**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет *ИСТ* Кафедра *ИВК*

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. кафедрой

/ /

подпись инициалы, фамилия

« » 20 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Тема *Мобильное приложение: Справочник клинической лабораторной диагностики*

Обучающийся / *Алексеенко Д.В.* /

подпись инициалы, фамилия

Обозначение работы *ВКР-УлГТУ-09.03.02-17/567-2022* Группа *ИСТбд-41*

Направление подготовки *09.03.02 «Информационные системы и технологии»*

Руководитель работы / *Тамьяров А.В.* /

подпись, дата инициалы, фамилия

Консультанты:

*экономический раздел* / *М. В. Рыбкина* /

наименование раздела подпись, дата инициалы, фамилия

/ /

наименование раздела подпись, дата инициалы, фамилия

Ульяновск, 2022

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет *ИСТ* Кафедра *ИВК*

Направление подготовки *информационные системы и технологии*

УТВЕРЖДАЮ

Зав. кафедрой

/ /

подпись инициалы, фамилия

« » 2022г.

ЗАДАНИЕ

**на выпускную квалификационную работу**

обучающемуся *Алексеенко Дмитрий Викторовичу* курса *4* группы *ИСТбд-41*

фамилия, имя, отчество

1. Тема работы *Мобильное приложение: Справочник клинической лабораторной диагностики*

утверждена приказом по университету №*\_422\_* от «*09*» марта20*22* г.

1. Срок сдачи обучающимся законченной работы «*20*» *июня* 20*22* г.
2. Исходные данные к работе *разработать мобильное приложение: справочник клинической лабораторной диагностики.*
3. Содержание пояснительной записки *техническое задание; информационное, алгоритмическое, программное обеспечение системы; тестирование системы.*
4. Перечень графического материала (чертежей): *схема алгоритма авторизации пользователя, схема алгоритма поиска статей.*
5. Календарный график работы на весь период (с указанием сроков выполнения и содержания отдельных этапов)

|  |  |  |
| --- | --- | --- |
| № этапа | Содержание этапа | Срок выполнения |
| *1* | *Проектирование архитектуры системы* | *20.03.2022* |
| *2* | *Создание системы* | *14.04.2022* |
| *3* | *Тестирование системы* | *13.05.2022* |
| *4* | *Написание пояснительной записки* | *20.05.2022* |
| *5* | *Создание чертежей* | *08.06.2022* |
| *6* | *Прохождение нормоконтроля* | *20.06.2022* |
| *7* | *Защита* | *23.06.2022* |

1. Консультанты

|  |  |  |  |
| --- | --- | --- | --- |
| Раздел | Ф.И.О. консультанта | Подпись, дата | |
| задание выдал | задание принял |
| Экономический | Рыбкина М.В. |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата выдачи задания « » 2022 г.

Руководитель

директор научно-образовательного центра

«Точка кипения УлГТУ» / *Тамьяров А.В.\_*/

должность, учёная степень, учёное звание подпись инициалы, фамилия

Задание принял к исполнению / *Алексеенко Д.В.*/

подпись инициалы, фамилия

**АННОТАЦИЯ**

**Выпускная квалификационная работа** Алексеенко Дмитрия Викторовича по теме «Мобильное приложение: Справочник клинической лабораторной диагностики». Руководитель: Тамьяров Андрей Валерьевич. Защищена на кафедре «Измерительно-вычислительные комплексы» УлГТУ в 2022 году.

**Пояснительная записка:** 80 страниц, 6 разделов, 1 приложение, 22 рисунков, 27 таблиц, 7 источников.

**Ключевые слова:** мобильное приложение, клинический лабораторный анализ, JavaScript, React native, PHP, Symfony, Docker.

Мобильное приложение «Справочник клинической лабораторной диагностики» разработан для того, чтобы пациент и сотрудники лаборатории имели быстрй доступ к показателям анализов. Приложение дает возможность создавать пользовательские статьи и лейкоцитарный счетчик. Продукт был разработан на языке JavaScript и фреймворке React native на клиенсткой части и на языке PHP и фреймворку Symfony на стороне сервера.

**СОДЕРЖАНИЕ**

[СПИСОК ИСПОЛЬЗОВАННЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ 8](#_Toc106783447)

[ВВЕДЕНИЕ 9](#_Toc106783448)

[1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ СИСТЕМЫ 11](#_Toc106783449)

[1.1 Назначение и цели создания системы 11](#_Toc106783450)

[1.2 Характеристика объекта автоматизации 11](#_Toc106783451)

[1.2.1 Общее описание 11](#_Toc106783452)

[1.2.2 Состав и порядок функционирования 11](#_Toc106783453)

[1.3 Общие требования к системе 12](#_Toc106783454)

[1.3.1 Требования к структуре и функционированию системы 12](#_Toc106783455)

[1.3.2 Дополнительные требования 12](#_Toc106783456)

[1.4 Требования к функциям, выполняемым системой 12](#_Toc106783457)

[1.4.1 Функция авторизации 13](#_Toc106783458)

[1.4.2 Функция регистрации 13](#_Toc106783459)

[1.4.3 Функция добавления статей в избранное 13](#_Toc106783460)

[1.4.4 Функция локального создания статей 14](#_Toc106783461)

[1.5 Требования к видам обеспечения 15](#_Toc106783462)

[1.5.1 Требования к информационному обеспечению 15](#_Toc106783463)

[1.5.2 Требования к алгоритмическому обеспечению 15](#_Toc106783464)

[1.5.3 Требования к программному обеспечению 15](#_Toc106783465)

[1.6 Анализ аналогичных разработок 15](#_Toc106783466)

[1.6.1 Мобильное приложение Справочник анализов 16](#_Toc106783467)

[1.6.2 Мобильное приложение Анализы. Элементарно 16](#_Toc106783468)

[1.6.3 Сравнение аналогов и разрабатываемой системы 17](#_Toc106783469)

[1.6.4 Вывод по проделанному анализу 17](#_Toc106783470)

[2. ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ 18](#_Toc106783471)

[2.1 Выбор средств управления данными 18](#_Toc106783472)

[2.2 Проектирование базы данных 18](#_Toc106783473)

[2.2.1 Концептуальная схема 18](#_Toc106783474)

[2.2.2 Внутренняя схема базы данных 20](#_Toc106783475)

[2.3 Проектирование файлов данных 21](#_Toc106783476)

[2.4 Организация сбора, передачи, обработки и выдачи информации 21](#_Toc106783477)

[3 АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ 22](#_Toc106783478)

[3.1 Алгоритм авторизации 22](#_Toc106783479)

[3.2 Алгоритм поиска статей 22](#_Toc106783480)

[4 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 25](#_Toc106783481)

[4.1 Структура программного обеспечения и функции его компонентов 25](#_Toc106783482)

[4.2 Выбор компонентов программного обеспечения 25](#_Toc106783483)

[4.2.1 Операционная система 25](#_Toc106783484)

[4.2.2 Инструментальное средство разработки 26](#_Toc106783485)

[4.2.3 Вспомогательное программное обеспечение 27](#_Toc106783486)

[4.3 Разработка прикладного программного обеспечения 27](#_Toc106783487)

[4.3.1 Структура прикладного программного обеспечения 27](#_Toc106783488)

[4.3.2 Серверная часть 29](#_Toc106783489)

[4.3.2.1 Роут /api/auth 29](#_Toc106783490)

[4.3.2.2 Роут /api/registration 29](#_Toc106783491)

[4.3.2.3 Роут /api/articles 29](#_Toc106783492)

[4.3.2.4 Роут /api/add/favorite/article 30](#_Toc106783493)

[4.3.2.5 Роут /api/get/favorite/articles 30](#_Toc106783494)

[4.3.3 Клиентская часть 30](#_Toc106783495)

[4.3.3.1 Экран AuthorizationScreen 30](#_Toc106783496)

[4.3.3.2 Экран RegistrationScreen 30](#_Toc106783497)

[4.4 Особенности реализации, эксплуатации и сопровождения системы 31](#_Toc106783498)

[4.5 Руководство пользователя 32](#_Toc106783499)

[4.5.1 Требования к условиям эксплуатации 32](#_Toc106783500)

[4.5.2 Инициализация и настройка 32](#_Toc106783501)

[4.5.3 Порядок и особенности работы 32](#_Toc106783502)

[4.5.4 Условия и порядок тестирования 41](#_Toc106783503)

[5 ТЕСТИРОВАНИЕ СИСТЕМЫ 45](#_Toc106783504)

[5.1 Условия и порядок тестирования 45](#_Toc106783505)

[5.2 Исходные данные для контрольных примеров 45](#_Toc106783506)

[5.2.1 Авторизация пользователя 45](#_Toc106783507)

[5.2.2 Регистрация пользователя 45](#_Toc106783508)

[5.2.3 Добавление статьи в избранное 45](#_Toc106783509)

[5.3 Результаты тестирования 46](#_Toc106783510)

[6 ЭКОНОМИЧЕСКИЙ РАЗДЕЛ 47](#_Toc106783511)

[6.1 Определение трудоемкости разработки системы 47](#_Toc106783512)

[6.2 Затраты на разработку системы 48](#_Toc106783513)

[6.2.1 Затраты на материальные ресурсы 48](#_Toc106783514)

[6.2.2 Затраты на электроэнергию 48](#_Toc106783515)

[6.2.3 Затраты на оплату труда 49](#_Toc106783516)

[6.2.4 Затраты на обязательные взносы 50](#_Toc106783517)

[6.2.5 Затраты на амортизационные отчисления 51](#_Toc106783518)

[6.3 Себестоимость проекта 51](#_Toc106783519)

[6.4 Определение экономической эффективности проекта 52](#_Toc106783520)

[6.5 Выводы по технико-экономическому анализу и обоснованию проекта разработки 53](#_Toc106783521)

[ЗАКЛЮЧЕНИЕ 55](#_Toc106783522)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 56](#_Toc106783523)

[ПРИЛОЖЕНИЕ А 57](#_Toc106783524)

**СПИСОК ИСПОЛЬЗОВАННЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ**

ЛС – лейкоцитарный счетчик.

СУБД – средства управления базой данных.

SQL – Structured query language (язык структурированных запросов).

SSH – Secure Shell (сетевой протокол удаленного доступа).

**ВВЕДЕНИЕ**

В настоящее время информационные технологии все больше появляются в самых разных сферах жизнедеятельности. Мобильные устройства в сегодняшней медицине играют важную роль и значительно изменило многие аспекты современной клинической практики. В настоящее время на рынке 70% мобильных приложений посвящены спорту и здоровью и предназначены для домашнего использования, а 30% приложений представляют пользу для профессионального использования в медицинской сфере.

Использование различных гаджетов для получения информации становится не новшеством, а активным помощником в учреждениях здравоохранения, что привело к быстрому развитию рынка медицинский приложений для различных платформ. Самыми актуальными платформами на сегодняшний день является мобильные информационные системы, что не удивительно, ведь смартфон наиболее удобен для быстрого поиска информации.

На сегодняшний день справочно-информационные системы и приложения для врачей являются одним из наиболее актуальных ветвей развития мобильного здравоохранения во всем мире. Преимуществами такой системы являются упрощённый доступ к медицинскому справочнику и статьям, которые помогают в принятии обоснованных клинических решений и исключение человеческого фактора при получении результата.

В данный момент в интернете представлено крайне мало бесплатных мобильных приложений, которые предоставляют достоверную информацию быстро и в удобном формате. Дело в том, что прибыль от данных приложений не покрывает затраты на его разработку, а добавление платного контента зачастую не оправдывает себя и негативно воспринимается врачами.

Наибольшую потребность в медицинских мобильных приложениях испытывает сфера лабораторного анализа. Точность в данных приложениях крайне важна, а информация, представленная на данных ресурсах, требует достоверности и особенно актуальна для начинающих лаборантов.

Создание справочника клинической лабораторной диагностики позволит решить данные проблемы и быстро находить требуемую информацию через поиск, а авторизация в данном приложении поможет просматривать сохраняемые статьи не только на разных устройствах, но и мобильных платформах. Также, интересна функция добавления своих статей в приложение, что позволяет в одном месте собирать требуемую информацию и быстро получать доступ к ней.

# **1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ СИСТЕМЫ**

**1.1 Назначение и цели создания системы**

Разрабатываемое мобильное приложение предназначенно для быстрого и удобного доступа к справочным материалам, автоматизированного процесса создания пользовательских статей и возможностям лейкоцитарного счетчика.

Цели создания системы:

* упрощение доступа к справочным материалам;
* упрощение доступа к ЛС;
* автоматизация процесса создания статей.

**1.2 Характеристика объекта автоматизации**

1.2.1 Общее описание

Объектом автоматизации является процесс создания локальных пользовательских статей. Автоматизация этого процесса позволит пользователям создавать статьи с данными, которые они считают подтвердженными и использовать в работе

1.2.2 Состав и порядок функционирования

Для использования информационной системы необходими пройти авторизацию. После этого у пользователя будет доступ ко всем функциям приложения.

Чтобы создать статью, необходимо выбрать соответствующий пункт меню. Далее откроется форма для заполнения статьи данными, которые проверяются на наличие ошибок. После заполнения, пользователю будет доступа его статья для чтения.

**1.3 Общие требования к системе**

1.3.1 Требования к структуре и функционированию системы

Система должна быть реализована в виде кроссплатформенного мобильного приложения, которое сможет использовать любой пользователь, выполнивший вход.

В соста системы должны входить следующие подсистемы:

* подсистема авторизации и регистрации;
* подсистема ЛС;
* подсистема поиска статей.

В качестве дальнейшего развития системы, предполагается возможность создания статей для всех пользователей и калькулятор приготовления растворов.

1.3.2 Дополнительные требования

Мобильное приложение должно сохранять работоспособность и обеспечивать восстановление своих функци при возникновении внештатных ситуаций.

Унификации подлежат все составляющие пользовательского интерфейса: формы отображения информации на экране, типы и формы сообщений и запросов пользователей.

Разрабатываемая система должна использовать персональные сведения только в рамках системы, и не передавать их третьим лицам.

Так же система должна быть изолирована в среде развертывания системы, чтобы увечить отказоустойчивость и восстановление при сбоях.

**1.4 Требования к функциям, выполняемым системой**

В системе должны быть реализованы следующие функции:

* функция авторизации;
* функция регистрации;
* функция локального создания статей;
* функция добавление статей в избранное.

1.4.1 Функция авторизации

Для осуществления авторизации, пользователю необходимо ввести данные своей учетной записи. После авторизации, пользователь переходить на экран статей.

**Сроки реализации функции**

Функция должна быть реализована в первую очередь.

**Требования к входным данным**

* строка с логином пользователя;
* строка с паролем пользователя.

**Требования к выходным данным**

* временный идентификатор пользователя в системе.

1.4.2 Функция регистрации

Для осуществления регистрации в мобильном приложении, пользователю необходимо ввести логин и пароль.

**Сроки реализации функции**

Функция должна бать реализована в первую очередь.

**Требования к входным данным**

* строка с логином пользователя;
* строка с паролем пользователя.

**Требования к выходным данным**

* временный идентификатор пользователя в системе.

1.4.3 Функция добавления статей в избранное

Для того, чтобы добавить статью в избранное необходимо передать идентификатор выбранной статьи и идентификатор пользователя системы.

**Сроки реализации функции**

Функция должна быть реализована в последную очередь.

**Требования к входным данным**

* идентификатор статьи;
* идентификатор пользователя.

**Требования к выходным данным**

* обновленный список избранных статей.

1.4.4 Функция локального создания статей

Что бы локально добавить статью, необходимо получить детальную информацию о статье от пользователя и провести проверку введенных данных.

**Сроки реализации функции**

Функция должна быть реализована в последную очередь

**Требования к входным данным**

* строка с заголовком статьи;
* строка с категорией статьи;
* строка с нормой статьи;
* строка с описанием статьи.

**Требования к выходным данным**

Выходные данные для данной функции не требуются.

Данная функция позволяет администратору удалить компанию.

Входные данные:

* id (численный идентификатор) компании.

После удаления компании должны также быть удалены все связанные с ней данные: аккаунт, стажировки, контактные лица.

**1.5 Требования к видам обеспечения**

1.5.1 Требования к информационному обеспечению

Необходимо наличие базы данных для хранения следующей информации:

* данные о пользователях;
* данные о статьях.

Получить данные имеет возможность пользователь, который авторизовался в системе.

1.5.2 Требования к алгоритмическому обеспечению

Требования к алгоритмическому обеспечению не предъявляются.

1.5.3 Требования к программному обеспечению

Мобильное приложение должно поддерживаться современными смартфонами на платформах Android и iOS.

При разработке клиентской части необходимо использовать язык программирования JavaScript и фреймворк React native. Для разработки серверной части необходимо использовать язык программирования PHP и фреймворк Symfony. Так же необходимо использовать программное обеспечение для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации.

**1.6 Анализ аналогичных разработок**

Перед началом работы было необходимо определиться со стеком технологий, на котором будет реализован продукт, так как от выбранного стека зависит производительность и гибкость разрабатываемой системы. Также было необходимо определить, какой функционал должна реализовывать разрабатываемая система, чтобы выделиться на фоне других программных продуктов в данной сфере.

1.6.1 Мобильное приложение Справочник анализов

Мобильное приложение «Справочник анализов» предоставляет доступ к справочным материалам клиническо-лабораторных анализов. Для удобной навигации по приложению существует функция поиска показателя.

Достоинства:

* простой и понятный интерфейс;
* кроссплатформенность;
* бесплатное использование;
* наличие поиска.

Недостатки:

* отстутствует раздел избранное;
* редкое обновление базы данных;
* малый функционал.

1.6.2 Мобильное приложение Анализы. Элементарно

Мобильное приложение «Анализы. Элементарно» предоставляет доступ к справочным материалам клинических лабораторных анализов, распределенных по возрастам. В приложении реализован поиск для удобной навигации. Также сущестует возможность сохранения показателя в избранное.

Достоинства:

* современный интерфейс;
* раздел избранное;
* бесплатное использование.

Недостатки:

* отсутсвует версия под ОС Android;
* вылеты приложение;

1.6.3 Сравнение аналогов и разрабатываемой системы

Сравнение аналогов и разрабатываемой системы по ряду критериев представлено в таблице 1.

Таблица 1 – Сравнение аналогов и разрабатываемой системы

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование/критерий** | **Справочник анализов** | **Анализы. Элементарно** | **Разрабатываемая**  **ИС** |
| Современный интерфейс | − | + | − |
| Кроссплатформенность | + | − | + |
| Раздел избранное | − | + | + |
| Бесплатное использование | + | + | + |
| Лейкоцитарный счетчик | − | − | + |
| Создание пользовательских статей | − | − | + |

1.6.4 Вывод по проделанному анализу

Таким образом разрабатываемое мобильное приложение представляет собой уникальную разработку, которая позволяет пользователям просматривать и добавлять в избранное показатели анализов. Для работников лабораторий приложение предлагает дополнительные возможности такие, как лейкоцитарный счетчик и локальное добавление статей.

**2 ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ**

2.1 Выбор средств управления данными

В разрабатываемом мобильном приложении, для организации хранения и обработки данных было принято решение использовать систему управления базами данных MySQL.

MySQL является одной из самых популярных систем управления базами данных. Эта система имеет открытый исходный код и является полностью бесплатной.

Данная СУБД является реляционной, что в нашем случае имеет больше плюсов, чем минусов. Реляционные базы данных очень плохо работают с неструктурированными данными, для этого отлично подходят решения, использующие NoSQL, но в нашем случае такой необходимости нет.

Реляционные базы данных позволяет сделать логические связи таблиц, чтобы не было необходимости отслеживать их зависимости. Например, при использовании MongoDB, которая является не реляционной базой данных, приходится вручную связывать таблицы. Все это заставляет потратить большое количество времени разработчиков для создания необходимой структуры.

Так же MySQL намного лучше работает с большими объемами данных, чем их не реляционные аналоги. Все это делает данную СУБД наиболее подходящим для использования в разрабатываемом приложении.

2.2 Проектирование базы данных

2.2.1 Концептуальная схема

Концептуальная схема базы данных представлена на рисунке 1.

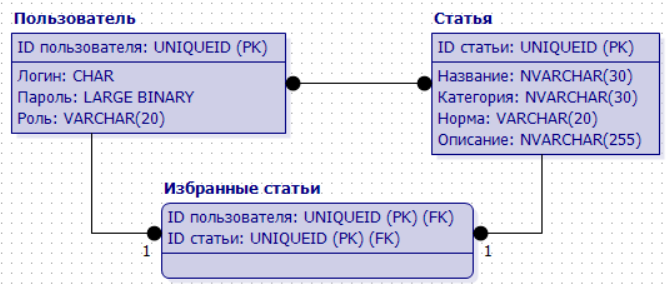


Рисунок 1 – Концептуальная схема базы данных в нотации IDEF1X

Сущности концептуальной схемы представлены в таблице 2.

Таблица 2 – Сущности концептуальной схемы

|  |  |  |
| --- | --- | --- |
| **№** | **Название сущности** | **Описание** |
| 1 | Пользователи | Сведение о пользователях, использующие данную систему |
| 2 | Статьи | Сведения о статьях |
| 3 | Избранных статьи | Сведения о избранных статьях пользователей |

Атрибуты сущностей и их описание представлены в таблицах 3 – 5.

Таблица 3 – Атрибуты сущности Пользователи

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Тип** | **Описание** |
| 1 | ID пользователя | Уникальный идентификатор | Уникальный автоматически генерируемый идентификатор пользователя |
| 2 | Логин | Текстовый | Логин пользователя системы |
| 3 | Пароль | Бинарный массив | Пароль пользователя системы |
| 4 | Роль | Текстовый | Роль пользователя системы |

Таблица 4 – Атрибуты сущности Статьи

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Описание** |
| 1 | ID статьи | Уникальный идентификатор | Уникальный автоматически генерируемый идентификатор статьи |
| 2 | Название | Текстовый | Название статьи |
| 3 | Категория | Текстовый | Категория статьи |
| 4 | Норма | Текстовый | Нормальное значение статьи |
| 5 | Описание | Текстовый | Описание статьи |

Таблица 5 – Атрибуты сущности Избранные статьи

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Описание** |
| 1 | ID пользователя | Уникальный идентификатор | Уникальный автоматически генерируемый идентификатор пользователя |
| 2 | ID статьи | Уникальный идентификатор | Уникальный автоматически генерируемый идентификатор статьи |

2.2.2 Внутренняя схема базы данных

Физическая модель базы данных, разработанная по концептуальной схеме, представлена в таблицах 6 – 8.

Таблица 6 – Таблица физической модели users (Пользователи)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | id [ID пользователя] | uuid | uuid\_generate\_v4() | от 1 до 2^63 |
| 2 | login [Логин] | character varying | Нет | Без ограничений |
| 3 | password [пароль] | character varying | Нет | Без ограничений |
| 4 | role [Роль] | character varying | Нет | Без ограничений |

Таблица 7 – Таблица физической модели articles (Статьи)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | id [ID статьи] | uuid | uuid\_generate\_v4() | от 1 до 2^63 |
| 2 | title [Название] | character varying | Нет | Без ограничений |
| 3 | category [Категория] | character varying | Нет | Без ограничений |
| 4 | norm [Норма] | character varying | Нет | Без ограничений |
| 5 | description [Описание] | character varying |  | Без ограничений |

Таблица 8 – Таблица физической модели favorite\_articles (Избранные статьи)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название поля** | **Тип и размер** | **Значение по умолчанию** | **Допустимые значения** |
| 1 | id [ID пользователя] | uuid | uuid\_generate\_v4() | от 1 до 2^63 |
| 2 | id [ID статьи] | uuid | uuid\_generate\_v4() | от 1 до 2^63 |

Индексы таблиц представлены в таблицах 9 – 11.

Таблица 9 – Индексы таблицы users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название индекса | Поля индекса | Порядок сортировки | Уникальность |
| 1 | <Первичный ключ> | id | По возрастанию | да |

Таблица 10 – Индексы таблицы articles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название индекса | Поля индекса | Порядок сортировки | Уникальность |
| 1 | <Первичный ключ> | id | По возрастанию | да |

Таблица 11 – Индексы таблицы favorite articles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название индекса | Поля индекса | Порядок сортировки | Уникальность |
| 1 | <Вторичный ключ> | id | По возрастанию | да |
| 2 | <Вторичный ключ> | id | По возрастанию | да |

2.3 Проектирование файлов данных

Файлы данных в системы не применяются.

2.4 Организация сбора, передачи, обработки и выдачи информации

Для осуществления сбора информации от пользователя используется клиентская часть приложения. Передача информации реализована при помощи протокола HTTP. Используемая архитектура – REST, так как её функционал полностью покрывал все необходимые случаи реализации. Так же изначально рассматривалась архитектура GraphQL, которая позволяет запрашивать данные из базы данных по выборочным полям, но это вызвало бы лишнюю зависимость, необходимости в которой не было бы.

Первичная предобработка данных и приведение их в нужный вид происходит на клиентской части приложения. Валидация и хранение данных в БД реализуется на серверной части приложения.

3 АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СИСТЕМЫ

3.1 Алгоритм авторизации

**Общая характеристика:** алгоритм выполняет функцию проверки авторизации и утентификации пользователя.

**Используемые данные:** логин и пароль пользователя из формы авторизации.

**Результаты выполнения:** результатом выполнения данного алгоритма можеть быть два варианта, либо пользователь прошел авторизацию, либо получил сообщение о неудачной попытке входа. После успешной авторизации, пользователь системы переходит на экран статей.

**Логическое описание:** блок-схема алгоритма авторизации представлена на рисунке 2.

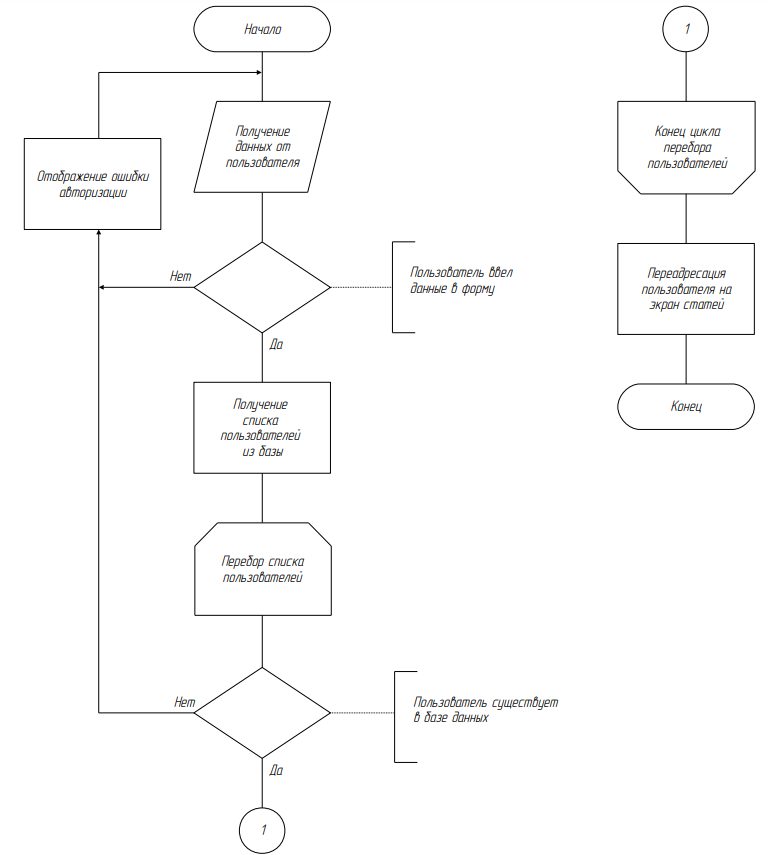
3.2 Алгоритм поиска статей

**Общая характеристика:** алгоритм предназначен для поиска и вывода статей, удовлетворяющих условиям поиска.

**Используемые данные:** веденная пользователем строка для поиска.

**Результаты выполнения:** результатом выполнения алгоритма является получение списка статей, удовлетворяющих условиям поиска

**Логическое описание:** блок-схема алгоритма поиска представлена на рисунке 3.

Рисунок 2 – Алгоритм авторизации

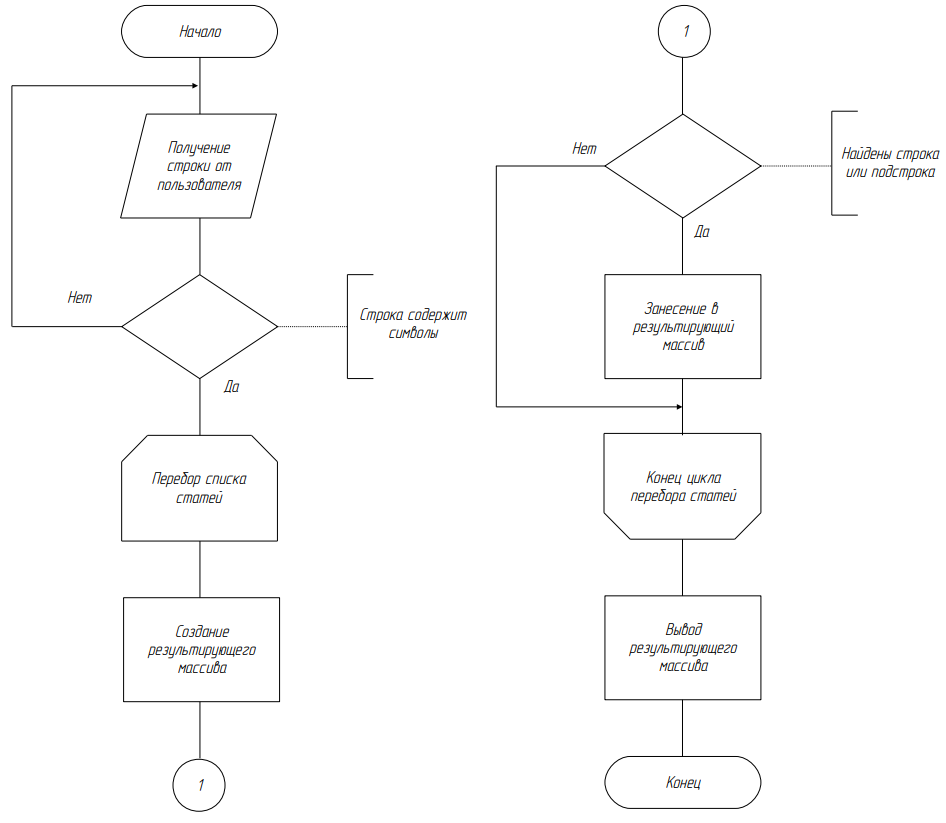


Рисунок 3 – Алгоритм поиска

4 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

4.1 Структура программного обеспечения и функции его компонентов

Данная система представляет клиент-серверное мобильное приложение. Для разработки клиентской части применялся язык JavaScript с использованием основной библиотеки React native. Серверная часть написана на языке PHP с использованием фреймворка Symfony. В качестве веб-сервера и прокси-сервера был использован Nginx. Так же было использовано ПО для автоматизации управления, развертывания и контейнеризации приложений Docker, и инструмент для запуска много контейнерных приложений docker-compose.

Клиентская часть мобильного приложения доступна для любого современного смартфона под управлением ОС Android и iOS. Доступ к серверной части и базе данных возможен через протокол SSH к серверу, на котором развернуто приложение.

4.2 Выбор компонентов программного обеспечения

4.2.1 Операционная система

Для данного приложения подходит любая современная операционная система, которая поддерживает систему контейнеризации Docker.

Операционные системы семейства Windows являются платными, при этом не несут никакой пользы для использования нашей системы. Так же Windows потребляет большое количество ресурсов сервера в фоновом режиме и занимает большое количество памяти по сравнению с операционными системами семейства UNIX.

Выбирая из UNIX-подобных операционных систем, было принято решение использовать операционную систему Ubuntu, так как она является самой популярной из своего семейства. Это говорит о том, что у данной системы развитая поддержка разнообразного оборудования, высокая безопасность и многочисленное сообщество, которое поможет с решением различных возникающих проблем.

4.2.2 Инструментальное средство разработки

В качестве инструментальной среды разработки был выбран редактор кода Visual Studio Code. На данный момент Visual Studio Code является одним из самых популярных редактором кода и с открытым исходным кодом. Большое сообщество позволяет использовать сторонние модули и плагины для удобства при разработке. Одним из аналогов является JetBrains WebStorm, который представляет собой IDE, включающий в себя набор специализированных инструментов. WebStorm является платным решением и требует больше производительных мощностей.

Для разработки клиентской части мобильного приложения был использован язык программирования JavaScript с использованием фреймворка React native. Это ускоряет разработку мобильного приложения и позволяет сделать его более оптимизированным. Среди популярных решений так же существует фреймворк Flutter. Данный фреймворк лучше подходит для мобильных приложений с более сложной логикой. Но он мене гибок, так как имеет меньшее количество дополнительны вспомогательных инструментов.

Для серверной части было решено использовать язык Python. Так же рассматривался язык Ruby с фреймворком Ruby on Rails, но сообщество данного языка и фреймворка меньше чем в python. Среди основных Python фрейммворков был выбор между Sanic и Django. Было решено использовать Sanic, так как он является более быстрым за счет реализации ее асинхронной работы запросов.

Для серверной части было решено использовать язык программирования PHP. Так же рассматривался язык программирования Ruby с фреймворком Ruby on Rails, но сообщество данного языка программирования и фреймворка меньше чем в PHP.

Среди основных PHP фреймворков был выбор между Symfony и Laravel. Было принято решение использовать Symfony, так как он является одним из самых первых фреймворков и обладает крупным сообществом.

4.2.3 Вспомогательное программное обеспечение

Для просмотра результатов необходимо использовать современный смартфон под управлением операционной системы Android или iOS.

Для работы приложения требуется веб–сервер. Для развертывания приложения в рабочем окружении была настроена и протестирована конфигурация NGINX в связке с системой автоматизации развертывания и контейнеризации Docker, которая представляет из себя набор контейнеров, содержащих минимально необходимую конфигурацию для функционирования отдельных блоков системы, таких как база данных MySQL, клиент часть приложения и серверная часть приложения.

Данная конфигурация позволяет облегчить процесс развертывания приложения вне зависимости от сервера, на котором оно будет разворачиваться. Все что нужно – это наличие последних версий Docker и Docker-compose на сервере.

4.3 Разработка прикладного программного обеспечения

4.3.1 Структура прикладного программного обеспечения

Структура программного продукта предоставлена на рисунке 4.

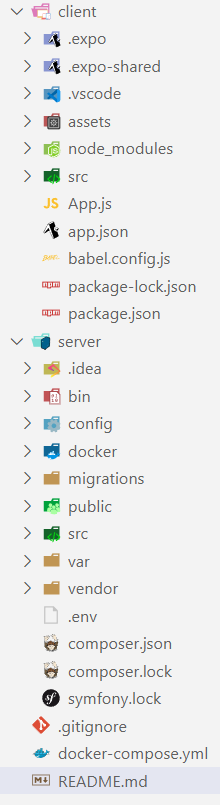


Рисунок 4 – Структура программного продукта

Каталог client содержит клиентскую часть мобильного приложения. Он содержит подкаталоги src, assets и node\_modules. Подкаталог src содержит в себе исходный код клиентской части мобильного приложения. Он содержит в себе каталоги такие как:

* api – содержит файлы, отвечающие за работу с серверной частью приложения;
* components – содержит компоненты приложения;
* navigation – содержит компоненты навигации приложения;
* screens – содержит сложные компоненты, которые являются экранами приложения;
* store – содержит компоненты глобального хранилища приложения.

Каталог server содержит серверную часть мобильного приложения. Он содержит подкаталог src, в котором храниться исходный код серверной части. В подкаталоге src находятся подкаталоги Entity и Controller. Подкаталог Entity содержит все сущности базы данных, используемых мобильным приложением. Подкаталог Controller отвечает за обработку запросов с клиентской части.

4.3.2 Серверная часть

4.3.2.1 Роут /api/auth

Обрабатывает POST-запрос на адресе /api/auth описание приведено в таблице 12.

Таблица 12 – Описание роута /api/auth

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | login() | Авторизация пользователя |

4.3.2.2 Роут /api/registration

Обрабатывает POST-запрос на адресе /api/registration описание приведено в таблице 13.

Таблица 13 – Описание роута /api/registration

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | registration() | Регистрация пользователя |

4.3.2.3 Роут /api/articles

Обрабатывает GET-запрос на адресе /api/articles описание приведено в таблице 14.

Таблица 14 – Описание роута /api/articles

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | getAllArticles() | Получение всех статей |

4.3.2.4 Роут /api/add/favorite/article

Обрабатывает POST-запрос на адресе /api/add/favorite/article описание приведено в таблице 15.

Таблица 15 – Описание роута /api/add/favorite/article

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | addToFavorite() | Добавление статьи в избранное |

4.3.2.5 Роут /api/get/favorite/articles

Обрабатывает GET-запрос на адресе /api/get/favorite/articles описание приведено в таблице 16.

Таблица 16 – Описание роута /api/get/favorite/articles

|  |  |  |
| --- | --- | --- |
| № | Метод | Описание |
| 1 | getFavoriteArticles() | Получение избранных статей |

4.3.3 Клиентская часть

4.3.3.1 Экран AuthorizationScreen

Реализует экран авторизации пользователя в системе. Описание представленно в таблице 17.

Таблица 17 – Описание экрана AuthorizationScreen

|  |  |  |
| --- | --- | --- |
| № | Название элемента | Описание |
| Поля | | |
| 1 | formData | Данные формы (логин и пароль) |
| 2 | error | Ошибки формы |
| Методы | | |
| 1 | handleLogin() | Отправляет данные формы на сервер |

4.3.3.2 Экран RegistrationScreen

Реализует экран регистрации пользователя в системе. Описание приведено в таблице 18.

Таблица 18 – Описание экрана RegistrationScreen

|  |  |  |
| --- | --- | --- |
| № | Название элемента | Описание |
| Поля | | |
| 1 | formData | Данные формы (логин и пароль) |
| 2 | error | Ошибки формы |
| **Методы** | | |
| 1 | handleRegistration() | Отправляет данные формы на сервер |

4.3.3.3 Экран AddArticle

Реализует экран добавления статьи. Описание приведено в таблице 19.

Таблица 19 – Описание экрана AddArticle

|  |  |  |
| --- | --- | --- |
| № | Название элемента | Описание |
| Поля | | |
| 1 | articleData | Данные формы (заголовок, категория, норма и описание) |
| 2 | error | Ошибки формы |
| **Методы** | | |
| 1 | handleAdd() | Формирует и сохраняет статью |

4.3.3.4 Экран SearchScreen

Реализует экран поиска статей. Описание приведено в таблице 20.

Таблица 20 – Описание экрана SearchScreen

|  |  |  |
| --- | --- | --- |
| № | Название элемента | Описание |
| Поля | | |
| 1 | searchData | Данные формы (строка поиска) |
| 2 | error | Ошибки формы |
| **Методы** | | |
| 1 | handleSearch() | Выводит список статей, удовлетворяющих условию |

4.4 Особенности реализации, эксплуатации и сопровождения системы

Разработка данный системы предполагает наличие знаний и опыта в написании клиентской части, серверной части, конфигурирования веб-сервера и настройки системы контейнеризации. Эксплуатация системы осуществляется при базовом понимании каждой из частей.

Данная система развернута на удаленном сервере. Для ее конфигурации необходим доступ через протокол SSH.

Мобильное приложение имеет удобный и интуитивно понятный интерфейс, использование которого не требует квалификационных знаний. Система также может подвергаться расширению и добавлению нового функционала с помощью разработчиков.

4.5 Руководство пользователя

4.5.1 Требования к условиям эксплуатации

Для пользователя основные системные требования определяются требованиями для клиентской части.

Минимальные системные требования к техническому устройству:

* смартфон на ОС Android 8.0 или iOS 11.0;
* объем оперативной памяти 2Gb;
* свободное место на внутренней памяти устройства 200Mb;
* доступ в интернет.

4.5.2 Инициализация и настройка

Для запуска всех необходимых сервисов необходимо иметь ПО Docker. После чего необходимо выполнить команду docker compose up -d для запуска всех сервисов в фоновом режиме. Проверить работу сервисов можно при помощи команды docker ps.

4.5.3 Порядок и особенности работы

При первом запуске мобильного приложения, открывается экран авторизации с возможностью прохождения регистрации и формой ввода логина и пароля для авторизации в существующий аккаунт (рисунок 5). Для прохождения авторизации необходимо ввести в форму логин и пароль, затем нажать кнопку «Войти». При нажатии по кнопку «Регистрация», открывается экран регистрации. На нем есть форма для ввода логина и пароля пользователя системы (рисунок 6).

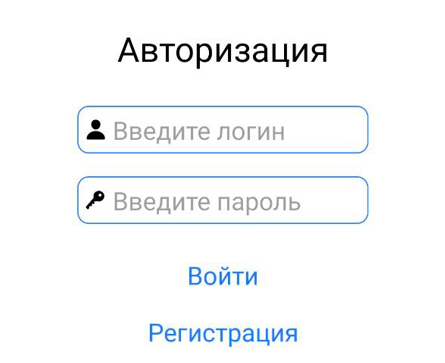


Рисунок 5 – Экран авторизации пользователя

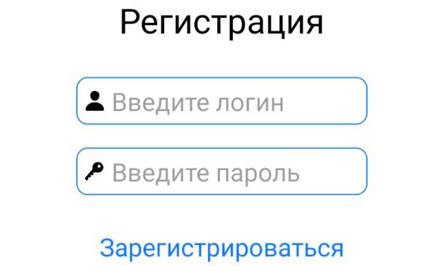


Рисунок 6 – Экран регистрации пользователя

После авторизации пользователя, открывается экран со статьями (рисунок 7). В нижней части экрана расположена панель навигации по приложению.

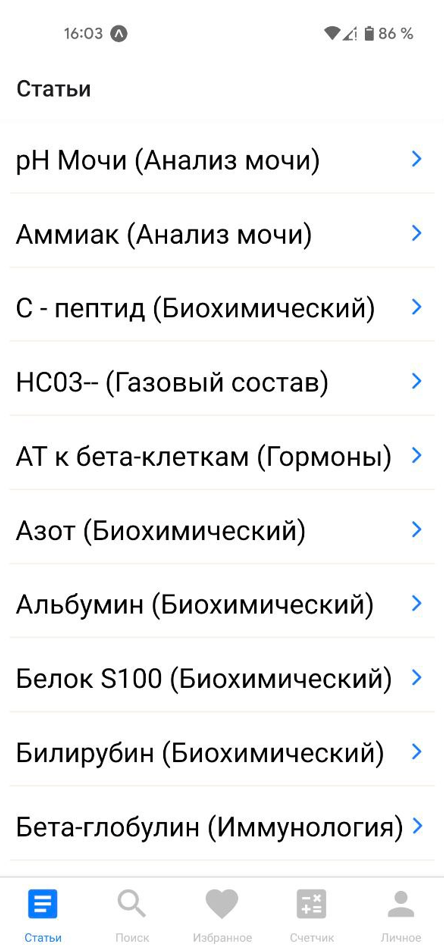


Рисунок 7 – Экран со статьями

При нажатии на интересующую пользователя статью, он перейдет на экран описания данной статьи (рисунок 8). На данной странице пользователь может ознакомиться с нормальным показателем и его описанием. Для возврата к списку статей необходимо нажать на кнопку в виде стрелки. Также существует возможность добавления статьи в «Избранное». Для этого необходимо нажать на кнопку в виде сердца. При успешном добавление статьи в избранное, пользователь получит уведомление с названием статьи и кнопка в виде сердца окрасится в красный цвет (рисунок 9). Для отмены этого действия необходимо нажать на кнопку в виде сердца ещё раз. Также будет показано уведомление об удалении статьи из «Избранного» и сердце окрасится в черный цвет (рисунок 10).

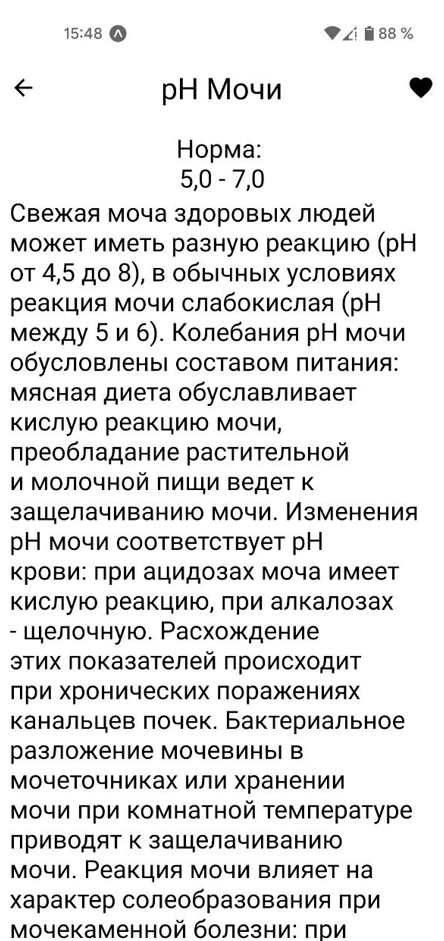


Рисунок 8 – Экран описания статьи

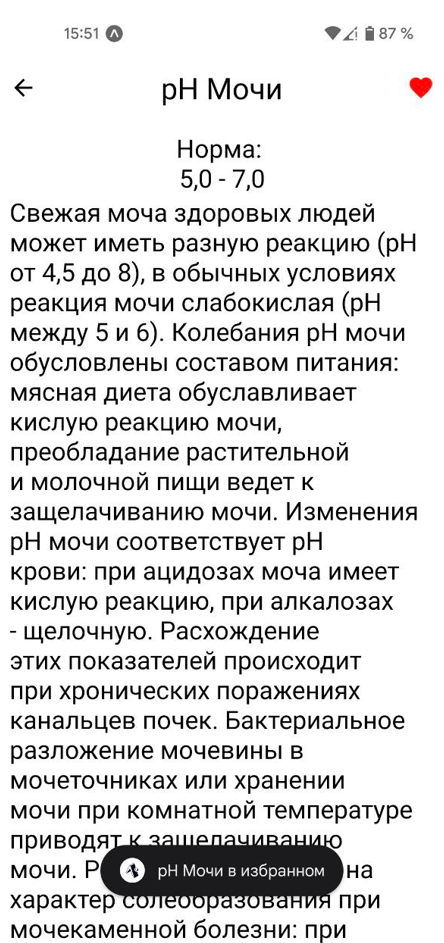


Рисунок 9 – Добавление статьи в избранное

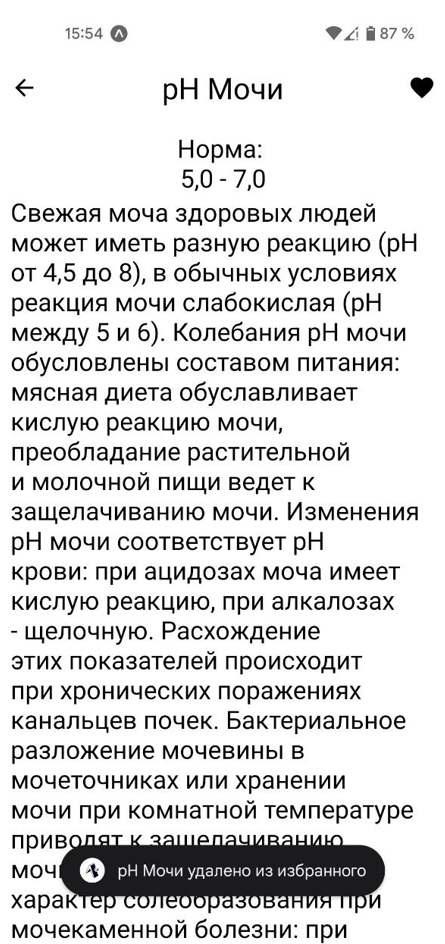


Рисунок 10 – Удаление статьи из избранного

Выбрав в панели навигации вкладку «Поиск», пользователь попадает на экран поиска статей. Здесь расположена форма ввода для поискать и кнопка «Искать» для получения результатов поиска (рисунок 11).



Рисунок 11 – Экран поиска статей

Введя данные в поле ввода и нажав на кнопку «Искать», пользователь получает список статей, удовлетворяющих условиям поиска (рисунок 12). При нажатии на название статьи, пользователь попадет на экран описания статьи.



Рисунок 12 – Результат поиска

При переходе на вкладку избранное, пользователь получает список избранных статей (рисунок 13). Нажав на названии статьи, пользователь переходит на экран описания статьи.

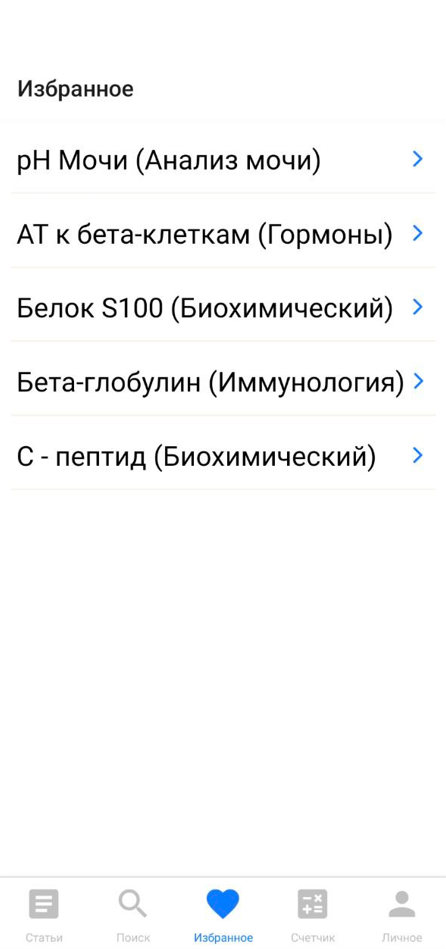


Рисунок 13 – Экран с избранными статьями

Перейдя на вкладку «Счетчик», пользователь попадает на экран лейкоцитарного счетчика (рисунок 14). Для ввода типа клеток, необходимо нажимать на кнопку с названием клетки. Так же есть возможность отмены ввода клетки, путем нажатия на стрелку. При вводе ста клеток и нажатии по кнопке «Результат», пользователь переходит на экран результата работы лейкоцитарной формулы. Здесь описан результат работы лейкоцитарного счетчика в виде таблице, где указаны клетки крови и их количество (рисунок 15).



Рисунок 14 – Экран лейкоцитарный счетчик

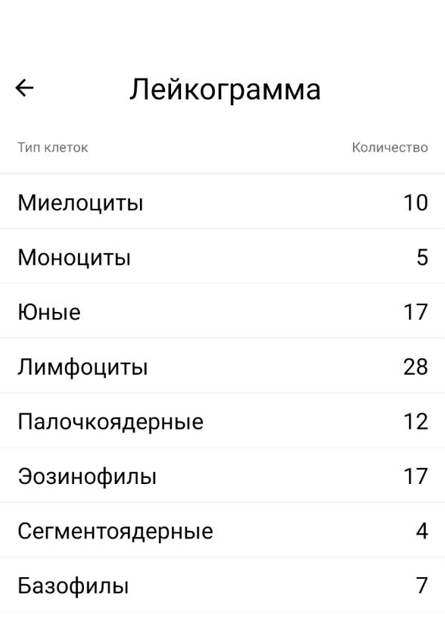


Рисунок 15 – Результат работы лейкоцитарного счетчика

Во вкладке «Личное» отображается приветственное сообщение с логином пользователя, кнопка выхода из аккаунта пользователя и дополнительные возможности, такие как локальное добавление статей в справочник (рисунок 16). При нажатии на кнопку «Выйти из учетной записи», пользователь осуществляет выход из своего аккаунта и переходит на страницу авторизации.

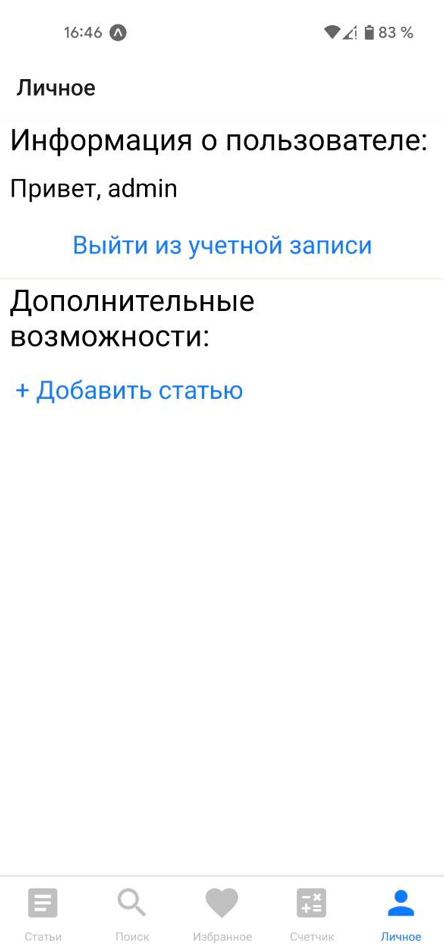


Рисунок 16 – Экран личное

Нажав на кнопку «Добавить статью», пользователь переходит на экран добавления статьи в справочник (рисунок 17). Здесь расположена форма для указания заголовка статьи, её категории, нормального значения и описание. Заполнив эти данные и нажав на кнопку «Добавить», статья попадает в основной список статей и может быть просмотрена.

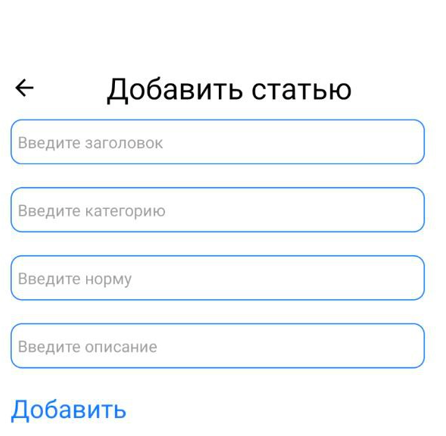


Рисунок 17 – Экран добавить статью

4.5.4 Условия и порядок тестирования

Во время эксплуатации могут возникать различные исключительные ситуации, в резульате некооректных действий пользователя или ошибочных вводаж. Мобильное приложение выдает сообщение пользователю с пояснением той или иной ситуации, и продолжает функционировать в рабочем состоянии. Примеры подоных ситуаций рассмотрены ниже.

При вводе пустых строк на экране авторизации, появится сообщение об ошибке (рисунок 18).

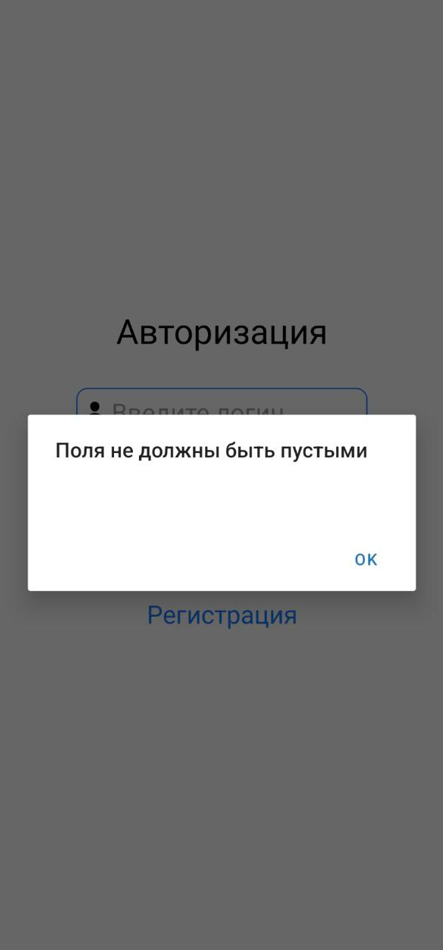


Рисунок 18 – Ввод некорректных данных при авторизации

Если пользователя не существует в базе данных мобильного приложения, появится уведомление о некорректном вводе данных пользователя (рисунок 19).

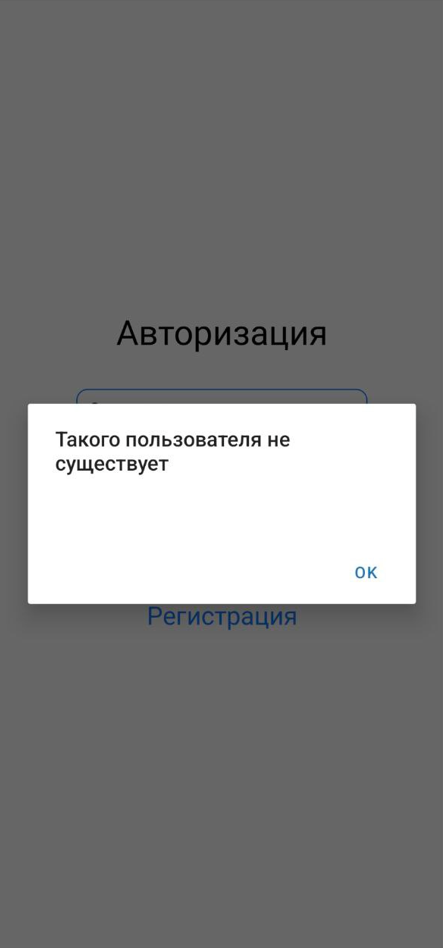


Рисунок 19 – Ошибка при авторизации

Если во время регистрации пользователь не ввел данные в форму ввода и нажал кнопку «Зарегистрироваться», он получит уведомление о некорректном вводе (рисунок 20).

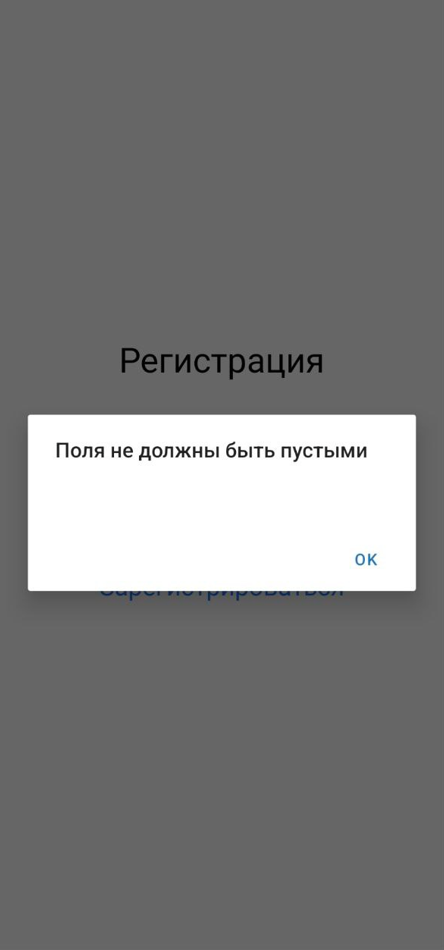


Рисунок 20 – Ввод некорретных данных при регистрации

Предотвращение получения результата работы лейкоцитарного счетчика, при неполном заполнении клеток и возникновение сообщения об этом (рисунок 21).

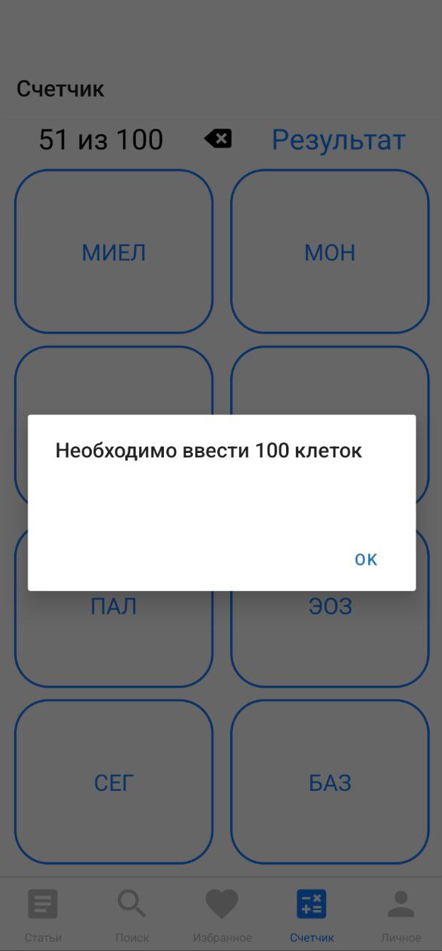


Рисунок 21 – Введено недостаточное количество клеток

При попытке добавить статью, без заполненных полей, возникает сообщение о том, что пользователь не заполнил данные (рисунок 22).

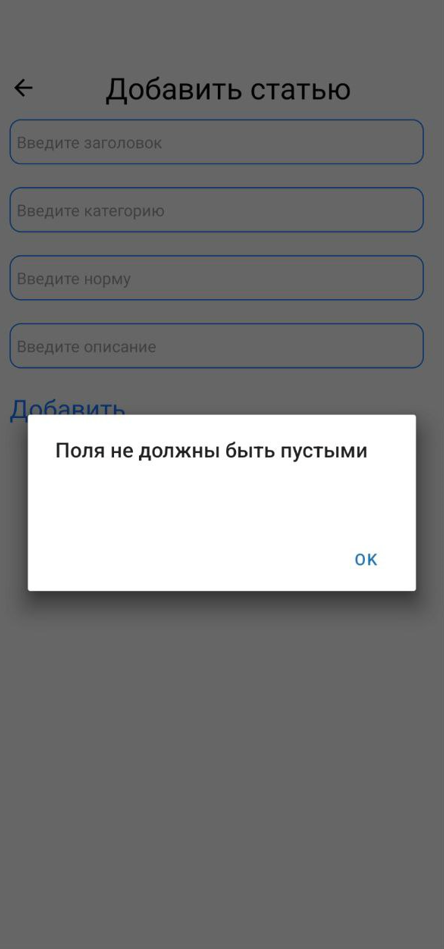


Рисунок 22 – Некорректный ввод данных статьи

**5 ТЕСТИРОВАНИЕ СИСТЕМЫ**

5.1 Условия и порядок тестирования

Объектом для тестирования является разработанный генератор РПД. В качестве проверки будем использовать модель «чёрный ящик». Данный тип проверки является ручным. Тестирование проходит в порядке снятия входных и выходных данных.

5.2 Исходные данные для контрольных примеров

5.2.1 Авторизация пользователя

Пользователь вводит свой логин и пароль для авторизации и нажимает на кнопку «Войти».

Возможные ситуации:

* пользователь авторизовался и перешел на экран статьи;
* пользователь не ввел логин или пароль;
* пользователь ввел неверный логин или пароль.

5.2.2 Регистрация пользователя

Пользователь вводит логин и пароль для создания аккаунта и нажимает на кнопку «Регистрация».

Возможные ситуации:

* пользователь зарегистрировался и перешел на экран авторизации;
* пользователь не ввел логин или пароль.

5.2.3 Добавление статьи в избранное

Пользователь вводит заголовок, категорию, нормальное значение и описание стать и нажимает на кнопку «Добавить».

Возможные ситуации:

* пользователь добавил статью и перешел на экран статей;
* пользователь не ввел заголовок, категорию, норму или описание.

5.3 Результаты тестирования

По результатам тестирования системы можно сделать вывод, что основной функионал приложения работает корректно. Вомзожные ошибки обработаны, проработаны возможные сценарии попыток вывода приложения из строя. Работа мобильного приложения и его реакция на возникающие ситуации объяснена в п. 4.5.4 исключительные ситуации и их обработка.

**6 ЭКОНОМИЧЕСКИЙ РАЗДЕЛ**

**6.1 Определение трудоемкости разработки системы**

В первую очередь необходимо определить ожидаемую продолжительность работы q. Она рассчитывается как математическое ожидание для равномерного распределения:

,

где – минимальная продолжительность работы;

– максимальная продолжительность работы.

Перечень работ и трудоемкость их выполнения приведен в таблице 21.

Таблица 21 – Трудоемкость работ по разработке интернет-портала

|  |  |
| --- | --- |
| **Наименование рабочего процесса** | **Трудоемкость (чел./ч)** |
| Формирование требований к системе |  |
| Разработка технического задания |  |
| Моделирование структуры и функционала |  |
| Реализация программного решения |  |
| Отладка и тестирование системы |  |
| Написание рабочей документации |  |
| Сдача проекта |  |
| **Итого** | **368** |

Самым трудоемки процессом разработке будет процесс реализации программного решения.

**6.2 Затраты на разработку системы**

Затраты на разработку системы – это сумма всех затрат на производство и реализацию продукта, составленная по экономическим элементам расходов.

Расчёт затрат формируется через составление общей сметы, включающей в себя следующие статьи:

* затраты на материальные ресурсы;
* затраты на электроэнергию;
* затраты на оплату труда;
* затраты на обязательные взносы;
* амортизация основных фондов.

6.2.1 Затраты на материальные ресурсы

Затраты на материальные ресурсы проекта представлены в таблице 22.

Таблица 22 – Затраты на материальные ресурсы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Наименование** | **Количество израсходованного материала** | **Цена за единицу, руб.** | **Сумма, руб.** |
| 1 | Ноутбук | 1шт | 40 000 | 40 000 |
| 2 | Смартфон | 1шт | 28 000 | 28 000 |
| **Итого затраты на материальные ресурсы** | | | | 73 000 |

**6.2.2 Затраты на электроэнергию**

Общая сумма затрат на электроэнергию рассчитывается по формуле:

где i –вид электрооборудования;

n – количество электрооборудования;

– паспортная мощность i-го электрооборудования, кВт;

– коэффициент использования мощности i-го электрооборудования (принимается );

– время работы i-го оборудования за весь период разработки, ч;

Ц – цена электроэнергии, .

Результаты расчётов представлены в таблице 23.

Таблица 23 – Затраты на электроэнергию

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Наименование оборудования** |  |  |  | **Ц, руб/кВт·ч** | **Сумма, руб.** |
| Ноутбук | 0,07 | 0,8 | 368 | 2,82 | 58,2 |
| Искусственное освещение | 0,05 | 201 | 22,7 |
| Смартфон | 0,02 | 68 | 3,1 |
| **Итого** | | | | | 84 |

**6.2.3 Затраты на оплату труда**

Оплату труда разработчика планируется выплачивать в повременной форме. В таком случае она будет зависеть от количества отработанных часов и от квалификации разработчика. При анализе будет считать, что разработчик имеет среднюю квалификацию.

Сумма основных затрат на оплату труда определяется по формуле:

Где – часовая ставка i-го работника, руб.;

– время на разработку системы, ч;

i – порядковый номер работника;

n – количество работников.

Результаты расчета приведены в таблице 24.

Таблица 24 – Затраты на оплату труда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Категория работника** | **Трудоемкость разработки, чел./ч** | **Часовая ставка, руб./ч** | **Сумма, руб.** |
| 1 | Fullstack-разработчик | 368 | 330 | 121 400 |
| **Итого затраты на основную оплату труда** | | | | 121 400 |

### **6.2.4 Затраты на обязательные взносы**

Отчисления на социальные нужды являются обязательными отчислениями по нормам, установленным законодательством государственного социального страхования нормам в Фонд социального страхования РФ, Пенсионный фонд РФ, фонды обязательного медицинского страхования от затрат на оплату труда работников, включаемых в себестоимость продукции, по элементу «Затраты на оплату труда» (кроме тех видов оплат, на которые страховые взносы не начисляются).

Результаты расчета обязательных взносов приведены в таблицы 25.

Таблица 25 – Отчисления на обязательные взносы

|  |  |
| --- | --- |
| **Вид** | **Отчисления, %** |
| Пенсия:   * Страховое * Накопительное | 16 |
| 6 |
| Социальные выплаты | 2,9 |
| Медицинское страхование | 5,1 |
| Взносы на страхование от несчастных случаев | 0,2 |
| Итого обязательных взносов | 30,2 |

Использую заработную плату, которая была рассчитана ранее, вычислим размер отчислений в социальные фонды по формуле:

,

где ЕСН – это размер единого социального налога в рублях;

ЗП – итоговая заработная плата сотрудника в рублях.

### **6.2.5 Затраты на амортизационные отчисления**

Данный вид затрат представляет собой средства в виде денег, направляемых на ремонт или строительство, изготовление новых основных средств. Сумма таких затрат включается в издержки производства продукта и учитывается при расчете его стоимости.

При расчёте амортизационных отчислений используется линейный способ. Расходы на амортизацию рассчитываются по формуле:

где – стоимость i-го оборудования, руб.;

– годовая норма амортизации i-го оборудования, % доля;

– время работы i-го оборудования за весь период разработки, ч;

– эффективный фонд времени работы i-го оборудования за год, ч/год (40-часовая рабочая неделя на 52 недели/год, кроме праздников);

i – вид оборудования;

n – количество оборудования.

6.3 Себестоимость проекта

Итоговый расчет себестоимости проекта приведен в таблице 26.

Таблица 26 – Себестоимость проекта

|  |  |  |
| --- | --- | --- |
| **№** | **Статьи затрат** | **Сумма, руб.** |
| 1 | Затраты на материальные ресурсы | 73 500 |
| 2 | Затраты на электроэнергию | 84 |
| 3 | Затраты на оплату труда | 121 400 |
| 4 | Затраты на обязательные взносы | 36 663 |
| 5 | Амортизация основных фондов | 2385 |
| Итого по смете | | 234 032 |

6.4 Определение экономической эффективности проекта

После расчетов совокупности всех затрат необходимо рассчитать прибыль, которую получит предприятие от разработки данной системы.

Экономическая эффективность – результативность экономической системы, выражающаяся в отношении полезных конечных результатов ей функционирования к затраченным ресурсам. Она определяется как разность между годовой экономией (или годовым приростом) и нормативной прибылью

Повышение производительности труда высчитывается по формуле:

где – временные затраты до автоматизации, мин.;

– экономия времени (разница затрат после и до автоматизации), мин.;

Прирост производительности труда представлен в таблице 27.

Таблица 27 – Прирост производительности труда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид работы | До автоматизации Ti, мин | После автоматизации (Ti-ΔTi), мин | Экономия ΔTi, мин | Повышение производительности труда Pi, % |
| Время доступа к справочным материалам | 10 | 1 | 9 | 900 |

На данный момент средняя ставка сотрудника лаборатории в России составляет 190руб/час. В России около 20 тысяч сотрудников лаборатории.

Среднегодовая заработная плата составит:

Годовой прирост высчитывается по формуле:

где Зг – среднегодовая заработная плата пользователя, руб.;

– повышение производительности труда, %.

П = 47 025 \* 9 = 423 225 (руб).

Годовой экономический эффект Э (руб), рассчитывается по формуле:

где П – годовая экономия (или годовой прирост), руб.;

К – единовременные затраты, руб;

– нормативный коэффициент эффективности (сложившийся в IT-отрасли уровень рентабельности = 0,15).

Э = 423 225 – 234 032 \* 0,15 = 388 120,2 руб.

Как видно из расчётов, экономический эффект намного превышает затраты на создание системы (Э > 0).

Расчётный коэффициент экономической эффективности (абсолютная экономическая эффективность) составит: 1.8.

Ep>Eн это значит, что капитальные затраты можно считать целесообразными.

Срок окупаемости проекта:

6.5 Выводы по технико-экономическому анализу и обоснованию проекта разработки

Произведённый экономический анализ эффективности создания и эксплуатации программного продукта доказывает целесообразность его использования.

Согласно рассчитанным данным о смете затрат была получена себестоимость проекта – 234 032 руб. Также в ходе проведения разработки проекта вычисленная чистая прибыль (экономический эффект) составила 388 120,2 pуб.

Разрабатываемая система окупается за 6,6 месяца. Следовательно, организации, принявшие решение применить данную ИС, в будущем сократят затраты на автоматизируемые системой бизнес-процессы.

**ЗАКЛЮЧЕНИЕ**

Результатом выполнения выпускной квалификационной работы является удобное в применении, кроссплатформенное мобильное приложение. Оно позволяет получать справочную информацию в удобном виде и создавать пользовательские статьи при необходиости. Приложение обладает системами поиска и избранного, улучшающие пользовательский опыт работы с приложением. У данной системы нет аналогов, так как аналогичные справочники больше предназначенны для пациентов.

Дальнейшее развитие системы предполагает создание дополнительного функционала для сотрудников лабораторий, такие как калькуляторы приготовления и разбавления растворов, созданий статей для всех пользователей системы, и расширенный ЛС.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. ГОСТ 19.401-78. Единая система программной документации. Текст программы. Требования к содержанию и оформлению. – Введ. 1980-01-01. –М.: Стандартинформ, 2010. – 2 с.
2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Стандартинформ, 2010. – 8 c.
3. Оценка трудоемкости разработки программного продукта: метод указания / сост. Шанченко Н.И. – Ульяновск: УлГТУ, 2015. – 40с.
4. Родионов, В. В. Выполнение выпускной квалификационной работы бакалавра: учебно-методическое пособие для студентов направления 09.03.02 «Информационные системы и технологии» / В. В. Родионов. – Ульяновск: УлГТУ, 2017. – 74 с.
5. MySQL Documentation [Электронный ресурс]. – Режим доступа: https://dev.mysql.com/doc/,свободный. – Загл. с экрана (дата обращения: 01.06.2022).
6. React native Documentation [Электронный ресурс]. – Режим доступа: <https://reactnative.dev/docs/getting-started.html>, свободный. – Загл. с экрана (дата обращения: 07.06.2022).
7. Symfony Documentation [Электронный ресурс]. – Режим доступа: <https://symfony.com/doc/current/index.html>, свободный. – Загл. с экрана (дата обращения 10.06.2022).

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

Файл client/src/components/Header.js

import React from 'react';

import { View, Text, TouchableOpacity, StyleSheet, ToastAndroid } from 'react-native';

import MaterialIcons from 'react-native-vector-icons/MaterialIcons'

import { useDispatch, useSelector } from 'react-redux';

import { addToFavourite, deleteFromFavourite } from '../store/favouriteSlice'

export default function Header({ item, backHandler }) {

    const favourites = useSelector(state => state.favourites.favouriteArticles)

    const dispatch = useDispatch();

    const existInFavourite = (favourites, currentArticle) => {

        let isExist = false;

        favourites.forEach(item => {

            if (item.id == currentArticle.id) {

                isExist = true;

            }

        })

        return isExist

    }

    const favouriteHandler = () => {

        let check = existInFavourite(favourites, item);

        if (!check) {

            dispatch(addToFavourite(item));

            ToastAndroid.show(item.title + " в избранном", ToastAndroid.SHORT);

        } else {

            dispatch(deleteFromFavourite(item.id));

            ToastAndroid.show(item.title + " удалено из избранного", ToastAndroid.SHORT);

        }

    }

    const check = existInFavourite(favourites, item);

    return (

        <View style={styles.headerContainer}>

            <TouchableOpacity style={styles.backButtonContainer} onPress={backHandler}>

                <MaterialIcons name="arrow-back" size={24} color='black' />

            </TouchableOpacity>

            <View style={styles.titleContainer}>

                <Text style={styles.titleText}>{item.title}</Text>

            </View>

            <TouchableOpacity style={styles.favContainer} onPress={favouriteHandler} >

                <MaterialIcons name="favorite" size={24} color={check ? "red" : 'black'} />

            </TouchableOpacity>

        </View>

    )

}

const styles = StyleSheet.create({

    headerContainer: {

        backgroundColor: 'white',

        flexDirection: 'row',

        alignItems: 'center',

        paddingTop: 60,

        paddingHorizontal: 10,

        paddingBottom: 10,

        justifyContent: 'space-between',

    },

    backButtonContainer: {

    },

    titleContainer: {

    },

    titleText: {

        fontSize: 26

    },

    favContainer: {

    }

})

Файл client/src/components/List.js

import React from 'react'

import { ScrollView } from 'react-native'

import TouchablaItem from './TouchableItem'

export default function List({ data, navigation }) {

    return (

        <ScrollView>

            {data.map((item) => {

                return <TouchablaItem title={item.title + " (" + item.category + ")"} onPress={() => navigation.push('description', { item })} />

            })}

        </ScrollView>

    )

}

Файл client/src/components/TouchableItem.js

import React from 'react'

import { View, Text, TouchableOpacity, StyleSheet } from 'react-native'

import Feather from 'react-native-vector-icons/Feather'

export default function ToucheblaItem({ title, onPress }) {

    return (

        <TouchableOpacity style={styles.groupItem} onPress={onPress}>

            <Text style={styles.groupTitle}>{title}</Text>

            <View style={styles.groupIcon}>

                <Feather name='chevron-right' size={24} color='#007bff' />

            </View>

        </TouchableOpacity>

    )

}

const styles = StyleSheet.create({

    groupTitle: {

        fontSize: 24,

        marginLeft: 5

    },

    groupItem: {

        backgroundColor: 'white',

        width: '95%',

        height: 60,

        justifyContent: 'space-between',

        alignItems: 'center',

        marginLeft: 10,

        marginRight: 10,

        marginTop: 5,

        borderStyle: 'solid',

        borderBottomWidth: 1,

        borderBottomColor: '#FAF0E6',

        flexDirection: 'row'

    },

    groupIcon: {

        marginRight: 4

    }

})

Файл client/src/navigation/StackNavigator.js

import React from 'react';

import { createStackNavigator } from '@react-navigation/stack';

import TabsNavigator from './TabsNavigator'

import DescriptionScreen from '../screens/DescriptionScreen';

import ResultScreen from '../screens/ResultScreen';

import AuthorizationScreen from '../screens/AuthorizationScreen';

import RegistrationScreen from '../screens/RegistrationScreen'

import AddArticle from '../screens/AddArticle';

const Stack = createStackNavigator();

export default function StackNavigator() {

    return (

        <Stack.Navigator screenOptions={{

            headerShown: false

        }}>

            <Stack.Screen name='auth' component={AuthorizationScreen} />

            <Stack.Screen name='tabs' component={TabsNavigator} />

            <Stack.Screen name='description' component={DescriptionScreen} />

            <Stack.Screen name='result' component={ResultScreen} />

            <Stack.Screen name='register' component={RegistrationScreen} />

            <Stack.Screen name='adding' component={AddArticle} />

        </Stack.Navigator>

    )

}

Файл client/src/navigation/TabsNavigator.js

import \* as React from 'react';

import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

import MaterialIcons from 'react-native-vector-icons/MaterialIcons'

import ArticlesScreen from '../screens/ArticlesScreen';

import SearchScreen from '../screens/SearchScreen';

import FavouriteScreen from '../screens/FavouriteScreen';

import ProfileScreen from '../screens/ProfileScreen';

import CounterScreen from '../screens/CounterScreen';

const Tab = createBottomTabNavigator();

export default function TabsNavigator() {

    return (

        <Tab.Navigator screenOptions={{

            tabBarStyle: {

                backgroundColor: 'white',

                borderStyle: 'solid',

                borderTopWidth: 1,

                borderTopColor: '#DCDCDC',

                paddingBottom: 5,

                height: 65,

            },

            tabBarInactiveTintColor: '#C0C0C0',

            tabBarActiveTintColor: '#007bff',

        }}>

            <Tab.Screen name="Статьи" component={ArticlesScreen} options={{

                tabBarIcon: ({ color, size }) => {

                    return <MaterialIcons name="article" size={34} color={color} />

                }

            }} />

            <Tab.Screen name="Поиск" component={SearchScreen} options={{

                tabBarIcon: ({ color, size }) => {

                    return <MaterialIcons name="search" size={34} color={color} />

                }

            }} />

            <Tab.Screen name="Избранное" component={FavouriteScreen} options={{

                tabBarIcon: ({ color, size }) => {

                    return <MaterialIcons name="favorite" size={34} color={color} />

                }

            }} />

            <Tab.Screen name="Счетчик" component={CounterScreen} options={{

                tabBarIcon: ({ color, size }) => {

                    return <MaterialIcons name="calculate" size={34} color={color} />

                }

            }} />

            <Tab.Screen name="Личное" component={ProfileScreen} options={{

                tabBarIcon: ({ color, size }) => {

                    return <MaterialIcons name="person" size={34} color={color} />

                }

            }} />

        </Tab.Navigator>

    )

}

файл client/src/screens/AddArticle.js

import React, { useState } from 'react';

import { ScrollView, View, Text, StyleSheet, TextInput, TouchableOpacity, Alert } from 'react-native';

import MaterialIcons from 'react-native-vector-icons/MaterialIcons'

import { useDispatch } from 'react-redux';

import { addToArticles } from '../store/articlesSlice'

export default function AddArticle({ navigation }) {

    const [title, setTitle] = useState("");

    const [category, setCategory] = useState("");

    const [norm, setNorm] = useState("");

    const [description, setDescription] = useState("");

    const dispatch = useDispatch();

    const backHandler = () => {

        navigation.goBack();

    }

    const isEmpty = (string) => {

        if (string.trim() == '') {

            return true

        }

        return false;

    }

    const addArticleHandler = () => {

        let checkTitle = isEmpty(title);

        let checkCategory = isEmpty(category);

        let checkNorm = isEmpty(norm);

        let checkDesc = isEmpty(description);

        if (isEmpty(title)) {

            Alert.alert('Поля не должны быть пустыми')

        }

        if (isEmpty(category)) {

            Alert.alert('Поля не должны быть пустыми')

        }

        if (isEmpty(norm)) {

            Alert.alert('Поля не должны быть пустыми')

        }

        if (isEmpty(description)) {

            Alert.alert('Поля не должны быть пустыми')

        }

        dispatch(addToArticles({

            title,

            category,

            norm,

            description

        }))

        //Alert.alert('Статья ' + title + ' добавлена');

        //navigation.push('tabs');

    }

    return (

        <View style={styles.mainContainer}>

            <View style={styles.headerContainer}>

                <TouchableOpacity onPress={backHandler}>

                    <MaterialIcons name="arrow-back" size={24} color='black' />

                </TouchableOpacity>

                <Text style={styles.headerTitle}>Добавить статью</Text>

            </View>

            <ScrollView style={styles.addContainer}>

                <View style={styles.rowContainer}>

                    <TextInput placeholder='Введите заголовок' style={styles.input} value={title} onChangeText={newTitle => setTitle(newTitle)} />

                </View>

                <View style={styles.rowContainer}>

                    <TextInput placeholder='Введите категорию' value={category} onChangeText={newCategory => setCategory(newCategory)} />

                </View>

                <View style={styles.rowContainer}>

                    <TextInput placeholder='Введите норму' value={norm} onChangeText={newNorm => setNorm(newNorm)} />

                </View>

                <View style={styles.rowContainer}>

                    <TextInput placeholder='Введите описание' multiline={true} value={description} onChangeText={newDescription => setDescription(newDescription)} />

                </View>

                <TouchableOpacity style={styles.btn} onPress={addArticleHandler}>

                    <Text style={styles.btnText}>Добавить</Text>

                </TouchableOpacity>

            </ScrollView>

        </View>

    )

}

const styles = StyleSheet.create({

    headerContainer: {

        backgroundColor: 'white',

        flexDirection: 'row',

        alignItems: 'center',

        paddingTop: 60,

        paddingHorizontal: 10,

        paddingBottom: 10,

    },

    headerTitle: {

        fontSize: 26,

        marginLeft: 60,

        textAlign: 'center'

    },

    mainContainer: {

        backgroundColor: 'white',

        flex: 1,

    },

    rowContainer: {

        flexDirection: 'row',

        width: '95%',

        alignItems: 'center',

        backgroundColor: 'white',

        borderRadius: 10,

        padding: 5,

        marginBottom: 20,

        borderStyle: 'solid',

        borderWidth: 1,

        borderColor: '#007bff',

    },

    addContainer: {

        marginLeft: 10,

    },

    btn: {

        backgroundColor: 'white',

        marginBottom: 20

    },

    btnText: {

        fontSize: 22,

        color: "#007bff"

    }

})

файл client/src/screens/ArticlesScreen.js

import React from 'react'

import { ScrollView, StyleSheet } from 'react-native'

import List from '../components/List';

//new

import { useSelector } from 'react-redux'

export default function ArticlesScreen({ navigation }) {

    const data = useSelector(state => state.articles.articles)

    return (

        <ScrollView style={styles.articlesContainer}>

            <List data={data} navigation={navigation} />

        </ScrollView>

    )

}

const styles = StyleSheet.create({

    articlesContainer: {

        backgroundColor: 'white'

    }

})

файл client/src/screens/AuthorizationScreen.js

import React, { useState } from 'react';

import { View, Text, TextInput, StyleSheet, TouchableOpacity, Alert } from 'react-native';

import Ionicons from 'react-native-vector-icons/Ionicons'

import { useDispatch, useSelector } from 'react-redux';

import { setCurrentUser } from '../store/usersSlice'

export default function AuthorizationScreen({ navigation }) {

    const [login, setLogin] = useState("");

    const [password, setPassword] = useState("");

    const users = useSelector(state => state.users.users);

    const curUs = useSelector(state => state.users.currentUser);

    const dispatch = useDispatch();

    const correctAuth = () => {

        let isCorrect = false;

        let isEmpLogin = false;

        let isEmpPassword = false;

        if (login.trim() == "" && password.trim() == "") {

            Alert.alert('Поля не должны быть пустыми ')

        }

        users.forEach(user => {

            if ((user.login == login) && (user.password == password)) {

                dispatch(setCurrentUser(user));

                console.log(curUs);

                isCorrect = true;

            }

        })

        if (isCorrect) {

            navigation.push('tabs')

        } else {

            Alert.alert('Такого пользователя не существует')

        }

    }

    return (

        <View style={styles.authContainer}>

            <Text style={styles.title}>Авторизация</Text>

            <View style={styles.inputContainer}>

                <Ionicons name="person" size={20} color="black" />

                <TextInput placeholder='Введите логин' style={styles.searchInput} value={login} onChangeText={newLogin => setLogin(newLogin)} />

            </View>

            <View style={styles.inputContainer}>

                <Ionicons name="key" size={20} color="black" />

                <TextInput placeholder='Введите пароль' style={styles.searchInput} value={password} onChangeText={newPassword => setPassword(newPassword)} />

            </View>

            <TouchableOpacity style={styles.btn} onPress={correctAuth}>

                <Text style={styles.btnText}>Войти</Text>

            </TouchableOpacity>

            <TouchableOpacity style={styles.btn} onPress={() => navigation.push('register')} >

                <Text style={styles.btnText}>Регистрация</Text>

            </TouchableOpacity>

        </View>

    )

}

const styles = StyleSheet.create({

    authContainer: {

        flex: 1,

        alignItems: 'center',

        justifyContent: 'center',

        backgroundColor: 'white'

    },

    loginContainer: {

        flexDirection: 'row'

    },

    input: {

        backgroundColor: 'yellow',

        width: '100%',

        padding: 10,

        margin: 10,

        fontSize: 22,

        borderRadius: 10

    },

    test: {

        flexDirection: 'row'

    },

    inputContainer: {

        flexDirection: 'row',

        width: '65%',

        alignItems: 'center',

        backgroundColor: 'white',

        borderRadius: 10,

        padding: 5,

        marginBottom: 20,

        borderStyle: 'solid',

        borderWidth: 1,

        borderColor: '#007bff'

    },

    searchInput: {

        marginLeft: 5,

        fontSize: 22,

        width: '100%'

    },

    title: {

        fontSize: 30,

        marginBottom: 30

    },

    btn: {

        backgroundColor: 'white',

        margin: 10

    },

    btnText: {

        fontSize: 22,

        color: "#007bff"

    }

})

файл client/scr/screens/CounterScreen.js

import React from 'react';

import { ScrollView, Text, StyleSheet, View, TouchableOpacity, Alert } from 'react-native';

import { useDispatch, useSelector } from 'react-redux';

import { increment, decrement, addToCells, deleteFromCells, setCountNull } from '../store/counterSlice'

import Entypo from 'react-native-vector-icons/Entypo'

export default function CounterScreen({ navigation }) {

    const cells = useSelector(state => state.counter.cells);

    const count = useSelector(state => state.counter.count);

    const dispatch = useDispatch();

    const getResult = () => {

        if (cells.length == 100) {

            navigation.push('result');

        } else {

            Alert.alert("Необходимо ввести 100 клеток")

        }

    }

    const mielBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "МИЕЛ";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const monBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "МОН";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const ynBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "ЮН";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const limBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "ЛИМ";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const palBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "ПАЛ";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const eosBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "ЭОЗ";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const segBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "СЕГ";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const basBtnHandler = () => {

        if (cells.length <= 99) {

            const cell = "БАЗ";

            dispatch(increment());

            dispatch(addToCells(cell));

        }

        getResult();

    }

    const cancelStep = () => {

        if (cells.length > 0) {

            dispatch(deleteFromCells());

            dispatch(decrement());

        }

    }

    return (

        <ScrollView style={styles.container}>

            <View style={styles.countHeader}>

                <Text style={styles.countText}>{count} из 100</Text>

                <TouchableOpacity onPress={cancelStep}>

                    <Entypo name="erase" size={24} color="black" />

                </TouchableOpacity>

                <TouchableOpacity style={styles.countBtnContainer} onPress={getResult}>

                    <Text style={styles.countBtn}>Результат</Text>

                </TouchableOpacity>

            </View>

            <View style={styles.countBody}>

                <View style={styles.row}>

                    <TouchableOpacity style={styles.countElement} onPress={mielBtnHandler}>

                        <Text style={styles.textElement}>МИЕЛ</Text>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.countElement} onPress={monBtnHandler}>

                        <Text style={styles.textElement}>МОН</Text>

                    </TouchableOpacity>

                </View>

                <View style={styles.row}>

                    <TouchableOpacity style={styles.countElement} onPress={ynBtnHandler}>

                        <Text style={styles.textElement}>ЮН</Text>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.countElement} onPress={limBtnHandler}>

                        <Text style={styles.textElement}>ЛИМ</Text>

                    </TouchableOpacity>

                </View>

                <View style={styles.row}>

                    <TouchableOpacity style={styles.countElement} onPress={palBtnHandler}>

                        <Text style={styles.textElement}>ПАЛ</Text>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.countElement} onPress={eosBtnHandler}>

                        <Text style={styles.textElement}>ЭОЗ</Text>

                    </TouchableOpacity>

                </View>

                <View style={styles.row}>

                    <TouchableOpacity style={styles.countElement} onPress={segBtnHandler}>

                        <Text style={styles.textElement}>СЕГ</Text>

                    </TouchableOpacity>

                    <TouchableOpacity style={styles.countElement} onPress={basBtnHandler}>

                        <Text style={styles.textElement}>БАЗ</Text>

                    </TouchableOpacity>

                </View>

            </View>

        </ScrollView>

    )

}

const styles = StyleSheet.create({

    countHeader: {

        flexDirection: 'row',

        backgroundColor: 'white',

        justifyContent: 'space-evenly',

        alignItems: 'center',

        paddingBottom: 10

    },

    countText: {

        fontSize: 25

    },

    countBtnContainer: {

        backgroundColor: 'white'

    },

    countBtn: {

        fontSize: 25,

        color: "#007bff"

    },

    countBody: {

        flexDirection: "column",

    },

    countElement: {

        paddingHorizontal: 20,

        paddingVertical: 20,

        borderStyle: 'solid',

        borderWidth: 2,

        borderRadius: 30,

        borderColor: "#007bff",

        backgroundColor: 'white',

        width: 175,

        height: 145,

        alignItems: 'center',

        justifyContent: 'center'

    },

    row: {

        flexDirection: 'row',

        justifyContent: 'space-evenly',

        marginBottom: 10

    },

    container: {

        backgroundColor: 'white'

    },

    textElement: {

        fontSize: 20,

        color: "#007bff"

    }

})

файл client/scr/screens/DescriptionScreen.js

import React from 'react';

import { ScrollView, Text, View, StyleSheet } from 'react-native';

//new

import Header from '../components/Header';

export default function DescriptionScreen({ route, navigation }) {

    const { item } = route.params;

    const backHandler = () => {

        navigation.goBack()

    }

    return (

        <View style={styles.mainContainer}>

            <Header item={item} backHandler={backHandler} />

            <ScrollView style={styles.descContainer}>

                <View style={styles.normContainer}>

                    <Text style={styles.normText}>Норма: {'\n'}{item.norm}</Text>

                </View>

                <Text style={styles.descr}>{item.description}</Text>

            </ScrollView>

        </View>

    )

}

const styles = StyleSheet.create({

    normContainer: {

    },

    descContainer: {

        backgroundColor: 'white',

        padding: 10

    },

    normText: {

        fontSize: 22,

        textAlign: 'center'

    },

    mainContainer: {

        backgroundColor: 'white',

        flex: 1

    },

    descr: {

        fontSize: 22,

    }

})

файл client/scr/screens/FavoriteScreen.js

import React from 'react'

import { ScrollView, StyleSheet } from 'react-native'

import List from '../components/List'

//new

import { useSelector } from 'react-redux'

export default function FavouriteScreen({ navigation }) {

    const favourites = useSelector(state => state.favourites.favouriteArticles)

    return (

        <ScrollView style={styles.favContainer}>

            <List data={favourites} navigation={navigation} />

        </ScrollView>

    )

}

const styles = StyleSheet.create({

    favContainer: {

        backgroundColor: 'white'

    }

})

файл client/scr/screens/ProfileScreen.js

import React from 'react'

import { View, Text, StyleSheet, Pressable, TouchableOpacity } from 'react-native'

import { useSelector } from 'react-redux';

export default function ProfileScreen({ navigation }) {

    const currentUser = useSelector(state => state.users.currentUser);

    const btnHandler = () => {

        navigation.push('auth');

    }

    const addHandler = () => {

        navigation.push('adding');

    }

    return (

        <View style={styles.container}>

            <View style={styles.infoContainer}>

                <Text style={styles.infoHeader}>Информация о пользователе:</Text>

                <View style={styles.infoRow}>

                    <Text style={styles.infoText}>Привет, </Text>

                    <Text style={styles.infoText}>{currentUser[0].login ? currentUser[0].login : 'undefined'}</Text>

                </View>

                {/\* <View style={styles.infoRow}>

                    <Text style={styles.infoText}>Ваша роль: </Text>

                    <Text style={styles.infoText}>{currentUser[0].isAdmin ? 'Сотрудник лаборатории' : 'Пользователь'}</Text>

    </View> \*/ }

                <View style={styles.btnArea}>

                    <Pressable style={styles.btnContainer} onPress={btnHandler}>

                        <Text style={styles.btnText}>Выйти из учетной записи</Text>

                    </Pressable>

                </View>

            </View>

            <View>

                <Text style={styles.infoHeader}>Дополнительные возможности:</Text>

                <View>

                    <Pressable style={styles.btnContainer} onPress={addHandler}>

                        <Text style={styles.btnTextt}>+ Добавить статью</Text>

                    </Pressable>

                </View>

            </View>

        </View>

    )

}

const styles = StyleSheet.create({

    infoContainer: {

        flexDirection: 'column',

        backgroundColor: 'white',

        borderStyle: 'solid',

        borderBottomWidth: 1,

        borderBottomColor: '#FAF0E6'

    },

    infoHeader: {

        fontSize: 26,

        marginBottom: 10,

        marginLeft: 10

    },

    infoRow: {

        flexDirection: 'row',

        marginLeft: 10,

        marginBottom: 5

    },

    infoText: {

        fontSize: 22

    },

    btnText: {

        fontSize: 22,

        color: "#007bff",

    },

    btnTextt: {

        fontSize: 22,

        color: "#007bff",

        marginLeft: 10

    },

    btnContainer: {

        margin: 5

    },

    btnArea: {

        justifyContent: 'center',

        alignItems: 'center',

        backgroundColor: 'white',

        margin: 10

    },

    container: {

        flex: 1,

        backgroundColor: 'white'

    },

})

файл client/scr/screens/RegistrationScreen.js

import React, { useState } from 'react';

import { View, Text, TextInput, StyleSheet, Button, TouchableOpacity, Alert } from 'react-native';

import Ionicons from 'react-native-vector-icons/Ionicons'

import { useDispatch, useSelector } from 'react-redux';

import { createUser } from '../store/usersSlice'

export default function RegistrationScreen({ navigation }) {

    const [login, setLogin] = useState("");

    const [password, setPassword] = useState("");

    const users = useSelector(state => state.users.users);

    const dispatch = useDispatch();

    const registration = () => {

        if (login != "" && password != "") {

            const currentUser = {

                id: users[users.length - 1].id + 1,

                login: login,

                password: password,

                isAdmin: false

            }

            dispatch(createUser(currentUser));

            console.log(users);

            navigation.push('auth');

        }

        Alert.alert("Поля не должны быть пустыми");

    }

    return (

        <View style={styles.authContainer}>

            <Text style={styles.title}>Регистрация</Text>

            <View style={styles.inputContainer}>

                <Ionicons name="person" size={20} color="black" />

                <TextInput placeholder='Введите логин' style={styles.searchInput} value={login} onChangeText={newLogin => setLogin(newLogin)} />

            </View>

            <View style={styles.inputContainer}>

                <Ionicons name="key" size={20} color="black" />

                <TextInput placeholder='Введите пароль' style={styles.searchInput} value={password} onChangeText={newPassword => setPassword(newPassword)} />

            </View>

            <TouchableOpacity style={styles.btn} onPress={registration}>

                <Text style={styles.btnText}>Зарегистрироваться</Text>

            </TouchableOpacity>

        </View>

    )

}

const styles = StyleSheet.create({

    authContainer: {

        flex: 1,

        alignItems: 'center',

        justifyContent: 'center',

        backgroundColor: 'white'

    },

    loginContainer: {

        flexDirection: 'row'

    },

    input: {

        backgroundColor: 'yellow',

        width: '100%',

        padding: 10,

        margin: 10,

        fontSize: 22,

        borderRadius: 10

    },

    test: {

        flexDirection: 'row'

    },

    inputContainer: {

        flexDirection: 'row',

        width: '65%',

        alignItems: 'center',

        backgroundColor: 'white',

        borderRadius: 10,

        padding: 5,

        marginBottom: 20,

        borderStyle: 'solid',

        borderWidth: 1,

        borderColor: '#007bff'

    },

    searchInput: {

        marginLeft: 5,

        fontSize: 22

    },

    title: {

        fontSize: 30,

        marginBottom: 30

    },

    btn: {

        backgroundColor: 'white',

        margin: 10

    },

    btnText: {

        fontSize: 22,

        color: "#007bff"

    }

})

файл client/scr/screens/ResultScreen.js

import React from 'react'

import { ScrollView, StyleSheet, Text, Button, View, TouchableOpacity } from 'react-native'

import { useDispatch, useSelector } from 'react-redux';

import MaterialIcons from 'react-native-vector-icons/MaterialIcons'

import { DataTable } from 'react-native-paper';

export default function ResultScreen({ navigation }) {

    const cells = useSelector(state => state.counter.cells);

    const resultArray = cells;

    const mie = resultArray.filter(item => item == "МИЕЛ");

    const mon = resultArray.filter(item => item == "МОН");

    const yn = resultArray.filter(item => item == "ЮН");

    const lim = resultArray.filter(item => item == "ЛИМ");

    const pal = resultArray.filter(item => item == "ПАЛ");

    const eos = resultArray.filter(item => item == "ЭОЗ");

    const seg = resultArray.filter(item => item == "СЕГ");

    const bas = resultArray.filter(item => item == "БАЗ");

    const backHandler = () => {

        navigation.goBack();

    }

    return (

        <View style={styles.mainContainer}>

            <View style={styles.headerContainer}>

                <TouchableOpacity onPress={backHandler}>

                    <MaterialIcons name="arrow-back" size={24} color='black' />

                </TouchableOpacity>

                <Text style={styles.headerTitle}>Лейкограмма</Text>

            </View>

            <ScrollView style={styles.resultContainer}>

                <DataTable>

                    <DataTable.Header style={styles.dataHeader}>

                        <DataTable.Title>Тип клеток</DataTable.Title>

                        <DataTable.Title numeric>Количество</DataTable.Title>

                    </DataTable.Header>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Миелоциты</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{mie.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Моноциты</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{mon.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Юные</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{yn.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Лимфоциты</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{lim.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Палочкоядерные</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{pal.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Эозинофилы</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{eos.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Сегментоядерные</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{seg.length}</DataTable.Cell>

                    </DataTable.Row>

                    <DataTable.Row>

                        <DataTable.Cell textStyle={styles.dataText}>Базофилы</DataTable.Cell>

                        <DataTable.Cell textStyle={styles.dataText} numeric>{bas.length}</DataTable.Cell>

                    </DataTable.Row>

                </DataTable>

            </ScrollView>

        </View>

    )

}

const styles = StyleSheet.create({

    headerContainer: {

        backgroundColor: 'white',

        flexDirection: 'row',

        alignItems: 'center',

        paddingTop: 60,

        paddingHorizontal: 10,

        paddingBottom: 10,

    },

    headerTitle: {

        fontSize: 26,

        marginLeft: 80,

        textAlign: 'center'

    },

    mainContainer: {

        backgroundColor: 'white'

    },

    resultContainer: {

        backgroundColor: 'white',

        height: '100%',

    },

    rowTable: {

        flexDirection: 'row',

        justifyContent: 'space-around',

        alignItems: 'center'

    },

    dataHeader: {

    },

    dataText: {

        fontSize: 20

    }

})

файл client/scr/screens/SearchScreen.js

import React, { useState } from 'react'

import { View, TextInput, Text, Keyboard, Pressable, StyleSheet, ToastAndroid, ScrollView, Alert } from 'react-native'

import MaterialIcons from 'react-native-vector-icons/MaterialIcons'

import List from '../components/List'

//new

import { useSelector } from 'react-redux'

export default function SearchScreen({ navigation }) {

    const [text, setText] = useState('');

    const [searchList, setSearchList] = useState([]);

    const data = useSelector(state => state.articles.articles)

    const search = (data, searchPhrase) => {

        const output = [];

        let sp = searchPhrase.toLowerCase();

        if (sp == '' || sp == " ") {

            return output;

        }

        for (let i = 0; i < data.length; i++) {

            const titleData = data[i].title.toLowerCase();

            if (titleData == sp) {

                output.push(data[i]);

                continue;

            }

            if (titleData.includes(sp)) {

                output.push(data[i]);

                continue;

            }

        }

        return output;

    }

    const checkSearch = (output) => {

        if (output.length == 0) {

            Alert.alert("Ничего не найдено");

        }

    }

    const buttonHandler = () => {

        const result = search(data, text);

        setText('');

        checkSearch(result);

        setSearchList(result);

        Keyboard.dismiss()

    }

    return (

        <View style={styles.scrCont}>

            <View style={styles.searchContainer}>

                <MaterialIcons name="search" size={24} color="black" style={{ marginRight: 5 }} />

                <TextInput placeholder='Введите показатель' style={styles.textInput} value={text} onChangeText={newText => setText(newText)} />

                <Pressable onPress={buttonHandler} style={styles.searchBtn}>

                    <Text style={styles.textBtn}>Искать</Text>

                </Pressable>

            </View>

            <ScrollView style={styles.scrl}>

                <List data={searchList} navigation={navigation} />

            </ScrollView>

        </View>

    )

}

const styles = StyleSheet.create({

    searchContainer: {

        backgroundColor: 'white',

        flexDirection: 'row',

        alignItems: 'center',

        paddingHorizontal: 10,

        marginVertical: 10,

    },

    textInput: {

        width: '70%',

        fontSize: 22

    },

    scrCont: {

        backgroundColor: 'white',

        borderStyle: 'solid',

        borderBottomColor: '#DCDCDC',

        borderBottomWidth: 1,

        flex: 1

    },

    searchBtn: {

        backgroundColor: 'white'

    },

    textBtn: {

        color: "#007bff",

        fontSize: 22

    },

    scrl: {

        backgroundColor: 'white'

    }

})

файл client/src/store/index.js

import { configureStore } from "@reduxjs/toolkit";

import favouriteReducer from './favouriteSlice';

import articlesReducer from "./articlesSlice";

import counterReducer from "./counterSlice"

import usersReducer from "./usersSlice";

export default configureStore({

    reducer: {

        favourites: favouriteReducer,

        articles: articlesReducer,

        counter: counterReducer,

        users: usersReducer

    },

})

файл server/src/Controller/articleController.php

<?php

namespace App\Controller;

use App\Entity\Article;

use App\Service\ArticleService;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\JsonResponse;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\Routing\Annotation\Route;

class ArticleController extends AbstractController

{

    private ArticleService $articleService;

    public function \_\_construct(

        ArticleService $articleService

    )

    {

        $this->articleService = $articleService;

    }

    /\*\*

     \* @Route("/api/articles")

     \*/

    public function getAllArticles(): JsonResponse

    {

        $articles = $this->articleService->getAllArticles();

        $result = [];

        foreach ($articles as $article) {

            /\*\* @var Article $article \*/

            $result[] = [

                'id' => $article->getId(),

                'title' => $article->getTitle(),

                'description' => $article->getDescription(),

                'category' => $article->getCategory(),

                'norm' => $article->getNorm()

            ];

        }

        return $this->json($result);

    }

    /\*\*

     \* @Route("/api/add/favorite/article")

     \*/

    public function addToFavorite(Request $request): JsonResponse

    {

        $content = json\_decode($request->getContent(), true)[0];

        $user = $content['userId'];

        $article = $content['articleId'];

        $this->articleService->bindArticleToUser($user, $article);

        return $this->json('Articles added to favorite!');

    }

    /\*\*

     \* @Route("/api/get/{userId}/articles")

     \*/

    public function getFavoriteArticlesByUser(int $userId): JsonResponse

    {

       $articles =  $this->articleService->getFavoriteArticlesByUser($userId);

       $result = [];

       foreach ($articles as $article) {

            $result[] = [

                'id' => $article->getId(),

                'title' => $article->getTitle(),

                'description' => $article->getDescription(),

                'category' => $article->getCategory()

            ];

       }

       return $this->json($result);

    }

}

файл server/scr/controllers/authController.php

<?php

namespace App\Controller;

use App\Entity\User;

use App\Service\AuthService;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\JsonResponse;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\Routing\Annotation\Route;

class AuthController extends AbstractController

{

    private AuthService $authService;

    public function \_\_construct(

        AuthService $authService

    )

    {

        $this->authService = $authService;

    }

    /\*\*

     \* @Route("/api/auth")

     \*/

    public function login(Request $request): JsonResponse

    {

        $content = json\_decode($request->getContent(), true)[0];

        $login = $content['login'];

        $password = $content['password'];

        $user = $this->authService->login($login, $password);

        if ($user instanceof User) {

            return $this->json('Login successful!');

        } else {

            return $this->json('User not found');

        }

    }

    /\*\*

     \* @Route("/api/registration")

     \*/

    public function registration(Request $request): JsonResponse

    {

        $content = json\_decode($request->getContent(), true)[0];

        $login = $content['login'];

        $password = $content['password'];

        $user = $this->authService->registration($login, $password);

        if ($user->getLogin() === $login && $user->getPassword() === $password) {

            return $this->json('User successful created!');

        } else {

            return $this->json('User not created!');

        }

    }

}

файл server/src/Entity/Article.php

<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

use App\Repository\ArticleRepository;

/\*\*

 \* @ORM\Entity(repositoryClass=ArticleRepository::class)

 \*/

class Article

{

    /\*\*

     \* @ORM\Id()

     \* @ORM\GeneratedValue()

     \* @ORM\Column(type="integer")

     \*/

    private int $id;

    /\*\*

     \* @ORM\Column(type="string")

     \*/

    private string $category;

    /\*\*

     \* @ORM\Column(type="string")

     \*/

    private string $title;

    /\*\*

     \* @ORM\Column(type="string")

     \*/

    private string $norm;

    /\*\*

     \* @ORM\Column(type="text")

     \*/

    private string $description;

    /\*\*

     \* @return int

     \*/

    public function getId(): int

    {

        return $this->id;

    }

    /\*\*

     \* @param int $id

     \*/

    public function setId(int $id): void

    {

        $this->id = $id;

    }

    /\*\*

     \* @return string

     \*/

    public function getCategory(): string

    {

        return $this->category;

    }

    /\*\*

     \* @param string $category

     \*/

    public function setCategory(string $category): void

    {

        $this->category = $category;

    }

    /\*\*

     \* @return string

     \*/

    public function getTitle(): string

    {

        return $this->title;

    }

    /\*\*

     \* @param string $title

     \*/

    public function setTitle(string $title): void

    {

        $this->title = $title;

    }

    /\*\*

     \* @return string

     \*/

    public function getNorm(): string

    {

        return $this->norm;

    }

    /\*\*

     \* @param string $norm

     \*/

    public function setNorm(string $norm): void

    {

        $this->norm = $norm;

    }

    /\*\*

     \* @return string

     \*/

    public function getDescription(): string

    {

        return $this->description;

    }

    /\*\*

     \* @param string $description

     \*/

    public function setDescription(string $description): void

    {

        $this->description = $description;

    }

}

файл server/src/Entity/User.php

<?php

namespace App\Entity;

use Doctrine\Common\Collections\Collection;

use Doctrine\ORM\Mapping as ORM;

use App\Repository\UserRepository;

/\*\*

 \* @ORM\Entity(repositoryClass=UserRepository::class)

 \*/

class User

{

    /\*\*

     \* @ORM\Id()

     \* @ORM\GeneratedValue()

     \* @ORM\Column(type="integer")

     \*/

    private int$id;

    /\*\*

     \* @ORM\Column(type="string")

     \*/

    private string $login;

    /\*\*

     \* @ORM\Column(type="string")

     \*/

    private string $password;

    /\*\*

     \* @ORM\Column(type="string")

     \*/

    private string $role;

    /\*\*

     \* @ORM\ManyToMany(targetEntity="App\Entity\Article", cascade={"persist"})

     \* @ORM\JoinTable(name="favorite\_articles",

     \*      joinColumns={@ORM\JoinColumn(name="user\_id", referencedColumnName="id")},

     \*      inverseJoinColumns={@ORM\JoinColumn(name="articl\_id", referencedColumnName="id")}

     \* )

     \*/

    private Collection $articles;

    /\*\*

     \* @return int

     \*/

    public function getId(): int

    {

        return $this->id;

    }

    /\*\*

     \* @param int $id

     \*/

    public function setId(int $id): void

    {

        $this->id = $id;

    }

    /\*\*

     \* @return string

     \*/

    public function getLogin(): string

    {

        return $this->login;

    }

    /\*\*

     \* @param string $login

     \*/

    public function setLogin(string $login): void

    {

        $this->login = $login;

    }

    /\*\*

     \* @return string

     \*/

    public function getPassword(): string

    {

        return $this->password;

    }

    /\*\*

     \* @param string $password

     \*/

    public function setPassword(string $password): void

    {

        $this->password = $password;

    }

    /\*\*

     \* @return string

     \*/

    public function getRole(): string

    {

        return $this->role;

    }

    /\*\*

     \* @param string $role

     \*/

    public function setRole(string $role): void

    {

        $this->role = $role;

    }

    /\*\*

     \* @return Collection

     \*/

    public function getArticles(): Collection

    {

        return $this->articles;

    }

    /\*\*

     \* @param Article $articles

     \*/

    public function setArticles(Article $articles): void

    {

        $this->articles->add($articles);

    }

}