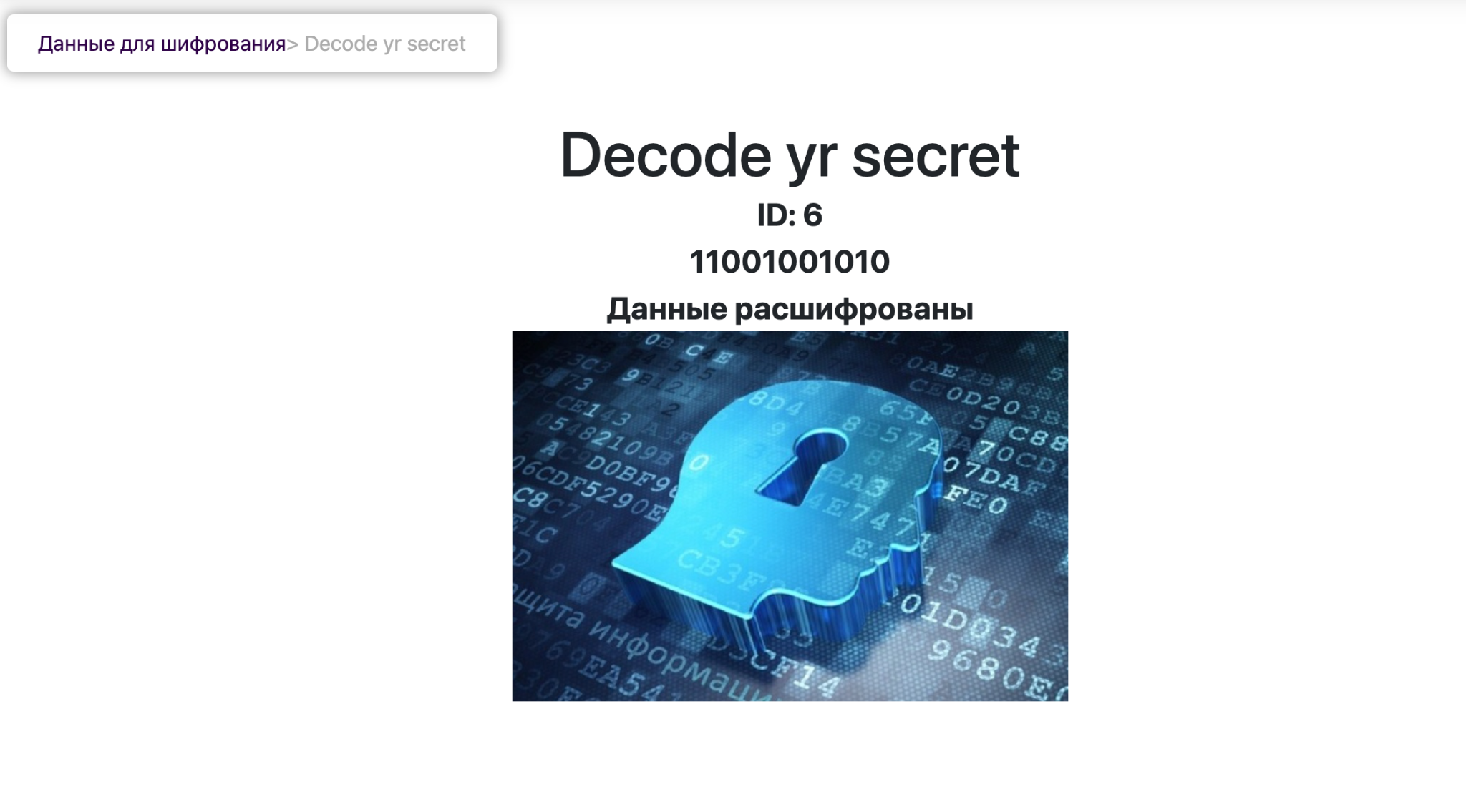
На странице с подробным описанием исходных данных (рис. 14) отображается подробная информация о данных.

Рисунок 14 - Страница с подробным описанием бинарных данных

На странице со списком заявок на кодирование либо исправление ошибок на (рис. 15, 16) отображается список заявок. В зависимости от типа пользователя этот список будет функционально отличаться. Так, для шифровальщиков отображается список созданных ими заявок: статус, дата создания, дата формирования, дата завершения, алгоритм кодирования, подсчитанные результаты и кнопка «Подробнее» для открытия заявки. Также можно отфильтровать заявки по статусу, либо по дате формирования, выбрав либо одну дату, либо промежуток.

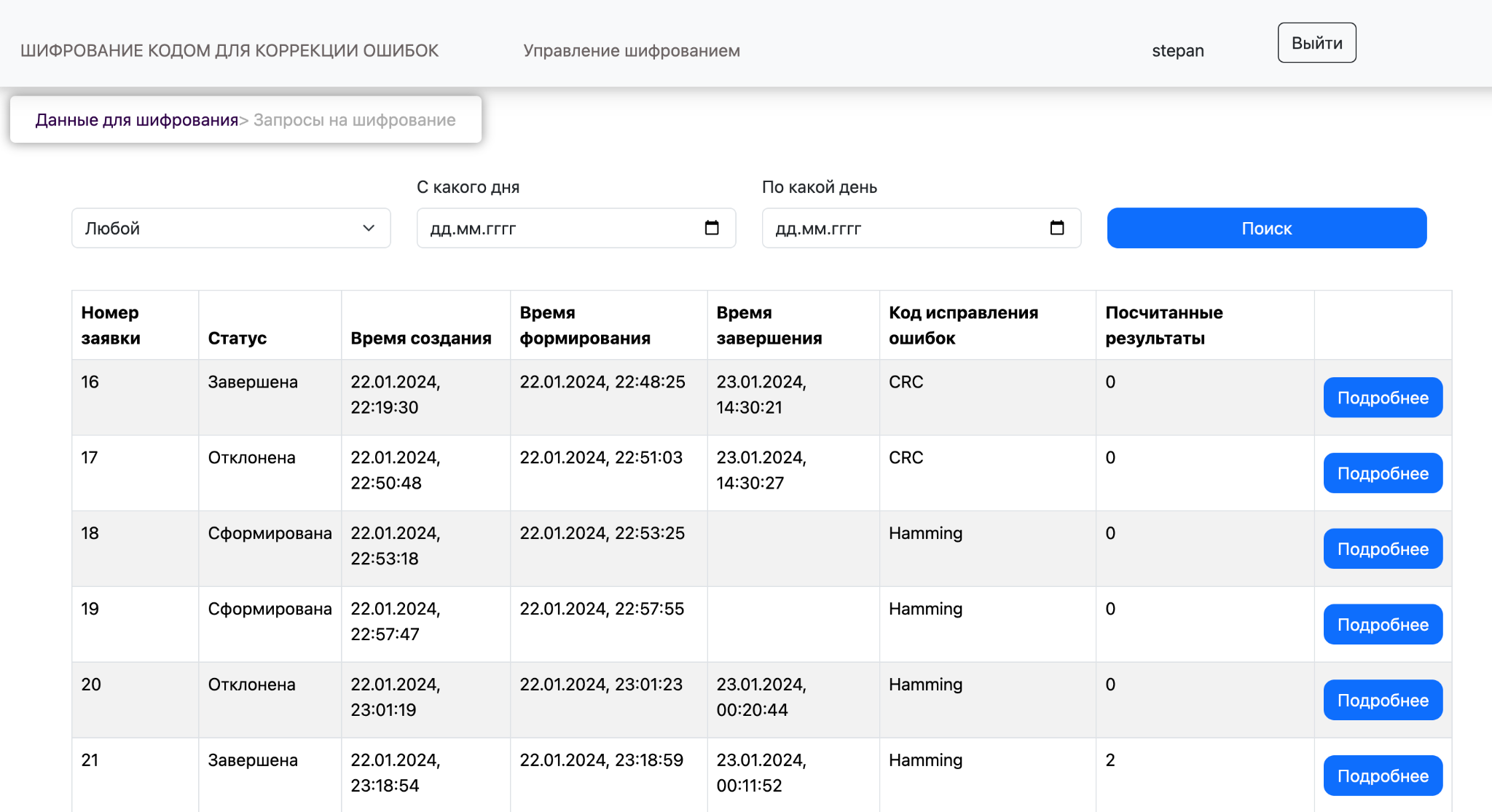


Рисунок 15 - Страница со списком заявок (пользователь)

Для модератора функция этой страницы шире. Для них отображается список всех заявок всех пользователей с более подробной информацией: создатель, статус, дата создания, дата формирования, дата завершения, алгоритм кодирования, подсчитанные результаты, кнопка «Подробнее» для открытия заявки, и две кнопки «Подтвердить» и «Отменить» для подтверждения или отмены заявки. Также присутствует фильтр по создателю.

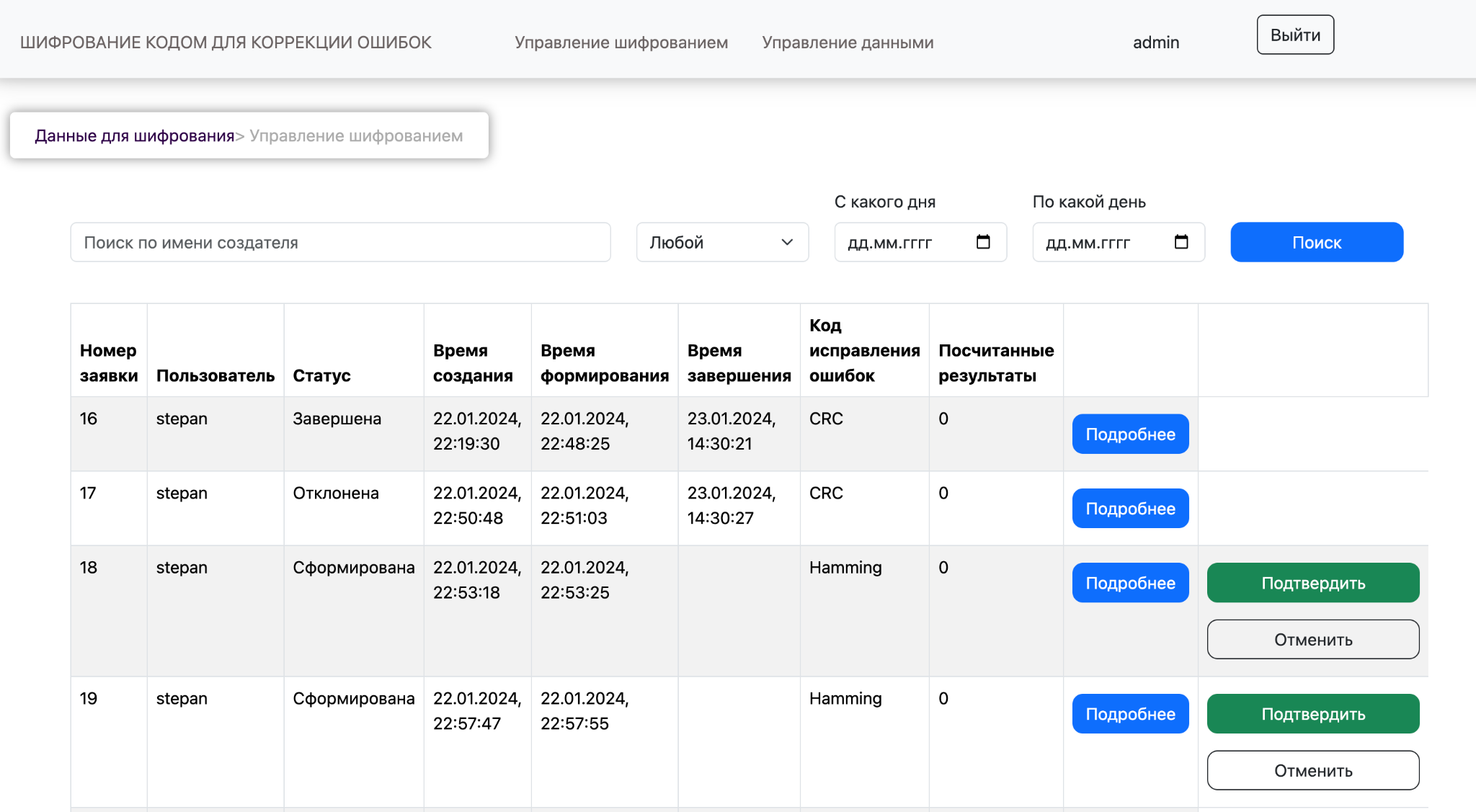


Рисунок 16 - Страница со списком заявок (модератор)

На странице с подробным описанием заявки на кодирование либо исправления ошибок (рис. 17) отображается подробная информация. Список выбранных исходных данных в виде карточек, а также все поля заявки.

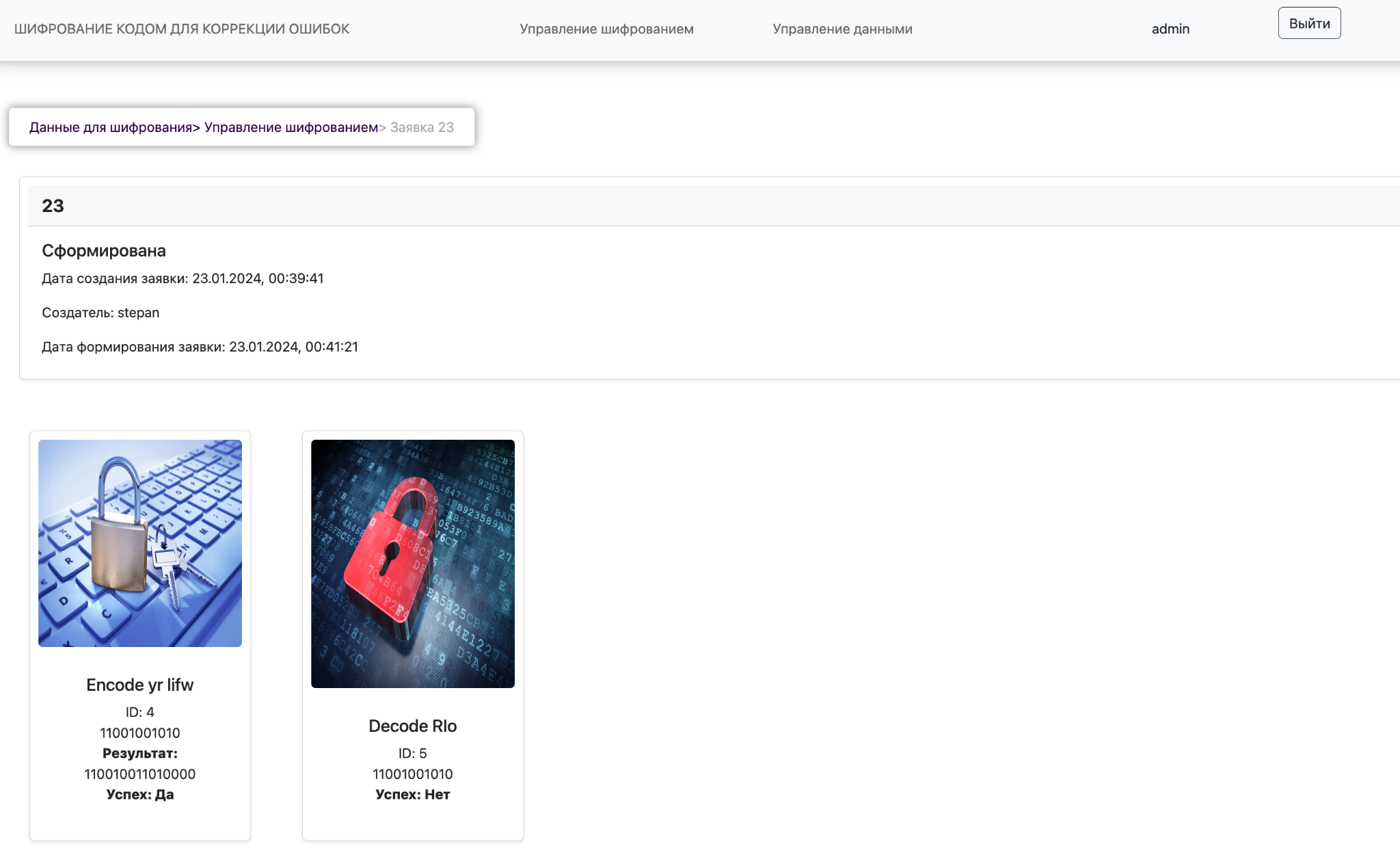


Рисунок 17 - Страница с подробным описанием заявки

На странице с таблицей исходных данных (рис. 18) модератор может в компактном и удобном формате просмотреть список всех исходных данных, существующих в системе. Также можно открыть существующие данные для редактирования или перейти на страницу создания новых исходных данных (рис. 19). Присутствует фильтрация по названию.

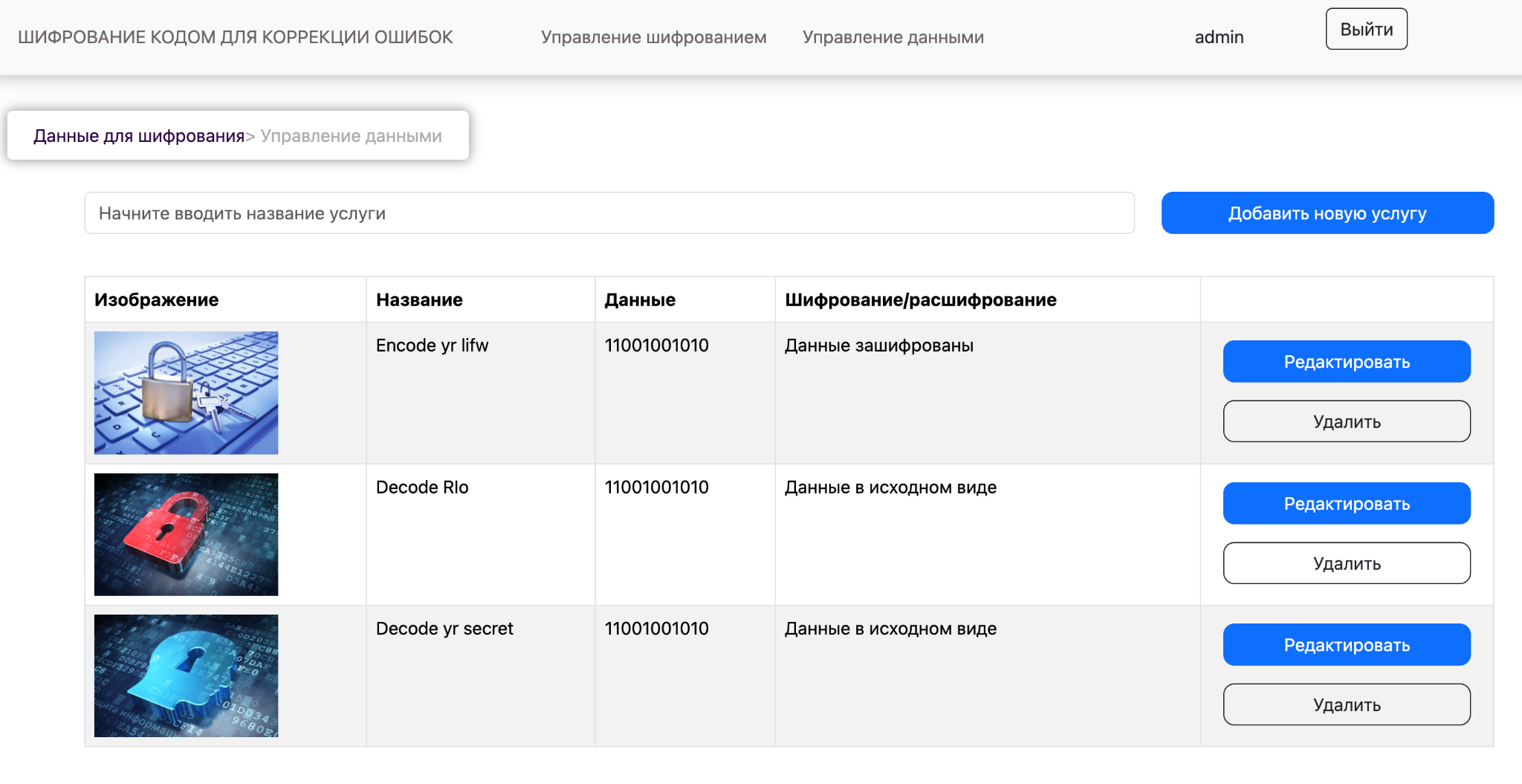


Рисунок 18 - Страница с таблицей исходных данных

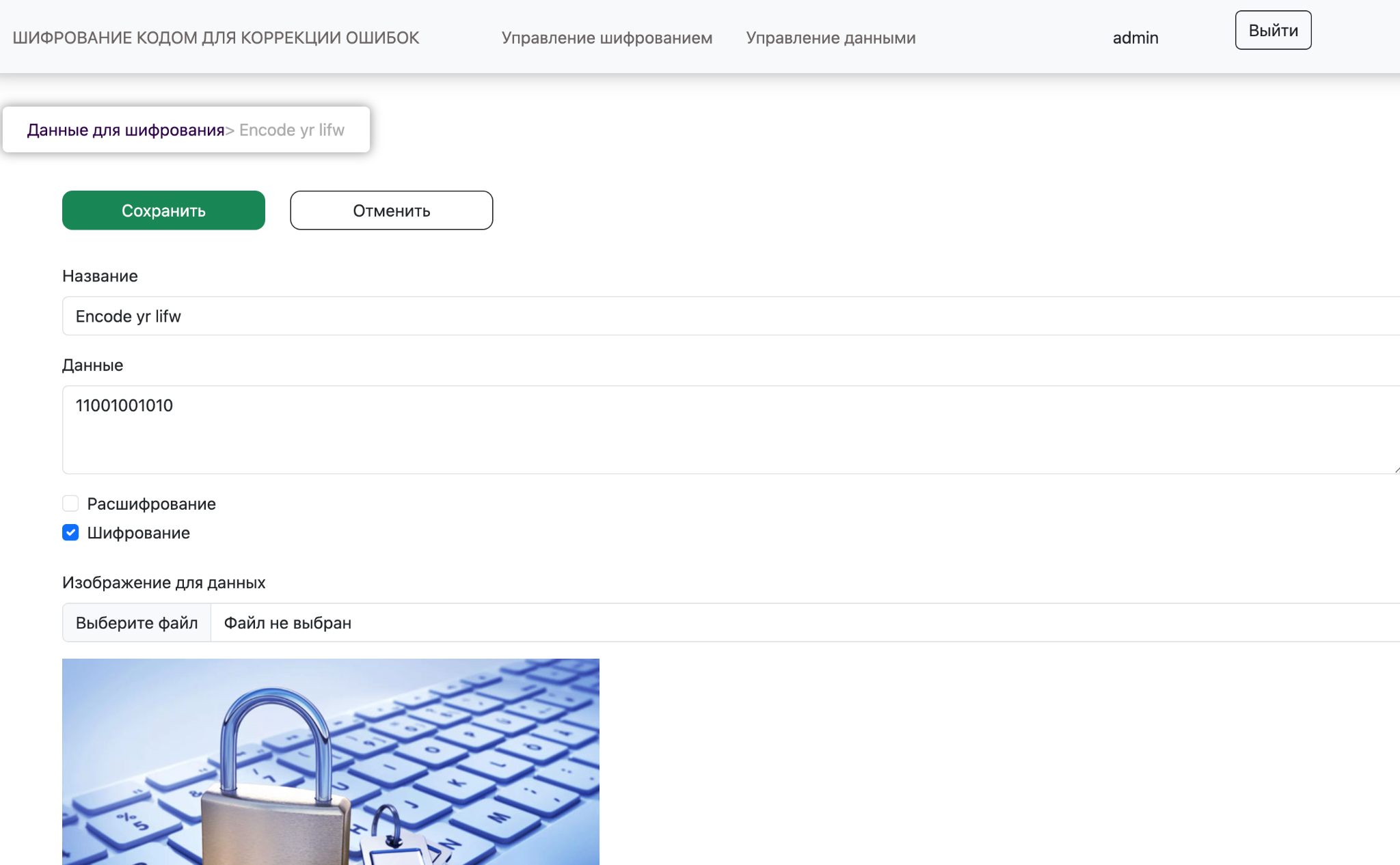


Рисунок 19 - Страница редактирования исходных данных

На странице редактирования/создания исходных данных (рис. 19) модератор может отредактировать существующие исходные данные или создать новые. Для редактирования доступны все поля: название, сами исходные данные, поле, информирующее данные для шифрования или для расшифровки, и изображение, связанное с данными.

# **ЗАКЛЮЧЕНИЕ**

В ходе работы были достигнуты следующие результаты:

1. Был разработан дизайн приложения с помощью набора стилей CSS и HTML тегов.
2. База данных была создана и расположена в docker контейнере.
3. Был создан веб-сервис на Golang, с использованием веб-фреймворка Gin.
4. Разработан интерфейс гостя с использованием технологии React Framework и подключен к веб-сервису.
5. Приложение интерфейса было развернуто на сервисе Github Pages по ссылке https://stepanpopov.github.io/rip\_frontend/.
6. В веб-сервис добавлена авторизация через JWT, а методы задокументированы через Swagger.
7. Реализован интерфейс пользователя. Доступ к нему имеют только авторизованные пользователи.
8. Выделенный сервис был разработан и развернут в отдельном виртуальном окружении Python.
9. Реализован интерфейс модератора для подтверждения новых заявок на кодирование/исправление ошибок и редактирования/добавления исходных данных.
10. Было реализовано мобильное приложение на React Native, повторяющее интерфейс веб-приложения на React.
11. Подготовлен набор документации, включающий РПЗ, ТЗ и набор UML диаграмм.
12. Исходный код проекта доступен в GitHub https://github.com/stepanpopov/rip

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

* 1. Код Хэмминга. [Электронный ресурс] // Wiki. URL: https://en.m.wikipedia.org/wiki/Hamming\_code
  2. Код Хэмминга. Пример работы алгоритма [Электронный ресурс] // Habr. URL:

https://habr.com/ru/articles/140611/

* 1. Raid-массивы [Электронный ресурс] // ITMO. URL: https://de.ifmo.ru/bk\_netra/page.php?tutindex=28&index=77&layer=2
  2. CRC [Электронный ресурс] // Wiki. URL: https://ru.m.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9\_%D0%B8%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9\_%D0%BA%D0%BE%D0%B4
  3. Руководство по Go [Электронный ресурс] // GoLang. URL: https://metanit.com/go/tutorial/ (дата обращения: 11.09.2023)
  4. Руководство по Gin Web Framework [Электронный ресурс] // GoLang. URL: https://github.com/gin-gonic/gin (дата обращения: 12.09.2023).
  5. Руководство по Gorm [Электронный ресурс] // Gorm. URL: https://gorm.io/docs/index.html (дата обращения: 14.10.2023)
  6. Полное практическое руководство по Docker [Электронный ресурс] // Habr. URL: https://habr.com/ru/articles/310460/ (дата обращения: 10.10.2023).
  7. Руководство по React [Электронный ресурс] // Metanit. URL: https://metanit.com/web/react/ (дата обращения: 11.11.2023).
  8. Quick Start – React [Электронный ресурс] // React. URL: https://react.dev/learn (дата обращения: 11.11.2023).
  9. Руководство по React Bootstrap [Электронный ресурс] // React Bootstrap. URL: <https://react-bootstrap.netlify.app/docs/getting-started/introduction>
  10. Руководство по React Native [Электронный ресурс] // React Nativ. URL: <https://reactnative.dev/docs/getting-started>
  11. Руководство по PostgreSQL [Электронный ресурс] // Metanit. URL: https://metanit.com/sql/postgresql/ (дата обращения: 05.10.2023).

**Приложение. Техническое задание**

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

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Дисциплина «Разработка интернет-приложений»

Техническое задание

Тема: «Шифрование кодом для коррекции ошибок»

Студент: Попов С.Д.

Группа ИУ5-53Б

Преподаватель: Канев А.И.

2023г.

1. **Цель работы**

Реализовать систему для шифрования/ коррекции ошибок кодом коррекции ошибок, включающую в себя веб-сервис, веб-приложение, десктопное приложение и выделенный сервис для шифрования.

1. **Назначение**

Система предназначена для клиентов, модераторов и администраторов.

Для получения доступа пользователю необходимо создать заявку на получение результата шифрования/коррекции ошибок выбранных данных. Система предоставляет автоматизированный способ создания, учета и ведения заявок. Также она позволяет модераторам принимать или отклонять заявки. Модераторы имеют возможность редактировать существующие данные или создавать новые.

1. **Задачи**:
   1. Разработать дизайн приложения.
   2. Создать базу данных в PostgreSQL.
   3. Создать веб-сервис на языке Golang 1.20.
   4. Реализовать интерфейс гостя на технологии React.
   5. Развернуть веб-приложение React на Github Pages.
   6. Добавить авторизацию и аутентификацию в веб-сервис.
   7. Реализовать интерфейс пользователя в React.
   8. Реализовать интерфейс модератора в React.
   9. Создать мобильное приложение на React Native.
   10. Создать асинхронный сервис для расчета кода коррекции ошибок на Python.
   11. Подготовить набор документации, включающий РПЗ, ТЗ и набор диаграмм.
2. **Методы веб-сервиса:**

| **№** | **Метод** | **Описание** | **Url** | **Входные данные** | **Выходные данные** |
| --- | --- | --- | --- | --- | --- |
| * 1. Методы авторизации и аутентификации | | | | | |
| 4.1.1. | POST | Регистрация | /api/auth/register | login: string  password: string |  |
| 4.1.2. | POST | Вход в аккаунт | /api/auth/login | login: string  password: string | jwt token |
| 4.1.3. | GET | Выход из аккаунта Доступно только авторизированным пользователям | /api/auth/logout | jwt token |  |
| * 1. Методы данных-услуг для шифрования/расшифровки | | | | | |
| 4.2.1. | GET | Возвращает список данных, название которых подходит под введенный параметр, а также id черновика-заявки при его наличии и при наличии jwt токена. | /api/dataService/ | dataname: string  jwt token | {  draftID: int,  [  dataID: int,  dataName: string,  encode: bool,  blob: string,  active: bool,  imageURL: string  ]  } |
| 4.2.3. | GET | Возвращает данные с нужным id | /api/dataService/{ id } | Id: int | {  dataID: int,  dataName: string,  encode: bool,  blob: string,  active: bool,  imageURL: string  } |
| 4.2.4. | POST | Добавляет нужные данные в заявку-черновик. Если черновика не существует, то он создастся. Доступно только для авторизированных клиентов. | /api/dataService/draft/{id} | id: int | {  draftID: int  } |
| 4.2.5. | POST | Создает новые данные для шифрования/расшифровки.  Доступно только администраторам | /api/dataServices/ | jwt token  {  dataName: string,  encode: bool,  blob: string,  imageURL: string  } | {  dataID: int  } |
| 4.2.6. | PUT | Обновляет информацию о данных. Доступно только администраторам | api/dataServices/{id} | jwt token  id: int  {  dataName: string,  encode: bool,  blob: string,  imageURL: string  } | {  dataID: int  } |
| 4.2.7. | DELETE | Удаляет данные. Доступно только администраторам | /api/dataServices/{id} | jwt token  id: int | {  message: “deleted”  } |
| 4.2.8. | POST | Добавляет картинку к данным. Доступно только администраторам | api/dataService/{id}/image | jwt token  id: int | {  message: “uploaded”  } |
| * 1. Методы заявок для шифрования/коррекции ошибок | | | | | |
| 4.3.1. | GET | Возвращает список заявок пользователя.  Доступно только авторизированным пользователям | api/encryptDecryptRequest/filter | jwt token  status: string,  start\_date: date,  end\_date: date | {  requestID: int,  status: int,  creationDate: date,  finishDate: date,  formDate: date,  moderator: string,  creator: string  } |
| 4.3.2. | GET | Возвращает подробную информацию о заявке вместе с данными-услугами.  Доступно только авторизированным пользователям | api/encryptDecryptRequest/{id} | jwt token  id: int | {  request: {  requestID: int,  status: int,  creationDate: date,  finishDate: date,  formDate: date,  moderator: string,  creator: string  },  Data: [  {  requestID: int,  status: int,  creationDate: date,  finishDate: date,  formDate: date,  moderator: string,  creator: string  }  ]  } |
| 4.3.3. | PUT | Формирует заявку пользователя. Доступно только авторизированным пользователям | api/encryptDecryptRequest/form/{id} | jwt token | {  request: {  requestID: int,  status: int,  creationDate: date,  finishDate: date,  formDate: date,  moderator: string,  creator: string  },  Data: [  {  requestID: int,  status: int,  creationDate: date,  finishDate: date,  formDate: date,  moderator: string,  creator: string  }  ]  } |
| 4.3.4. | DELETE | Удаляет заявку пользователя. Возможно только если заявка находится в статусе “черновик”.  Доступно только авторизированным пользователям | api/encryptDecryptRequest/{id}/ | jwt token  id: int | {  message: “deleted”  } |
| 4.3.5. | DELETE | Удаляет из заявки пользователя услугу.  Доступно только авторизированным пользователям | api/encryptDecryptRequest/{req\_id}/delete/{id} | jwt token  req\_id: int  id: int | {  message: “deleted”  } |
| 4.3.6. | PUT | Отклоняет или завершает заявку.  Доступно только авторизированным модераторам. | api/encryptDecryptRequest /update\_moderator/:id | jwt token  id: int  action: “finish”/”reject” | {  message: string  } |
| 4.3.7 | PUT | Устанавливает рассчитанный код для данных в заявке. Вызывается внешним сервисом. | api/encryptDecryptRequest /update\_calculated/:id | token: string  reqID: int  {  dataID: int  success: bool  result: string  } | {  message: string  } |
| 4.4 Методы сервиса для расчет циклического кода | | | | | |
| 4.4.1. | PUT | Запускает асинхронный расчет циклического кода или кода Хэмминга. Вызывается основным сервисом. | api/calculate | {  "req\_id": number,  "calc": [  {  "id": int,  "data": string,  “encrypt\_type”: string  },  ]  } | {  message: OK  } |
|  | | | | | |

1. **Функциональные требования:**
   1. Главное меню. Здесь находятся кнопки для регистрации и авторизации или информация о текущем пользователе, и меню со списком доступных пунктов.
      1. Доступна всем пользователям.
      2. Действия
         1. Войти – переход на страницу 5.2. Только для гостей.
         2. Зарегистрироваться – переход на страницу 5.3. Только для гостей.
         3. Открыть список двоичных данных – переход на страницу 5.4. Для всех пользователей.
         4. Открыть список заявок на шифрование / коррекции ошибок – переход на страницу 5.6. Только для авторизированных пользователей.
         5. Управление данными – переход на страницу 5.9. Только для модераторов.
   2. Страница с формой авторизации. На этой странице находится форма авторизации.
      1. Доступна гостям
      2. Действия
         1. Войти – производит запрос (метод 4.4.2).
         2. Зарегистрироваться – перенаправляет на страницу 5.3.
   3. Страница с формой регистрации. На этой странице находится форма авторизации.
      1. Доступна гостям
      2. Действия
         1. Зарегистрироваться – производит запрос (метод 4.4.1.), в котором передаются введенные данные формы.
         2. Войти - перенаправляет пользователя на страницу 5.2.
   4. Страница со списком данных в двоичном виде. На этой странице располагается список всех данных и панель фильтрации.
      1. Доступна всем пользователям.
      2. Выводится информация о данных в виде карточек (метод 4.2.1.).
         1. Изображение для данных.
         2. Название данных.
         3. Сама полезная нагрузка данных.
      3. Действия
         1. Отфильтровать двоичные данные по названию (метод 4.2.1.).
         2. «Корзина» - перенаправляет на страницу заявки-черновика 5.7.
         3. Добавить данные в запрос на шифрование / коррекцию ошибок (метод 4.2.4.). Только для авторизированных пользователей.
         4. Подробнее – перенаправляет на страницу 5.5.
   5. Страница с подробным описанием двоичных данных
      1. Доступна всем пользователям.
      2. Выводится подробная информация о двоичных данных.
         1. Название данных.
         2. Сама полезная нагрузка данных.
         3. Зашифрованы или расшифрованы данные
   6. Страница со списком заявок на шифрование / коррекцию ошибок.
      1. Доступна только авторизированным пользователям.
      2. Выводится информация о процессах шифрования / коррекции ошибок в табличном формате (метод 4.3.1).
         1. Создатель. Только для модераторов.
         2. Статус заявки.
         3. Дата создания.
         4. Дата формирования.
         5. Дата завершения.
         6. Количество успешных результатов операций.
      3. Действия
         1. Открыть подробную информацию о заявке– перенаправляет на страницу 5.7.
         2. Установить заявку в статус «отклонено» или «завершено» (метод 4.3.6.). Только для модераторов.
         3. Отфильтровать список заявок по диапазону дат формирования и/или статусу (метод 4.3.1) и/или создателю (только для модератора).
   7. Страница с подробным описанием заявки. Пользователь может изменить набор данных для операций шифрования / коррекции ошибок в заявке. Модератор может просматривать заявки всех пользователей.
      1. Доступна только авторизированным пользователям.
      2. Выводится информация о заявке в карточной форме. Список двоичных данных, в виде карточек (метод 4.3.2.).
         1. Изображение данных.
         2. Название данных.
         3. Сама полезная нагрузка данных.
      3. Действия
         1. Удалить данные из заявки (метод 4.3.5.). Если заявка находится в статусе черновика. Только для создателя заявки.
         2. Сформировать заявку (метод 4.3.3.). Если заявка находится в статусе черновика. Только для создателя заявки.
         3. Удалить заявки (метод 4.3.4.). Если заявка находится в статусе черновика. Только для создателя заявки.
         4. Изменить тип кодирования (циклический код, код Хэмминга). Доступно в режиме редактирования.
         5. Открыть данные из заявки – перенаправляет на страницу 5.5.
   8. Страница редактирования/создания двоичных данных. Модератор может изменить существующие двоичные данные или создать новые.
      1. Доступна только модераторам.
      2. Выводится информация о данных в построчном формате.
         1. Название данных.
         2. Сама полезная нагрузка данных.
         3. Зашифрованы или расшифрованы данные.
         4. Изображение данных.
      3. Изменение полей данных. Можно изменять все вышеперечисленные поля двоичных данных (методы 4.2.5. и 4.2.6.).
   9. Страница с таблицей двоичных данных. Предоставляет модератору удобный способ отображения всех данных.
      1. Доступна только модераторам.
      2. Выводит информацию о двоичных в табличном виде (метод 4.2.1).
         1. Изображение данных.
         2. Название данных.
         3. Сама полезная нагрузка данных.
         4. Id данных.
      3. Действия
         1. Удалить данные (метод 4.2.7)
         2. Редактировать данные – перенаправляет на страницу 5.8
         3. Добавить данные – перенаправляет на страницу 5.8
2. **Требования к программному обеспечению:**
   1. Серверная часть
      1. ОС: Linux 6.5 / Windows 10
      2. Docker 24.0.6
      3. Python 3.11.5
      4. Golang 1.21.1
      5. Докер образы
      6. redis 6.2-alpine
      7. nginx 1.19.2-alpine
      8. minio RELEASE.2022-10-15T19-57-03Z
      9. postgres 16.1
   2. Клиентская часть
      1. ОС: Windows/MacOS/Linux
      2. Веб-браузер: Safari 11.1+/Chrome 40+/Opera 27+/Firefox 44+/Yandex 20+
3. **Требования к аппаратному обеспечению:**
   1. Серверная часть
      1. Процессор минимум 2-ядерный с частотой от 2 ГГц.
      2. Оперативная память от 4 Гб.
      3. Место на жестком диске от 2 Гб.
   2. Клиентская часть
      1. Процессор с частотой от 1ГГц.
      2. Оперативная память от 512 Мб.