МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.И. ЛЕНИНА»

Кафедра программного обеспечения компьютерных систем

ОТЧЕТ

ПО ЛАБОРАТОРНЫМ РАБОТАМ

ПО КУРСУ

«РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ»

Выполнила студентка группы 4-42 Е.А. Романова

подпись, дата

Иваново 2023

Оглавление

**1.** [**Задания 2**](#_Toc152850509)

**2.** [**Задания 11**](#_Toc152850519)

[**Ход работы 11**](#_Toc152850520)

**3.** [**Задание 20**](#_Toc152850521)

[**Ход работы 20**](#_Toc152850522)

**4.** [**Задания 24**](#_Toc152850524)

[**Ход работы 25**](#_Toc152850525)

[**Ссылки 33**](#_Toc152850527)

[**Вывод 34**](#_Toc152850528)

**Задания**

1. Выбрать тему и описать будущее приложение (назначение, цели, перечень задач на реализацию)

2. Предоставить макет

3. Создать проект на GitHub

4. Описать инфраструктуру

1. **Оператор сотовой связи**

Тема - разработка мобильного приложения "Оператор сотовой связи" с целью оптимизации управления тарифными планами, клиентской базой и договорами.

Главной целью является упрощение процессов управления тарифными планами, клиентской базой и договорами. Для достижения оптимальных результатов в управлении предполагается создание интерфейса, который бы обеспечивал высокую степень функциональности, предоставляя оператору все необходимые возможности для комфортного и эффективного выполнения своих обязанностей.

Исходя из вышеописанных целей, предполагается внедрение основных операций, таких как добавление, изменение и удаление, а также реализация функционала поиска по названиям и отчетной системы. Посредством данного функционала оператор сможет отслеживать договоры по временным интервалам и анализировать популярность тарифных планов. Кроме того, система будет обладать возможностью архивирования и отслеживания тарифов, блокировки я клиентов, пытающихся воспользоваться определенными тарифными планами, а также предотвращения заключения договоров с использованием данных тарифов.

Предполагается, что разработка мобильного приложения "Оператор сотовой связи" с вышеописанным функционалом значительно улучшит эффективность работы операторов, снизит вероятность ошибок в управлении клиентской базой и тарифными планами.

1. **Макет приложения**

Макет был разработан в программе Figma по ссылке –

<https://www.figma.com/file/yzwO4oi240Jh4HAFfQxnwN/Untitled?type=design&node-id=19%3A810&mode=design&t=1A6CTpjAOamiJPEe-1>

Окно входа и главного окна разработанного макета представлены на рисунках 1-2.

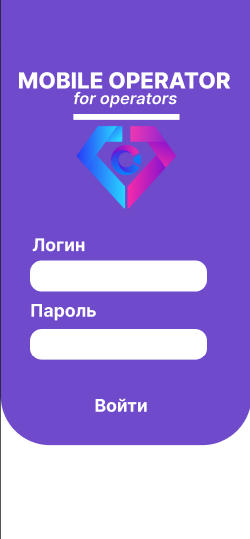
****

Рисунок 1 – Окно входа

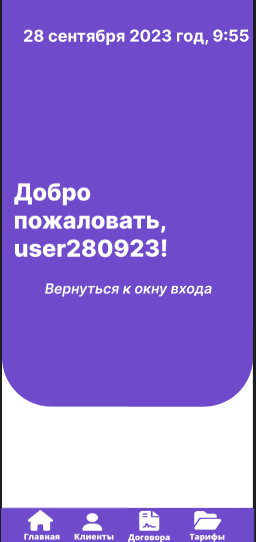


Рисунок 2 – Главное окно

Окно управления тарифами представлено на рисунке 3. В нем присутствует просмотр всех открытых тарифных планов без нагромождения, фильтрация по названию тарифа, табы для отдельного просмотра заархивированных и активных тарифов, а так же кнопки для добавления, изменения и удаления тарифов.



Рисунок 3 – Окно управления тарифами

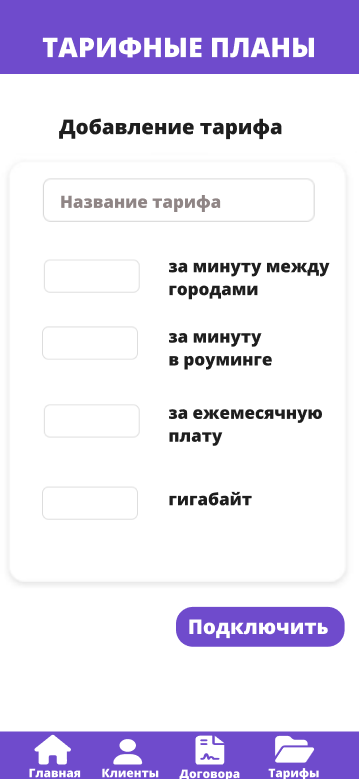


Рисунок 4 – Окно добавления тарифа

Окно управления клиентской базой представлено на рисунке 5. Присутствуют аналогичные по своим функциям кнопки, как указанные выше.

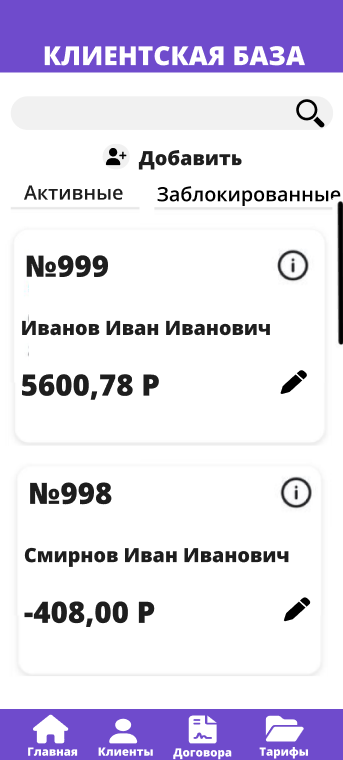


Рисунок 5 – Окно управления клиентской базой

Окно управления договорами представлено на рисунке 6. В нем помимо основных функций представлены два вида отчетности:

1. Вывод самого популярного тарифа.

Проходя по всем договорам, для каждого тарифа свой счетчик высчитывает количество раз, когда первый подключен, и выводит наиболее частоподключаемый

1. Интервальные договора.

Процедура, используя два поля ввода дат – начальной и конечной, выводит все договора, заключенные в этом промежутке.

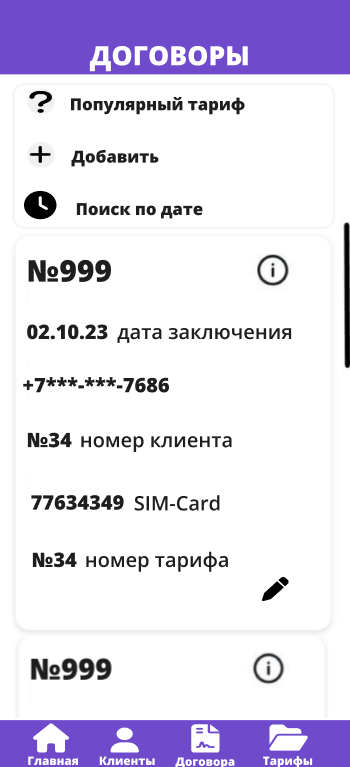


Рисунок 6 – Окно управления договорами

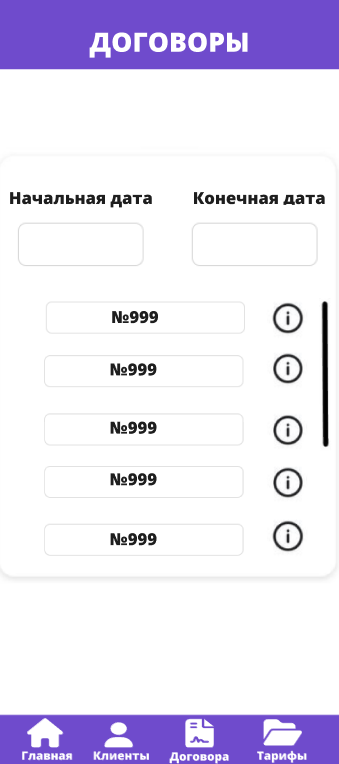


Рисунок 7 – Окно отчета о договорах

1. **Проект в GitHub**

Был создан проект в GitHub

1. **Инфраструктура**

Клиентское приложение будет написано на Java Script.

Серверная часть – на C#.

База данных – SQL Server.

**Задания**

1. Реализовать окна

2. Прописать команды и навигацию

**Ход работы**

В ходе разработки было создано семь компонентов, каждый из которых представляет собой отдельное окно в приложении.

1. **Окно входа в аккаунт оператора/администратора:**
   * Это окно предназначено для входа в систему и аутентификации пользователя.
2. **Окно приветствия:**
   * Приветственное окно отображается после успешного входа в систему и предназначено для приветствия пользователя.
3. **Окно клиентской базы:**
   * Данное окно содержит информацию о клиентах.

3.1. **Дополнительное окно добавления клиента:** дополнительное окно используется для ввода информации и добавления новых клиентов в базу данных.

1. **Окно тарифных планов:**
   * Это окно отображает информацию о доступных тарифных планах.

4.1. **Дополнительное окно открытия тарифного плана:** дополнительное окно предназначено для ввода информации и добавления новых тарифов в базу данных.

5**. Окно договоров:**

* + Это окно предназначено для отображения информации о договорах между клиентами и организацией.

5.1. **Дополнительное окно заключения нового договора:** - Дополнительное окно используется для заполнения информации и создания нового договора с клиентом.

Отдельно был реализован компонент **Menu**, который служит для создания навигационного меню в приложении, обеспечивая удобную навигацию между различными окнами.

Начало формы

Результат добавления компонентов представлен на рисунке 1.

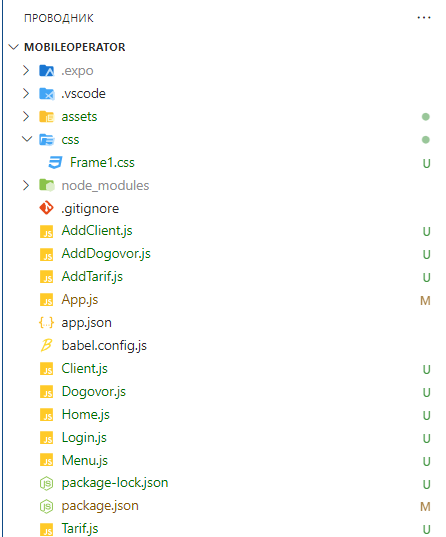


Рисунок 1 – Компоненты

Открытые экранные формы запущенного приложения представлены на рисунках 2-7.



Рисунок 2 – Окно входа в аккаунт



Рисунок 3 – Окно приветствия

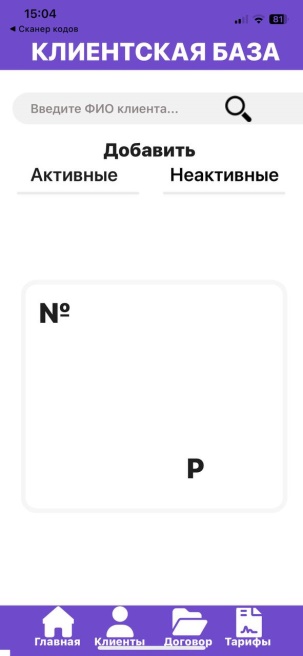


Рисунок 4 – Окно клиентской базы

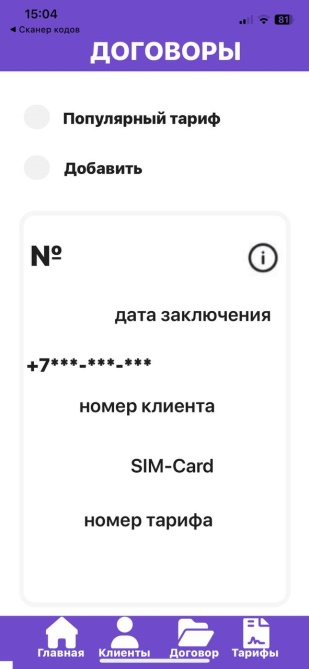


Рисунок 5 – Окно договоров



Рисунок 6 – Окно тарифных планов



Рисунок 7 – Окно добавления тарифа

Рассмотрим некоторые компоненты подробнее:

1. Данный код представляет компонент **Menu**, который служит в качестве навигационного меню в приложении.

В компоненте определены функции **handleReturnToTarif**, **handleReturnToDogovor**, **handleReturnToClient** и **handleReturnToHome**, которые вызывают методы навигации при нажатии на элементы, которые реализованы с использованием компонента **TouchableOpacity**. Каждый из них связан с соответствующим обработчиком навигации.

import React from "react";

import { View, Text, TouchableOpacity, Image } from "react-native";

import { StyleSheet } from "react-native";

const handleReturnToTarif = () => {

    navigation.navigate("Tarif");

  };

  const handleReturnToDogovor = () => {

    navigation.navigate("Dogovor");

  };

  const handleReturnToClient = () => {

    navigation.navigate("Client");

  };

  const handleReturnToHome = () => {

    navigation.navigate("Home");

  };

const Menu = ({ navigation, currentDate, handleReturnToLogin, handleReturnToTarif, handleReturnToDogovor, handleReturnToClient, handleReturnToHome }) => {

  return (

    <View style={styles.flexWrapperTwo}>

      <View style={styles.rectangle15} />

      <View style={styles.rectangle15Two} />

      <View style={styles.rectangle15Three} />

      <Image

        style={styles.Client}

        source={{

          uri: "https://i.ibb.co/mymKLkx/image.png",

        }}

      />

      <Image

        style={styles.Home}

        source={{

          uri: "https://i.ibb.co/Pmz2PQf/image.png",

        }}

      />

    <Image

        style={styles.Tarif}

        source={{

          uri: "https://i.ibb.co/T8RM1Gc/image.png",

        }}

      />

      <Image

        style={styles.Dogovor}

        source={{

          uri: "https://i.ibb.co/W5HJLVd/image.png",

        }}

      />

      <TouchableOpacity onPress={handleReturnToHome}>

            <Text style={styles.glavnajaTwo}>Главная</Text>

            </TouchableOpacity>

            <TouchableOpacity onPress={handleReturnToClient}>

      <Text style={styles.klientyTwo}>Клиенты</Text>

      </TouchableOpacity>

      <TouchableOpacity onPress={handleReturnToDogovor}>

        <Text style={styles.dogovoraTwo}>Договора</Text>

      </TouchableOpacity>

      <TouchableOpacity onPress={handleReturnToTarif}>

        <Text style={styles.tarifyTwo}>Тарифы</Text>

      </TouchableOpacity>

    </View>

  );

};

export default Menu;

1. Client представляет компонент, организующий интерфейс для отображения клиентской базы.

**Структура компонента:**

* View и стили:

Используется View для создания основного контейнера, в котором располагаются другие компоненты.

Используется объект styles из StyleSheet.create для стилизации компонента.

* Отображение клиентской базы:

В компоненте реализован интерфейс для отображения информации о клиентах, их статусе, балансе и т.д.

* Навигационное меню:

Внутри компонента встроен компонент Menu, который предоставляет навигацию к различным разделам приложения.

* Иконки и кнопки:

Используются изображения (Image) для иконок и кнопок.

Кнопка "Добавить" (TouchableOpacity), при нажатии на которую происходит навигация к экрану добавления клиента (handleAddClick).

import React from "react";

import { View, Text, Image, StyleSheet, TouchableOpacity } from "react-native";

import moment from "moment";

import 'moment/locale/ru';

import Menu from "./Menu";

moment.locale('ru');

const currentDate = moment().format("DD MMMM YYYY [год], H:mm");

const Tarif = ({ navigation }) => {

  const handleReturnToHome = () => {

    navigation.navigate("Home");

  };

  const handleReturnToTarif = () => {

    navigation.navigate("Tarif");

  };

  const handleReturnToDogovor = () => {

    navigation.navigate("Dogovor");

  };

  const handleReturnToClient = () => {

    navigation.navigate("Client");

  };

  const handleAddClick = () => {

    navigation.navigate("AddTarif");

  };

  return (

    <View style={styles.frame26}>

      <View style={styles.frame3}>

        <View style={styles.frame19}>

          <View style={styles.flexWrapperOne}>

            <View style={styles.rectangle15} />

            <View style={styles.rectangle11} />

            <View style={styles.rectangle20} />

            <View style={styles.rectangle21} />

            <Text style={styles.tarifnyePlany}>ТАРИФНЫЕ ПЛАНЫ</Text>

            <Text style={styles.otkrytNovyjTarifnyjPlan}>Открыть новый тарифный план</Text>

            <View style={styles.rectangle25} />

            <View style={styles.pngwing14} />

            <View style={styles.pngwing15} />

            <View style={styles.rectangle26} />

            <View style={styles.rectangle37} />

            <Image

              style={styles.pngwing19}

              source={{ uri: "https://static.overlay-tech.com/assets/37f29226-098f-4176-95b7-4d2dc6f94c4d.png" }}

            />

            <Image

              style={styles.image2}

              source={{ uri: "https://static.overlay-tech.com/assets/9007b6c4-bdfa-4877-ba7e-2201a3be44ab.png" }}

            />

            <Image

              style={styles.image3}

              source={{ uri: "https://static.overlay-tech.com/assets/b515bfad-d66c-478d-8bc1-455b5317afe4.png" }}

            />

            <Text style={styles.arhivnye}>Архивные</Text>

            <Text style={styles.aktivnye}>Активные</Text>

            <TouchableOpacity style={styles.button} onPress={handleAddClick}>

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

            </TouchableOpacity>

            <Text style={styles.bezlimit}>

             безлимит

            </Text>

            <Text style={styles.vnutriSeti}>

            внутри сети

            </Text>

            <Text style={styles.minutaTown}>

           за минуту{"\n"}между городами

            </Text>

            <Text style={styles.minutaRouming}>

           за минуту{"\n"}в роуминге

            </Text>

            <Text style={styles.num999}>Р/месяц</Text>

            <Text style={styles.vvediteNazvanieTarifa}>Введите название тарифа...</Text>

          </View>

        </View>

      </View>

      <View style={styles.menuContainer}>

        <Menu

          navigation={navigation}

          currentDate={currentDate}

          handleReturnToTarif={handleReturnToTarif}

          handleReturnToDogovor={handleReturnToDogovor}

          handleReturnToClient={handleReturnToClient}

          handleReturnToHome={handleReturnToHome}

        />

      </View>

      <View style={styles.flexWrapperTwo}>

        <View style={styles.rectangle15Two} />

        <View style={styles.folderOpenSolid1Two}>

        </View>

        <View style={styles.fileContractSolid1}>

        </View>

      </View>

    </View>

  );

};

export default Tarif;

**Задание**

Спроектировать остальные окна, реализовать готовое приложение

**Ход работы**

Для оптимизации просмотра элементов в приложении было принято решение воспользоваться компонентом "Card", интегрированным из библиотеки "react-native-elements".

Добавление новых клиентов в систему осуществляется через создание новой карточки с использованием метода "map" для обработки данных в массиве "cardData". const [cardData, setCardData] = useState([

    {

      isEditing: false,

    },

  ]);

Введено состояние "isEditing" в массив "cardData" для управления режимом редактирования.

Дополнительные функции, предназначенные для реализации операций CRUD, включают следующие методы:

const handleRedactirovatClick = (index) => {

    const updatedCardData = [...cardData];

    updatedCardData[index].isEditing = !updatedCardData[index].isEditing;

    setCardData(updatedCardData);

  };

  const handleDeleteClick = (index) => {

    setCardData((prevCardData) => {

      const updatedCardData = [...prevCardData];

      updatedCardData.splice(index, 1); // Remove the card at the specified index

      return updatedCardData;

    });

  };

  const handleAdd1Click = () => {

    const newCard = {

      isEditing: true, // Set isEditing to true for the newly added card

      balance: "",

      fio: "",

      num999: "",

    };

    setCardData((prevCardData) => [newCard, ...prevCardData]);

    setCardCount(cardCount + 1);

  };

Для обеспечения удобного просмотра доступных карточек в приложении был внедрен компонент "ScrollView".

<View style={styles.cardForContainer}>

  <ScrollView style={styles.cardScrollView} contentContainerStyle={styles.cardScrollContainer}>

    {cardData.map((card, index) => (

      <Card key={index} containerStyle={styles.cardContainer}>

        <Text style={styles.balance}>Р</Text>

        <Text style={styles.num999}>№</Text>

        <TouchableOpacity onPress={() => handleRedactirovatClick(index)}>

          <Image

            style={styles.redactirovat}

            source={{ uri: "https://i.ibb.co/fvWPsby/image.png" }}

          />

        </TouchableOpacity>

        <TouchableOpacity onPress={() => handleDeleteClick(index)}>

              <Image

                style={styles.delete}

                source={{ uri: "https://i.ibb.co/SVmDTwG/pngwing-com-8.png" }}

              />

            </TouchableOpacity>

        {card.isEditing ? (

          <>

            <TextInput

              style={styles.inputBalance}

              onBlur={() => setIsEditing(false)}

            />

            <TextInput

              style={styles.inputFIO}

              onBlur={() => setIsEditing(false)}

            />

            <TextInput

              style={styles.inputnum999}

              onBlur={() => setIsEditing(false)}

            />

          </>

        ) : null}

      </Card>

    ))}

  </ScrollView>

</View>

Результаты представлены на рисунках 1-2.

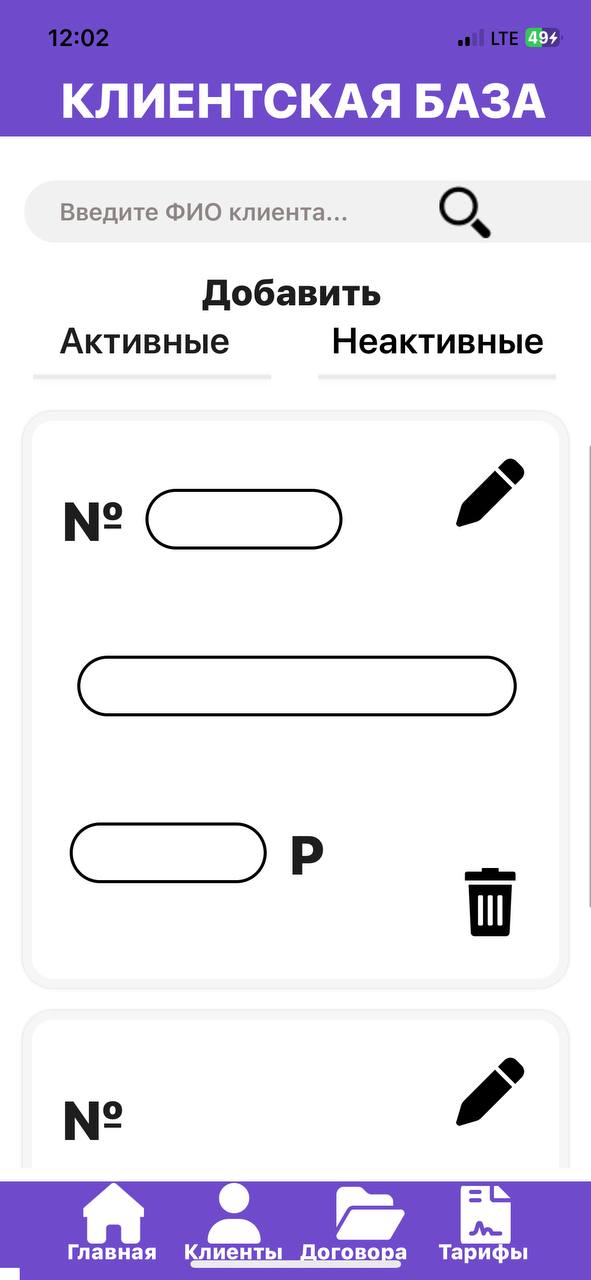


Рисунок 1 – Реализуемое пролистывание карточек

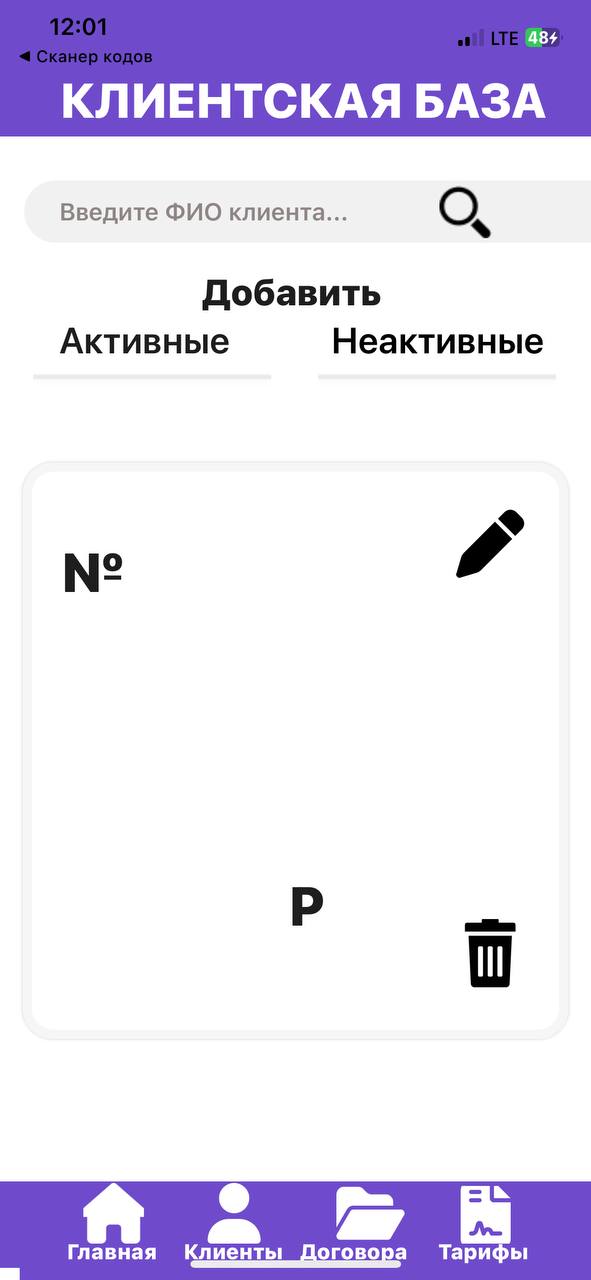


Рисунок 2 – Карточка

Также были настроены остальные экраны приложения в аналогичном стиле.

**Задания**

1. Подключить сервер
2. Реализовать бизнес-логику

**Ход работы**

Сервер расположен по адресу http://172.20.10.9:5050/swagger/index.html



Пример контроллера из серверной части для клиентской базы:

namespace ASPNetCoreApp.Controllers

{

[Route("api/[controller]")]

//[EnableCors]

[ApiController]

public class ClientsController : ControllerBase

{

private readonly OperatorContext \_context;

public ClientsController(OperatorContext context)

{

\_context = context;

if (\_context.Client.Count() == 0)

{

\_context.Client.Add(new Client

{

Balance=1,

Client\_number = 1,

Full name = "Ivanov I.I."

});

\_context.SaveChanges();

}

}

[HttpGet]

public IEnumerable<Client> GetAll()

{

return \_context.Client;

}

[HttpGet("{id}")]

public async Task<IActionResult> GetClient([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var Client = await \_context.Client.SingleOrDefaultAsync(m => m.Client\_Number == id);

if (Client == null)

{

return NotFound();

}

return Ok(Client);

}

[HttpPost]

// [Authorize(Roles = "user")]

public async Task<IActionResult> Create([FromBody] Client Client)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

\_context.Client.Add(Client);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetClient", new { Client\_Code = Client. Client\_Number }, Client);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update([FromRoute] int id, [FromBody] Client Client)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var item = \_context.Client.Find(id);

if (item == null)

{

return NotFound();

}

item.Client\_Number = Client.Client\_Number;

\_context.Client.Update(item);

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpDelete("{id}")]

// [Authorize(Roles = "user")]

public async Task<IActionResult> Delete([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var item = \_context.Client.Find(id);

if (item == null)

{

return NotFound();

}

\_context.Client.Remove(item);

await \_context.SaveChangesAsync();

return NoContent();

}

}

Соответственно запросы к серверу из клиентской части представлены следующим образом:

Запрос на получение всех клиентов:

  useEffect(() => {

    const getКлиентs = async () => {

      try {

        const response = await axios.get("http://172.20.10.9:5050/api/Клиентs");

        setCardData(response.data);

        setOriginalCardData(response.data); // Save the original data

      } catch (error) {

        console.error('Error fetching data:', error.message);

      }

    }

    getКлиентs();

  }, []);

Запрос на удаление клиента из клиентской базы:

 const handleDeleteClick = async (номер\_клиента, index) => {

    try {

      const response = await fetch(`http://172.20.10.9:5050/api/клиентs/${номер\_клиента}`, {

        method: 'DELETE',

      });

      if (response.ok) {

        console.log('Клиент успешно удален на сервере');

        await getКлиентs();

        setCardData((prevCardData) => {

          const updatedCardData = [...prevCardData];

          updatedCardData.splice(index, 1);

          return updatedCardData;

        });

      } else {

        console.error('Ошибка при удалении клиента на сервере:', response.status);

      }

    } catch (error) {

      console.error('Ошибка при отправке запроса:', error.message);

    }

  };

Запрос на добавление нового клиента в клиентскую базу:

  const handleAddClick = async () => {

    const Клиент = {

      номер\_клиента,

      баланс,

      фио,

    };

    const requestOptions = {

      method: 'POST',

      headers: { 'Content-Type': 'application/json' },

      body: JSON.stringify(Клиент),

    };

    try {

      const response = await fetch("http://172.20.10.9:5050/api/клиентs/", requestOptions);

      if (!response.ok) {

        // Check for an error status and handle it

        console.error('Ошибка при добавлении клиента на сервер:', response.status);

        const errorText = await response.text();

        console.log('Error response text:', errorText);

        return;

      }

      const data = await response.json();

      console.log(data);

      // If response is successful, call addКлиент function

      addКлиент(data);

    } catch (error) {

      // Handle general errors

      console.error('Ошибка при отправке запроса:', error);

    }

  };

Аналогично запросы были прописаны для договоров и тарифов.

Помимо основных запросов была реализована бизнес-логика.

Функция для поиска клиента по ФИО:

  const searchКлиент = (value) => {

    const lowercaseValue = value.toLowerCase();

    if (lowercaseValue === '') {

      setCardData(originalCardData); // Reset to original data when input is empty

    } else {

      const filteredData = originalCardData.filter(

        (клиент) => клиент.фио.toLowerCase().includes(lowercaseValue)

      );

      setCardData(filteredData);

    }

  };

Функция фильтрации клиентов в зависимости от баланса:

  const filteredКлиентs = cardData.filter((клиент) => {

    if (filterStatus === 'Активные') {

      return клиент.баланс > 0;

    } else if (filterStatus === 'Неактивные') {

      return клиент.баланс <= 0;

    }

    return false;

  });

Аналогично было прописано для страницы тарифов.

Для договоров была реализована функция вывода самого популярного тарифа по количеству подключений:

      const getMostPopularTariff = (data) => {

        const tariffMap = new Map();

        data.forEach(dogovor => {

          const tariffCode = dogovor.код\_тарифа\_FK;

          if (tariffMap.has(tariffCode)) {

            tariffMap.set(tariffCode, tariffMap.get(tariffCode) + 1);

          } else {

            tariffMap.set(tariffCode, 1);

          }

        });

        let maxTariffCode;

        let maxTariffCount = 0;

        tariffMap.forEach((count, code) => {

          if (count > maxTariffCount) {

            maxTariffCode = code;

            maxTariffCount = count;

          }

        });

        return `Популярный тариф №${maxTariffCode} \n подключений - ${maxTariffCount}`;

      };

Результаты представлены на рисунках 1-4.



Рисунок 1 – Тарифные планы

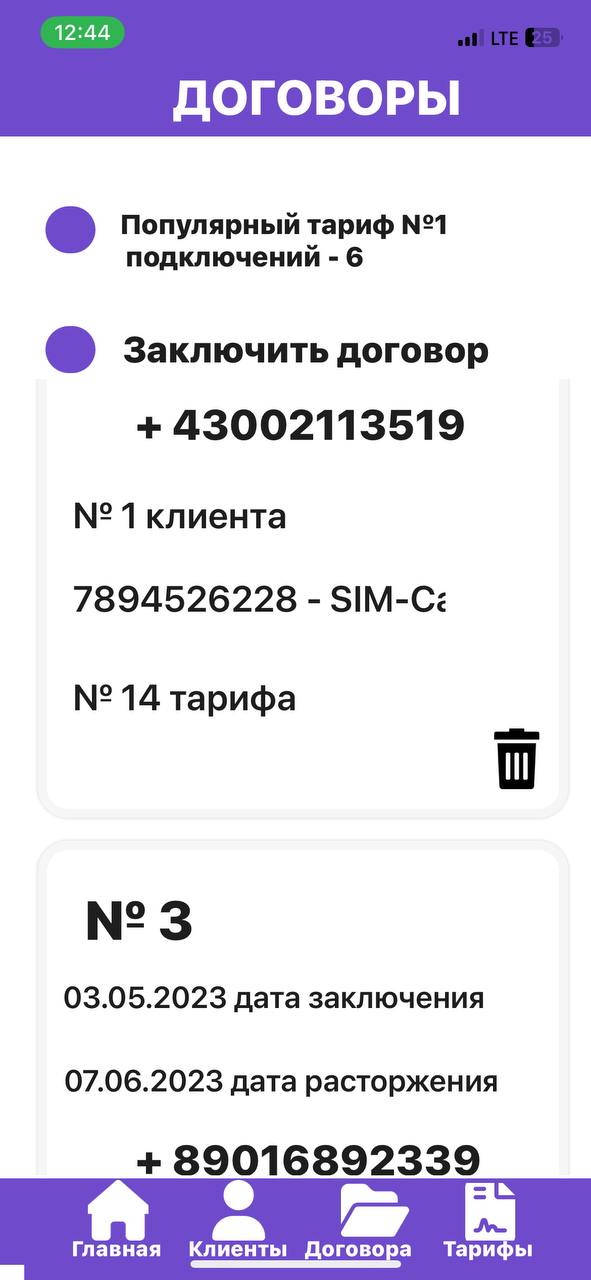


Рисунок 2 – Договоры

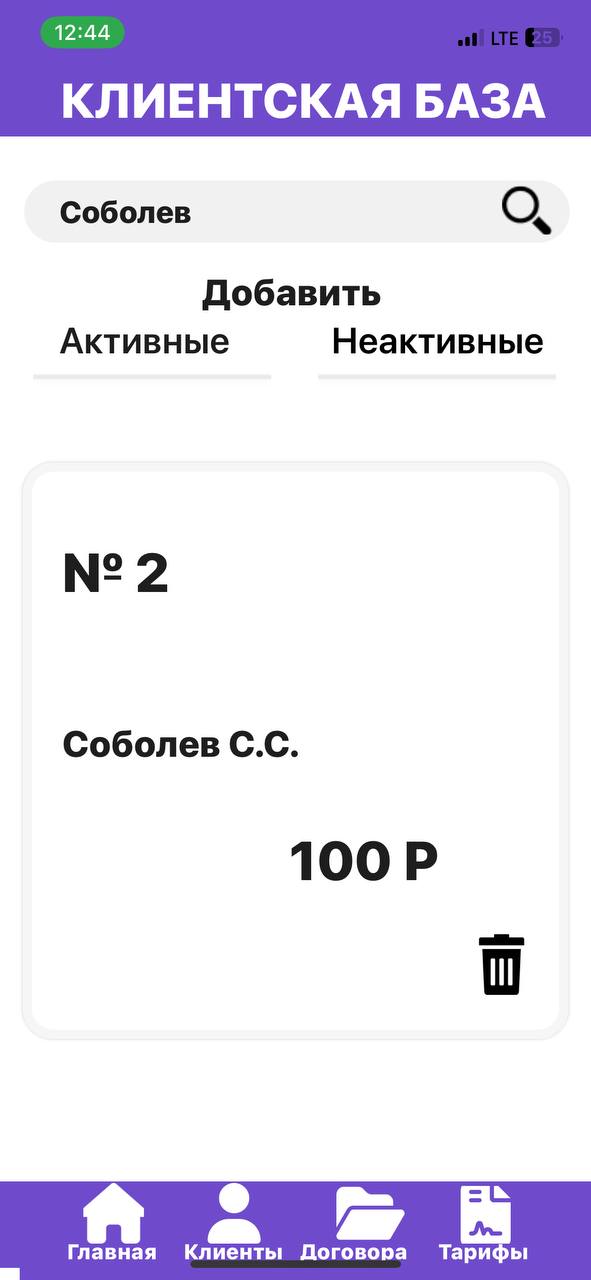


Рисунок 3 – Поиск клиента по фио



Рисунок 4 – Клиентская база

# Ссылки

Клиентская часть:

<https://github.com/krrinokk/MobileOperator-MO>

Серверная часть:

<https://github.com/krrinokk/ASPNetCore-MobileOperator>

# Вывод

В ходе лабораторных работ было создано мобильное приложение «MobileOperator» для управления услуг оператора сотовой связью с реализацией основных функций управления и бизнес-логикой для работы с базой данных.