Лабораторная работа 5. Разработка клиента

Клиент-серверные приложения представляют собой архитектурную модель, где взаимодействие между клиентом (пользовательским интерфейсом) и сервером (хранилищем данных или обработчиком логики) происходит по сети. Основные компоненты и их взаимодействие:

1. Клиент

Клиент — это программа, которая взаимодействует с пользователем и отправляет запросы к серверу. Чаще всего это:

- Веб-приложение (броузер), которое использует HTML, CSS, JavaScript для взаимодействия с сервером через HTTP-запросы.
- Десктопное приложение (например, на Java с использованием Swing или JavaFX), которое отправляет запросы на сервер.
- Мобильное приложение, которое отправляет запросы по сети к серверу для получения или отправки данных.

2. Сервер

Сервер — это программа, которая обрабатывает запросы клиента. Сервер содержит бизнес-логику и взаимодействует с базой данных для получения, обработки и отправки данных обратно клиенту. В типичном случае это:

- Веб-сервер, работающий на Java (например, Spring Boot) или другом языке программирования.
- Сервер может обрабатывать запросы на основе протоколов HTTP/HTTPS или на локальном сервере.

3. Процесс взаимодействия

1. Запрос клиента: когда пользователь взаимодействует с клиентским приложением (например, нажимает кнопку "отправить данные"), клиент отправляет запрос на сервер. Запрос содержит данные (например, форму регистрации или авторизации), которые клиент хочет передать на сервер.
2. Обработка сервером: сервер получает запрос, проверяет его корректность, выполняет бизнес-логику (например, проверяет авторизационные данные, сохраняет информацию в базу данных или извлекает данные).
3. Ответ от сервера: после выполнения необходимой логики сервер отправляет ответ клиенту. Это может быть данные в формате JSON или XML, статусный код (например, успешная операция 200 OK или ошибка 500 Internal Server Error).

4. Отображение на клиенте: Клиент получает ответ от сервера и обновляет интерфейс пользователя, отображая результат запроса.

4. Протоколы связи

- HTTP/HTTPS — основной протокол взаимодействия веб-клиента с сервером.
- WebSocket — позволяет поддерживать постоянное соединение для отправки данных в реальном времени (например, чат-приложения).
- REST и GraphQL — популярные архитектурные подходы для проектирования API.

5. База данных

Сервер часто взаимодействует с базой данных, хранящей данные пользователей, задачи и т.д. Обычно сервер выполняет SQL-запросы к реляционной базе данных (например, PostgreSQL, MySQL) или использует NoSQL-базы данных (MongoDB).

6. Безопасность

Клиент-серверные приложения должны обеспечивать безопасность данных. Например:

- Шифрование данных при передаче (SSL/TLS для HTTPS).
- Авторизация и аутентификация (OAuth, JWT).

Таким образом, клиентское приложение запрашивает и получает данные с сервера, а сервер обрабатывает эти запросы и управляет хранением и манипуляцией данными.

Цель работы: получить практические навыки работы с Spring Framework. Разработать клиент

Пример выполнения работы.

Для начала необходимо модернизировать существующее приложение для предоставления.

Новый код класса TaskController.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping("/")
public class TaskController {
    @Autowired
    private TaskRepository taskRepository;

    // Веб-методы
    @GetMapping("/")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR')") // Доступ для
пользователей и модераторов
    public String index(Model model) {
        List<Task> tasks = taskRepository.findAll();
        model.addAttribute("tasks", tasks);
        return "index";
    }

    @PostMapping("/addTask")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR')") // Доступ для
пользователей и модераторов
    public String addTask(@ModelAttribute Task task) {
        taskRepository.save(task);
        return "redirect:/";
    }

    @PreAuthorize("hasRole('MODERATOR')") // Только модераторы могут удалять
задачи
    @GetMapping("/deleteTask/{id}")
    public String deleteTask(@PathVariable Long id) {
        taskRepository.deleteById(id);
        return "redirect:/";
    }

    @PreAuthorize("hasRole('MODERATOR')") // Только модераторы могут
обновлять задачи
    @PostMapping("/updateTask/{id}")
    public String updateTask(@PathVariable Long id, @RequestParam boolean
completed) {
        Task task = taskRepository.findById(id).orElseThrow();
        task.setCompleted(completed);
        taskRepository.save(task);
        return "redirect:/";
```

```
    }

    // REST-методы
    @GetMapping("/api")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR')") // Доступ для
пользователей и модераторов
    public ResponseEntity<List<Task>> getAllTasks() {
        List<Task> tasks = taskRepository.findAll();
        return ResponseEntity.ok(tasks);
    }

}
```

Мы добавили Метод `getAllTasks()`:

1. Аннотация `@GetMapping("/api")`:

- Этот метод будет вызван при HTTP GET-запросе на путь `/api`.
- Это означает, что если клиент (например, приложение на Swing) отправит GET-запрос на `/api`, то вызовется этот метод `getAllTasks()`.

2. Аннотация `@PreAuthorize("hasRole('USER') or hasRole('MODERATOR')")`:

- Эта аннотация указывает, что доступ к методу разрешен только пользователям с ролями USER или MODERATOR.
- Если пользователь не обладает одной из этих ролей, доступ к этому методу будет запрещён, и сервер вернёт ошибку авторизации.

3. Метод `getAllTasks()`:

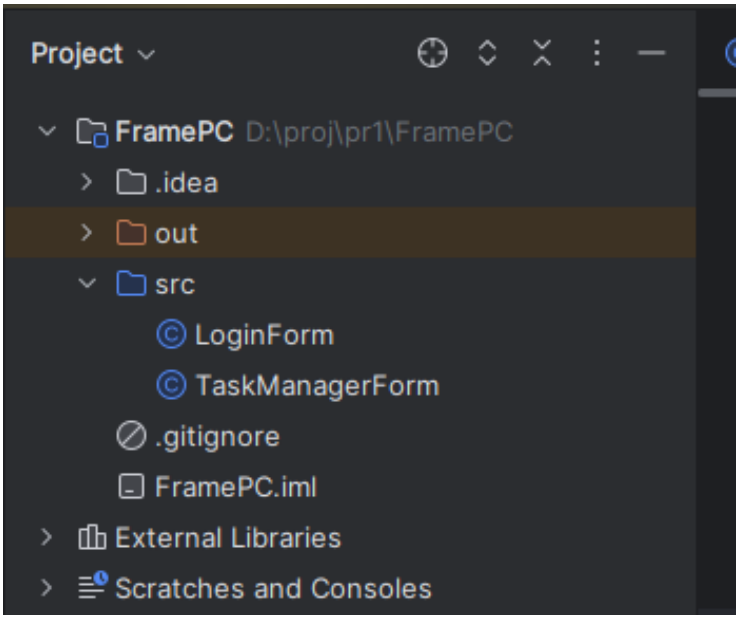
- Этот метод возвращает все задачи из базы данных.
- Внутри метода вызывается `taskRepository.findAll()`, который обращается к репозиторию (например, связанному с базой данных) и извлекает все записи таблицы Task (модель данных задач).
- Полученный список задач передаётся в виде ответа.

4. Возвращаемое значение `ResponseEntity.ok(tasks)`:

- Создаётся объект `ResponseEntity` со статусом 200 OK и телом ответа, которое содержит список задач (`tasks`).
- Этот объект отправляется клиенту (приложению), который запросил данные.

Этот код позволяет получить список всех задач из базы данных при запросе по пути `/api`.

теперь создадим новый проект для клиента. Его структура представлена на скриншоте ниже.



Код класса с формой авторизации:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Base64;

public class LoginForm extends JFrame {

    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;

    public LoginForm() {
        // UI компоненты
        setTitle("Login");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(3, 2));

        // Поля ввода для логина
        usernameField = new JTextField();
        passwordField = new JPasswordField();
        loginButton = new JButton("Login");

        add(new JLabel("Username:"));
        add(usernameField);
        add(new JLabel("Password:"));
        add(passwordField);
        add(new JPanel());  // Пустое место
        add(loginButton);

        // Обработчик авторизации
        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Попытка авторизации
```

```
                String username = usernameField.getText();
                String password = new String(passwordField.getPassword());
                String encodedAuth =
Base64.getEncoder().encodeToString((username + ":" + password).getBytes());

                // Если авторизация успешна, открываем главную форму
                TaskManagerForm taskManagerForm = new
TaskManagerForm(encodedAuth);
                taskManagerForm.setVisible(true);
                dispose();  // Закрываем форму авторизации
                System.out.println("Encoded Auth: " + encodedAuth);
            }
        });
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new LoginForm().setVisible(true);
        });
    }
}
```

Разберем код по частям, чтобы понять, что он делает и как работает.

Основные компоненты кода

Импорт библиотек:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Base64;
```

- `javax.swing.*`: Импортирует все классы из библиотеки Swing, используемой для создания графического интерфейса пользователя (GUI).
- `java.awt.*`: Импортирует классы для работы с компонентами интерфейса и событиями.
- `java.util.Base64`: Импортирует класс для кодирования и декодирования данных в формате Base64.

Класс `LoginForm`:

```
public class LoginForm extends JFrame {
```

- Этот класс расширяет `JFrame`, что означает, что он представляет собой окно GUI.

Объявление полей:

```
private JTextField usernameField;
private JPasswordField passwordField;
private JButton loginButton;
```

- `usernameField`: Поле для ввода имени пользователя.
- `passwordField`: Поле для ввода пароля (с маскировкой символов).
- `loginButton`: Кнопка для выполнения действия авторизации.

Конструктор `LoginForm`:

```
public LoginForm() {
```

- Этот метод вызывается при создании объекта `LoginForm` и инициализирует интерфейс.

Настройка интерфейса:

```
setTitle("Login");
setSize(300, 150);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new GridLayout(3, 2));
```

- Устанавливает заголовок окна, его размер и поведение при закрытии.
- Используется `GridLayout` для упрощения размещения компонентов в сетке 3 строки и 2 столбца.

Создание компонентов:

```
usernameField = new JTextField();
passwordField = new JPasswordField();
loginButton = new JButton("Login");
```

- Создаются текстовые поля и кнопка для ввода данных.

Добавление компонентов в окно:

```
add(new JLabel("Username:"));
add(usernameField);
add(new JLabel("Password:"));
add(passwordField);
add(new JPanel());  // Пустое место
add(loginButton);
```

- Создаются метки для полей ввода и сами поля, которые добавляются в окно.

Обработчик события для кнопки "Login":

```
loginButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Попытка авторизации
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());
```

```
            String encodedAuth =
Base64.getEncoder().encodeToString((username + ":" +
password).getBytes());

            // Если авторизация успешна, открываем главную форму
            TaskManagerForm taskManagerForm = new
TaskManagerForm(encodedAuth);
            taskManagerForm.setVisible(true);
            dispose();  // Закрываем форму авторизации
            System.out.println("Encoded Auth: " + encodedAuth);
        }
});
```

- Когда пользователь нажимает кнопку "Login", вызывается обработчик события.
- Получает введенные имя пользователя и пароль.
- Кодирует их в формате Base64 в виде строки `username:password`.
- Создает новый объект `TaskManagerForm`, передавая закодированную строку авторизации.
- Показывает главное окно (TaskManagerForm) и закрывает форму авторизации с помощью `dispose()`.
- Выводит закодированную строку в консоль для отладки.

Метод `main`:

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        new LoginForm().setVisible(true);
    });
}
```

- Этот метод является точкой входа для запуска приложения.
- Он использует `SwingUtilities.invokeLater`, чтобы создать и отобразить экземпляр `LoginForm` в потокобезопасной среде.

Этот код создает простую форму авторизации с полями для ввода имени пользователя и пароля. При нажатии на кнопку "Login" выполняется кодирование введенных данных в строку формата Base64 и открывается главное окно приложения `TaskManagerForm`. Форма авторизации закрывается, и закодированная строка выводится в консоль. Внешний вид формы представлен ниже.

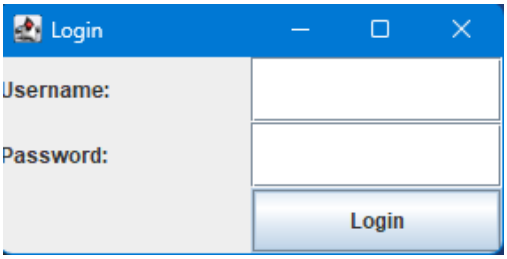


Рис. 1 – форма логина

Создадим теперь форму для вывода задач на экран:

```
import javax.swing.*;
import java.awt.*;
import java.io.BufferedReader;
```

```
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

public class TaskManagerForm extends JFrame {

    private JList<String> taskList;
    private DefaultListModel<String> taskListModel;
    private String authHeader;

    public TaskManagerForm(String authHeader) {
        this.authHeader = authHeader;
        setTitle("Task Manager");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        // Список задач
        taskListModel = new DefaultListModel<>();
        taskList = new JList<>(taskListModel);
        loadTasks(); // Загрузка задач из API
        add(new JScrollPane(taskList), BorderLayout.CENTER);

        // Панель для кнопок
        JPanel buttonPanel = new JPanel();
        JButton refreshButton = new JButton("Обновить задачи");
        buttonPanel.add(refreshButton);
        add(buttonPanel, BorderLayout.SOUTH);

        // Обработчик события для кнопки обновления
        refreshButton.addActionListener(e -> loadTasks());
    }
    private void loadTasks() {
        try {
            URL url = new URL("http://localhost:8080/api");
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Authorization", "Basic " + authHeader);

            int responseCode = conn.getResponseCode();
            System.out.println("Response Code: " + responseCode);

            if (responseCode == 200) {
                BufferedReader in = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
                StringBuilder jsonResponse = new StringBuilder();
                String inputLine;

                while ((inputLine = in.readLine()) != null) {
                    jsonResponse.append(inputLine);
                }
                in.close();

                // Парсинг JSON вручную
                parseJson(jsonResponse.toString());
            } else {
                JOptionPane.showMessageDialog(this, "Ошибка загрузки задач");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    private void parseJson(String json) {
        // Удаляем начальный и конечный квадратные скобки
        json = json.trim();
        if (json.startsWith("[") && json.endsWith("]")) {
            json = json.substring(1, json.length() - 1);
        }

        // Разделяем задачи по "},{" и обрабатываем каждую задачу
        String[] tasksArray = json.split("\\},\\{");
        int taskNumber = 1; // Счетчик задач

        // Очищаем список перед добавлением новых задач
        taskListModel.clear();

        for (String task : tasksArray) {
            // Удаляем фигурные скобки
            task = task.replace("{", "").replace("}", "").trim();

            // Извлекаем описание задачи
            String description = extractField(task, "description");
            if (description != null) {
                taskListModel.addElement("Задача " + taskNumber++ + ": " +
description);
            }
        }
    }

    private String extractField(String task, String fieldName) {
        String[] fields = task.split(",");
        for (String field : fields) {
            String[] keyValue = field.split(":");
            if (keyValue.length == 2) {
                String key = keyValue[0].trim().replace("\"", ""); // Убираем
кавычки
                String value = keyValue[1].trim().replace("\"", ""); //
Убираем кавычки
                if (key.equals(fieldName)) {
                    return value; // Возвращаем значение поля
                }
            }
        }
        return null; // Если поле не найдено
    }

    public static void main(String[] args) {
        // Пример использования
        String authHeader = "VXNlcjpwYXNzd29yZA=="; // Замените на ваш
заголовок авторизации
        SwingUtilities.invokeLater(() -> new
TaskManagerForm(authHeader).setVisible(true));
    }
}
```

Основные компоненты кода

1. Импорт библиотек:

```
import javax.swing.*;
import java.awt.*;
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
```

- `javax.swing.*`: Импортирует классы для создания графического интерфейса пользователя.
- `java.awt.*`: Импортирует классы для работы с компонентами интерфейса и компоновкой.
- `java.io.*`: Импортирует классы для работы с потоками ввода-вывода.
- `java.net.*`: Импортирует классы для работы с сетевыми подключениями.
- `java.util.*`: Импортирует классы для работы с коллекциями и утилитами.

2. Класс `TaskManagerForm`:

```
public class TaskManagerForm extends JFrame {
```

- Этот класс расширяет `JFrame`, что делает его окном графического интерфейса.

3. Объявление полей:

```
private JList<String> taskList;
private DefaultListModel<String> taskListModel;
private String authHeader;
```

- `taskList`: Компонент для отображения списка задач.
- `taskListModel`: Модель данных для списка задач, которая управляет содержимым `taskList`.
- `authHeader`: Строка для хранения заголовка авторизации, передаваемого в API.

4. Конструктор `TaskManagerForm`:

```
public TaskManagerForm(String authHeader) {
    this.authHeader = authHeader;
    setTitle("Task Manager");
    setSize(400, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
```

- Конструктор принимает строку `authHeader`, инициализирует окно с заголовком "Task Manager", задает его размер, поведение при закрытии и компоновку.

5. Настройка списка задач:

```
taskListModel = new DefaultListModel<>();
taskList = new JList<>(taskListModel);
loadTasks(); // Загрузка задач из API
add(new JScrollPane(tas
```

- Создается модель данных для списка задач и сам список. Затем вызывается метод `loadTasks()`, который загружает задачи из API. Список добавляется в центральную часть окна с прокруткой.

6. Создание панели для кнопок:

```
JPanel buttonPanel = new JPanel();
JButton refreshButton = new JButton("Обновить задачи");
buttonPanel.add(refreshButton);
add(buttonPanel, BorderLayout.SOUTH);
```

- Создается панель, содержащая кнопку "Обновить задачи", которая добавляется в нижнюю часть окна.

7. Обработчик события для кнопки обновления:

```
refreshButton.addActionListener(e -> loadTasks());
```

- При нажатии на кнопку вызывается метод `loadTasks()`, что позволяет обновить список задач.

8. Метод `loadTasks`:

```
private void loadTasks() {
    try {
        URL url = new URL("http://localhost:8080/api");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Authorization", "Basic " + authHeader);
```

- Метод устанавливает соединение с API по указанному URL, используя HTTP-метод GET и добавляет заголовок авторизации.

```
int responseCode = conn.getResponseCode();
System.out.println("Response Code: " + responseCode);
```

- Получает код ответа от сервера и выводит его в консоль.

```
if (responseCode == 200) {
    BufferedReader in = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
    StringBuilder jsonResponse = new StringBuilder();
    String inputLine;

    while ((inputLine = in.readLine()) != null) {
        jsonResponse.append(inputLine);
    }
    in.close();
```

- Если код ответа равен 200 (успешный запрос), считывает входной поток ответа и формирует строку `jsonResponse`, содержащую весь JSON-ответ от сервера.

```
  parseJson(jsonResponse.toString());
        } else {
            JOptionPane.showMessageDialog(this, "Ошибка загрузки задач");
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

- Если запрос успешен, вызывается метод `parseJson()`, передавая полученный JSON. В противном случае показывается сообщение об ошибке.

9. Метод `parseJson`:

```
private void parseJson(String json) {
    // Удаляем начальный и конечный квадратные скобки
    json = json.trim();
    if (json.startsWith("[") && json.endsWith("]")) {
        json = json.substring(1, json.length() - 1);
    }
```

- Удаляет начальные и конечные квадратные скобки из JSON-строки, так как предполагается, что это массив задач.

```
// Разделяем задачи по "},{" и обрабатываем каждую задачу
String[] tasksArray = json.split("\\},\\{");
int taskNumber = 1; // Счетчик задач
```

- Делит строку на отдельные задачи по разделителю "},{" и инициализирует счетчик задач. Очищает модель списка перед добавлением новых задач.

```
for (String task : tasksArray) {
    // Удаляем фигурные скобки
    task = task.replace("{", "").replace("}", "").trim();

    // Извлекаем описание задачи
    String description = extractField(task, "description");
    if (description != null) {
        taskListModel.addElement("Задача " + taskNumber++ + ": " +
description);
    }
}
```

- Для каждой задачи удаляет фигурные скобки и извлекает поле "description" с помощью метода `extractField()`. Если описание найдено, добавляет его в модель списка с нумерацией.

10. Метод `extractField`:

```
private String extractField(String task, String fieldName) {
    String[] fields = task.split(",");
    for (String field : fields) {
```

```
        String[] keyValue = field.split(":");
        if (keyValue.length == 2) {
            String key = keyValue[0].trim().replace("\"", ""); // Убираем
кавычки
            String value = keyValue[1].trim().replace("\"", ""); // Убираем
кавычки
            if (key.equals(fieldName)) {
                return value; // Возвращаем значение поля
            }
        }
    }
    return null; // Если поле не найдено
}
```

- Метод принимает строку задачи и название поля, ищет соответствующее значение и возвращает его. Если поле не найдено, возвращает `null`.

Результаты работы программы представлены ниже:

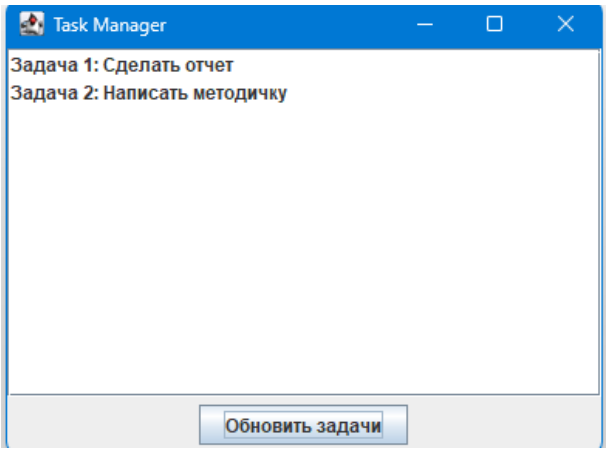


Рис 2 – Список задач в приложении

Рис. 3 – Список задач в браузере

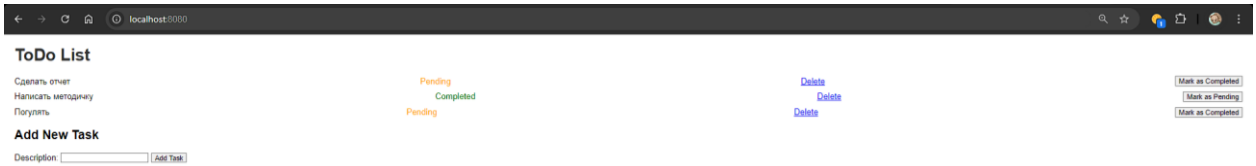


Рис. 4 – Добавляем задачу

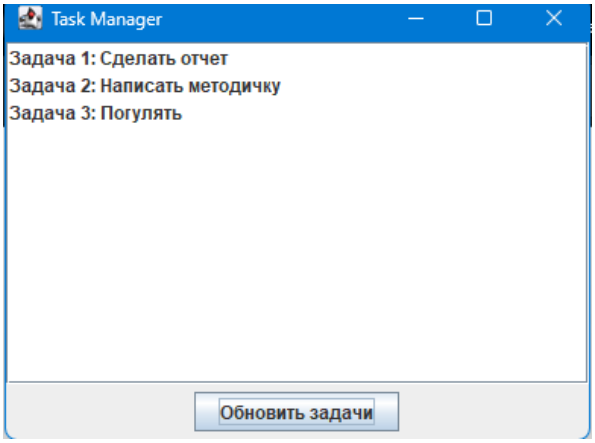


Рис.5 - Обновленный список задач

Код всех классов приведен в приложении.

Задание:

1. Изучить теоретический материал;

2. Написать приложение следуя примеру;

3. Выполнить дополнительное задание;

4. Сделать отчет.

Дополнительное задание.

- Реализуйте вывод дополнительных данных из БД, которые вы добавили в прошлой работе;
- Реализуйте функционал удаления задач в БД из клиентского приложения (право на удаление должно быть только у определенной роли);
- Реализуйте функционал добавления задач в БД из клиентского приложения;
- Реализуйте функционал сортировки задач;
- Улучшите пользовательский интерфейс.

Приложение 1.
SecurityConfig.java

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
```

```
org.springframework.security.config.annotation.method.configuration.EnableGlo
balMethodSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Bean
    public InMemoryUserDetailsManager userDetailsService() {
        UserDetails user = User.withUsername("user")
                .password("{noop}password") // Убедитесь, что вы используете
{noop} для не захешированного пароля
                .roles("USER")
                .build();

        UserDetails moderator = User.withUsername("moderator")
                .password("{noop}password")
                .roles("MODERATOR")
                .build();

        return new InMemoryUserDetailsManager(user, moderator);
    }

}
```

Приложение 2.
Task.java

```
mport jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String description;

    private boolean completed;

    // Конструктор по умолчанию
    public Task() {}

    // Конструктор с параметрами
    public Task(String description, boolean completed) {
        this.description = description;
        this.completed = completed;
    }

    // Getter and Setter for id
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    // Getter and Setter for description
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    // Getter and Setter for completed
    public boolean isCompleted() {
        return completed;
    }

    public void setCompleted(boolean completed) {
        this.completed = completed;
    }
}
```

Приложение 3.

TaskController.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping("/")
public class TaskController {
    @Autowired
    private TaskRepository taskRepository;

    // Веб-методы
    @GetMapping("/")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR')") // Доступ для
пользователей и модераторов
    public String index(Model model) {
        List<Task> tasks = taskRepository.findAll();
        model.addAttribute("tasks", tasks);
        return "index";
    }

    @PostMapping("/addTask")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR')") // Доступ для
пользователей и модераторов
    public String addTask(@ModelAttribute Task task) {
        taskRepository.save(task);
        return "redirect:/";
    }

    @PreAuthorize("hasRole('MODERATOR')") // Только модераторы могут удалять
задачи
    @GetMapping("/deleteTask/{id}")
    public String deleteTask(@PathVariable Long id) {
        taskRepository.deleteById(id);
        return "redirect:/";
    }

    @PreAuthorize("hasRole('MODERATOR')") // Только модераторы могут
обновлять задачи
    @PostMapping("/updateTask/{id}")
    public String updateTask(@PathVariable Long id, @RequestParam boolean
completed) {
        Task task = taskRepository.findById(id).orElseThrow();
        task.setCompleted(completed);
        taskRepository.save(task);
        return "redirect:/";
    }

    // REST-методы
    @GetMapping("/api")
    @PreAuthorize("hasRole('USER') or hasRole('MODERATOR')") // Доступ для
пользователей и модераторов
    public ResponseEntity<List<Task>> getAllTasks() {
```

```
        List<Task> tasks = taskRepository.findAll();
        return ResponseEntity.ok(tasks);
    }
    }
```

Приложение 4.

index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
```

```
    <title>ToDo Tracker</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        h1 {
            color: #333;
        }
        ul {
            list-style-type: none;
            padding: 0;
        }
        li {
            margin: 10px 0;
            display: flex;
            justify-content: space-between;
            align-items: center;
        }
        .completed {
            color: green;
        }
        .pending {
            color: orange;
        }
        form {
            display: inline;
        }
    </style>
</head>
<body>
<h1>ToDo List</h1>
<ul>
    <!-- Отображение задач -->
    <li th:each="task : ${tasks}">
        <span th:text="${task.description}"></span> <!-- Описание задачи -->
        <span th:class="${task.completed} ? 'completed' : 'pending'"
              th:text="${task.completed ? 'Completed' : 'Pending'}"></span>
<!-- Статус задачи -->

        <!-- Ссылка для удаления задачи -->
        <a th:href="@{/deleteTask/{id}(id=${task.id})}">Delete</a>

        <!-- Форма для обновления статуса задачи -->
        <form th:action="@{/updateTask/{id}(id=${task.id})}" method="post">
            <input type="hidden" name="completed" th:value="true"
th:if="${!task.completed}">
            <input type="hidden" name="completed" th:value="false"
th:if="${task.completed}">
            <button type="submit" th:text="${task.completed ? 'Mark as
Pending' : 'Mark as Completed'}"></button>
        </form>
    </li>
</ul>

<h2>Add New Task</h2>
<!-- Форма для добавления новой задачи -->
<form th:action="@{/addTask}" method="post">
    <label for="description">Description:</label>
    <input type="text" id="description" name="description" required>
    <button type="submit">Add Task</button>
</form>
</body>
</html>
```

Приложение 5.

LoginForm.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.util.Base64;

public class LoginForm extends JFrame {

    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;

    public LoginForm() {
        // UI компоненты
        setTitle("Login");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(3, 2));

        // Поля ввода для логина
        usernameField = new JTextField();
        passwordField = new JPasswordField();
        loginButton = new JButton("Login");

        add(new JLabel("Username:"));
        add(usernameField);
        add(new JLabel("Password:"));
        add(passwordField);
        add(new JPanel());  // Пустое место
        add(loginButton);

        // Обработчик авторизации
        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Попытка авторизации
                String username = usernameField.getText();
                String password = new String(passwordField.getPassword());
                String encodedAuth =
Base64.getEncoder().encodeToString((username + ":" + password).getBytes());

                // Если авторизация успешна, открываем главную форму
                TaskManagerForm taskManagerForm = new
TaskManagerForm(encodedAuth);
                taskManagerForm.setVisible(true);
                dispose();  // Закрываем форму авторизации
                System.out.println("Encoded Auth: " + encodedAuth);
            }
        });
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new LoginForm().setVisible(true);
        });
    }
}
```

Приложение 6.

TaskManagerForm.java

```
import javax.swing.*;
import java.awt.*;
import java.io.BufferedReader;
```

```
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

public class TaskManagerForm extends JFrame {

    private JList<String> taskList;
    private DefaultListModel<String> taskListModel;
    private String authHeader;

    public TaskManagerForm(String authHeader) {
        this.authHeader = authHeader;
        setTitle("Task Manager");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Список задач
        taskListModel = new DefaultListModel<>();
        taskList = new JList<>(taskListModel);
        loadTasks(); // Загрузка задач из API
        add(new JScrollPane(taskList), BorderLayout.CENTER);

        // Панель для кнопок
        JPanel buttonPanel = new JPanel();
        JButton refreshButton = new JButton("Обновить задачи");
        buttonPanel.add(refreshButton);
        add(buttonPanel, BorderLayout.SOUTH);

        // Обработчик события для кнопки обновления
        refreshButton.addActionListener(e -> loadTasks());
    }

    private void loadTasks() {
        try {
            URL url = new URL("http://localhost:8080/api");
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Authorization", "Basic " + authHeader);

            int responseCode = conn.getResponseCode();
            System.out.println("Response Code: " + responseCode);

            if (responseCode == 200) {
                BufferedReader in = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
                StringBuilder jsonResponse = new StringBuilder();
                String inputLine;

                while ((inputLine = in.readLine()) != null) {
                    jsonResponse.append(inputLine);
                }
                in.close();

                // Парсинг JSON вручную
                parseJson(jsonResponse.toString());
            } else {
                JOptionPane.showMessageDialog(this, "Ошибка загрузки задач");
            }

        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }

    private void parseJson(String json) {
        // Удаляем начальный и конечный квадратные скобки
        json = json.trim();
        if (json.startsWith("[") && json.endsWith("]")) {
            json = json.substring(1, json.length() - 1);
        }

        // Разделяем задачи по "},{" и обрабатываем каждую задачу
        String[] tasksArray = json.split("\\},\\{");
        int taskNumber = 1; // Счетчик задач

        // Очищаем список перед добавлением новых задач
        taskListModel.clear();

        for (String task : tasksArray) {
            // Удаляем фигурные скобки
            task = task.replace("{", "").replace("}", "").trim();

            // Извлекаем описание задачи
            String description = extractField(task, "description");
            if (description != null) {
                taskListModel.addElement("Задача " + taskNumber++ + ": " +
description);
            }
        }
    }

    private String extractField(String task, String fieldName) {
        String[] fields = task.split(",");
        for (String field : fields) {
            String[] keyValue = field.split(":");
            if (keyValue.length == 2) {
                String key = keyValue[0].trim().replace("\"", ""); // Убираем
кавычки
                String value = keyValue[1].trim().replace("\"", ""); //
Убираем кавычки
                if (key.equals(fieldName)) {
                    return value; // Возвращаем значение поля
                }
            }
        }
        return null; // Если поле не найдено
    }
}
```