

Диагностика уровня школьной тревожности

Введение

Документ посвящен приложению "Диагностика уровня школьной тревожности", разработанному с целью исследования уровня и характера тревожности у детей младшего и среднего школьного возраста.

Приложение представляет собой тест, состоящий из 58 вопросов, на каждый из которых требуется ответить "да" или "нет". Он предоставляет возможность проанализировать психологическое состояние школьников в контексте тревожности.

Документ включает в себя информацию о используемых технологиях (Java 17, Gson, Lombok, Javax Mail) и инструкции по началу работы с приложением. Особое внимание уделяется настройке отправки писем по электронной почте, включая установку почтового сервера и изменение конфигурационного файла с данными для отправки писем.

Цель приложения состоит в изучении уровня и характера тревожности, связанный со школьной у детей младшего и среднего школьного возраста. Тест состоит из 58 вопросов. На каждый вопрос требуется однозначно ответить "да" или "нет".

Содержание

- Технологии
- Начало работы

Технологии

- Java 17
- Gson
- Lombok
- Javax Mail

Начало работы

Для начала склонируйте репозиторий командой *git clone https://github.com/funcid/phillips-anxiety-test-desktop.git*. После клонирования репозитория, у вас появится локальная копия проекта на вашем компьютере. Теперь вы можете приступить к разработке или использованию приложения.

Настройка отправки писем по электронной почте

Данный файл предоставляет инструкции по настройке отправки писем по электронной почте в приложении. Вам потребуются следующие данные:

- mail.host
- mail.username
- mail.password
- mail.port

Установка почтового сервера

Перед тем, как приступить к настройке отправки писем, убедитесь что у вас установлен почтовый сервер. В данном примере мы будем использовать smtp.gmail.com в качестве почтового сервера

Изменение данных в конфигурационном файле

Для изменения данных, связанных с отправкой писем, вам потребуется перейти в конфигурационный файл mail.properties. Пример файла *mail.properties*:

```
mail.host=smtp.gmail.com
mail.username=some@gmail.com
mail.password=somepassword
mail.port=465
```

Найдите строки, содержащие данные для отправки писем, и замените их значения на ваши данные.

mail.host=smtp.gmail.com

mail.username=ВАША_ПОЧТА@gmail.com

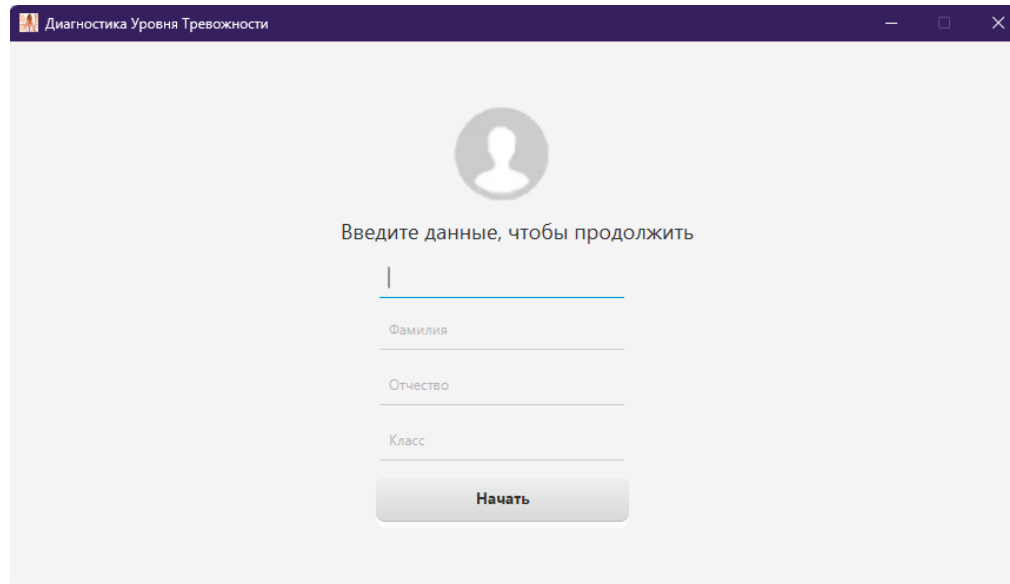
mail.password=ВАШ_ПАРОЛЬ

mail.port=465

Примечание

Убедитесь, что введены правильные данные для mail.username и mail.password. Это данные для вашей электронной почты, с которой будет осуществляться отправка писем.

На рисунке 1 представлено главное окно приложения, на котором вы можете обнаружить четыре поля для ввода данных: имя, фамилия, отчество и класс. Кроме того, на этом же изображении расположена кнопка "Начать", которая позволяет вам начать тестирование.



Диагностика Уровня Тревожности

Введите данные, чтобы продолжить

Имя

Фамилия

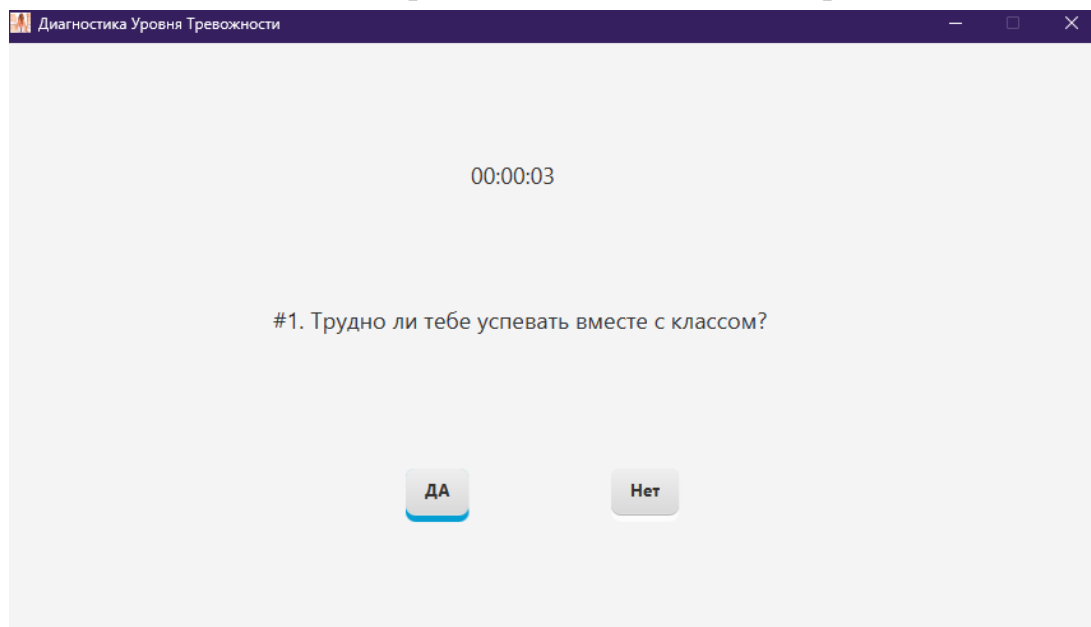
Отчество

Класс

Начать

Рисунок 1 - Главное окно.

На рисунке 2 представлено окно с вопросами, где вы можете увидеть таймер, отображающий время, которое пользователь затрачивает на прохождение теста. Также на изображении присутствуют две кнопки для ответов "да" и "нет", сопровождаемые текстом вопроса.



Диагностика Уровня Тревожности

00:00:03

#1. Трудно ли тебе успевать вместе с классом?

ДА Нет

Рисунок 2 - Окно с вопросами.

На рисунке 3 представлено финальное окно с двумя кнопками: "Отправить" и "Сохранить". Кнопка "Отправить" позволяет пользователю отправить сообщение с результатом теста на указанную почту, в то время как кнопка "Сохранить" сохраняет результат прохождения теста в формате Word документа. Также на изображении присутствует поле для ввода адреса электронной почты, на которую будет отправлено письмо, а также отображается сам результат прохождения теста.

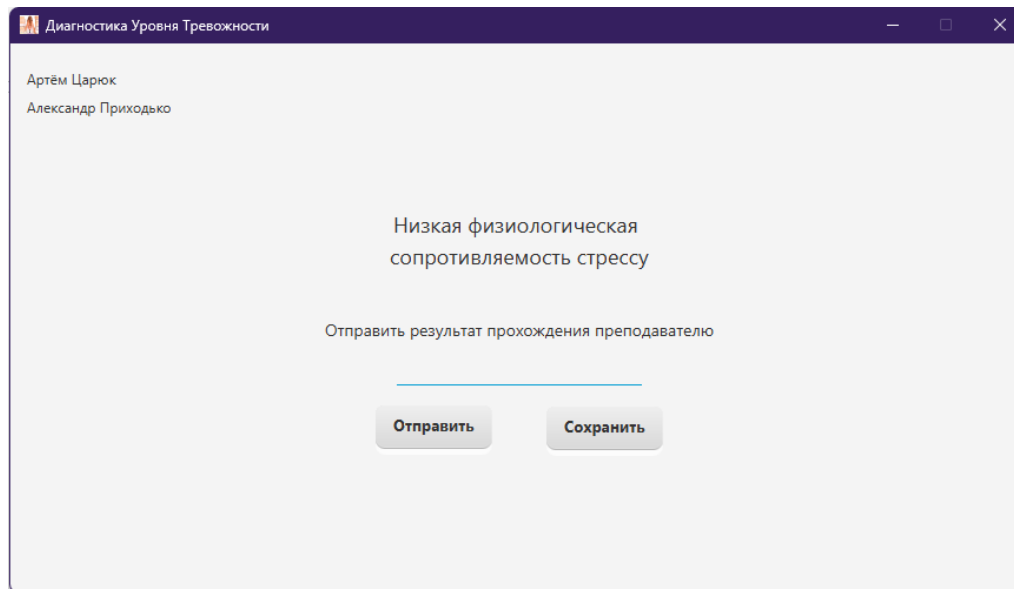


Рисунок 3 - Финальное окно.

На рисунке 4 представлен класс PrimaryStage. Это класс, который, представляет главное окно в JavaFX приложении. В данном случае, этот класс использует javafx.stage.Stage для создания главного окна и управления им.

Конструктор PrimaryStage: Принимает объект Stage в качестве параметра и создает запись (record) с именем PrimaryStage, которая оборачивает переданный Stage.

Метод setScene(Scene scene): Устанавливает переданный объект Scene в текущее окно (stage). Пытается загрузить класс javafx.application.Application. Если этот класс не найден, то выводит сообщение об ошибке, связанной с отсутствием библиотек JavaFX, и завершает программу. Устанавливает обработчик события закрытия окна, который вызывает Platform.exit(). Устанавливает размер окна в соответствии с размером сцены. Устанавливает заголовок окна. Запрещает изменение размеров окна. Устанавливает иконку окна из файла "logo.jpg". Вызывает метод showScene(scene), который отображает сцену в окне.

Метод showScene(Scene scene): Устанавливает переданный объект Scene в текущее окно (stage). Отображает окно.

Метод showAlert(Alert.AlertType alertType, String contextText, String headerText): Создает и отображает диалоговое окно (Alert) с заданными параметрами. В зависимости от типа алерта (AlertType) устанавливает соответствующий заголовок ("Ошибка!" или "Успешно!").

```

package me.reidj.anxietydiagnostic.controller;

import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.util.Objects;

public class ReidjJava
public record PrimaryStage(Stage stage) {

    1 usage 1 ReidjJava
    public void setScene(Scene scene) {
        try {
            Class.forName("javafx.application.Application");
        } catch (ClassNotFoundException e) {
            showAlert(
                Alert.AlertType.ERROR,
                contextText: "Библиотеки JavaFX не найдены. Установите Java с поддержкой JavaFX, например OracleJRE 17 с официального сайта.",
                headerText: "Ошибка"
            );
            System.exit(status: 1);
        }
        stage.setOnHidden(event -> Platform.exit());
        stage.sizeToScene();
        stage.setTitle("Диагностика Уровня Тревожности");
        stage.setResizable(false);
        stage.getIcons().add(new Image(Objects.requireNonNull(getClass().getResourceAsStream(name: "/images/Logo.jpg"))));
        showScene(scene);
    }

    3 usages 1 ReidjJava
    public void showScene(Scene scene) {
        stage.setScene(scene);
        stage.show();
    }

    4 usages 1 ReidjJava
    public void showAlert(Alert.AlertType alertType, String contextText, String headerText) {
        Alert alert = new Alert(alertType, contextText);
        alert.setTitle(alertType == Alert.AlertType.ERROR ? "Ошибка!" : "Успешно!");
        alert.setHeaderText(headerText);
        alert.show();
    }
}

```

Рисунок 4 - Класс PrimaryStage.

На рисунке 5 представлен класс MailSenderService. Этот код представляет собой простой класс для отправки электронных писем через протокол SMTP (Simple Mail Transfer Protocol) с использованием JavaMail API.

```
1 usage  ▲ reidj.java
public class MailSenderService {

    1 usage
    private static final String RESOURCES_PATH = "src/main/resources/mail.properties";

    5 usages
    private final Properties systemProperties = System.getProperties();
    5 usages
    private final Properties properties = loadFile(RESOURCES_PATH);

    1 usage  ▲ reidj.java
    public MailSenderService() {
        systemProperties.setProperty("mail.smtp.host", properties.getProperty("mail.host"));
        systemProperties.setProperty("mail.smtp.port", properties.getProperty("mail.port"));
        systemProperties.setProperty("mail.smtp.ssl.enable", "true");
        systemProperties.setProperty("mail.smtp.auth", "true");
    }

    1 usage  ▲ reidj.java
    public void send(String to, String subject, String text) {
        Session session = Session.getInstance(
            systemProperties,
            getPasswordAuthentication() → {
                return new PasswordAuthentication(properties.getProperty("mail.username"), properties.getProperty("mail.password"));
            }
        );
        generateMessage(to, session, subject, text);
    }

    1 usage  ▲ reidj.java
    private void generateMessage(String to, Session session, String subject, String text) {
        try {
            MimeMessage message = new MimeMessage(session);
            message.setFrom(new InternetAddress(properties.getProperty("mail.username")));
            message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
            message.setSubject("Диагностика уровня тревожности | " + subject);
            message.setText(text);
            Transport.send(message);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    1 usage  ▲ reidj.java
    public static Properties loadFile(String path) {
```

Рисунок 5 - Класс MailSenderService.

На рисунке 6 представлен класс QuestionController. Этот код представляет собой контроллер для вопросов в графическом интерфейсе приложения (видимо, JavaFX), и, используется в контексте тестирования пользователя.

```
public class QuestionController extends AbstractScene {

    1 usage
    private static final String JSON_FILE_PATH = "/questions.json";

    @FXML
    private Label questionLabel;
    @FXML
    private Label timeLabel;
    3 usages
    private Iterator<Question> questionIterator;
    1 usage
    private final Gson gson = new Gson();
    8 usages
    private Timeline timeline;

    public QuestionController() { super(path: "questions/questionsScene.fxml"); }

    @FXML
    private void initialize() {
        QuestionList questions = loadQuestionsFromJson();

        if (questions != null)
            questionIterator = questions.questions.iterator();

        startTimer();

        displayNextQuestion();
    }

    2 usages
    private Question displayNextQuestion() {
        if (questionIterator.hasNext()) {
            Question currentQuestion = questionIterator.next();

            questionLabel.setText("#" + currentQuestion.index + ". " + currentQuestion.text);

            return currentQuestion;
        } else {
            App.getApp().getPrimaryStage().showScene(App.getApp().getResultController().getScene());
            stopTimer();
        }
        return null;
    }

    @FXML
    void onNoHandler() { extracted(answer: "Нет"); }

    @FXML
    void onYesHandler() { extracted(answer: "Да"); }
```

```

private void extracted(String answer) {
    questionLabel.setText("");
    App.getApp().App
        .getUser() User
        .getQuestions() Map<Question, String>
        .put(displayNextQuestion(), answer);
}

1 usage
private QuestionList loadQuestionsFromJson() {
    try (InputStream inputStream = getClass().getResourceAsStream(JSON_FILE_PATH);
        InputStreamReader reader = new InputStreamReader(inputStream, StandardCharsets.UTF_8)) {
        return gson.fromJson(reader, QuestionList.class);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

// Метод для начала работы таймера
1 usage
private void startTimer() {
    if (timeline == null || timeline.getStatus() == Animation.Status.STOPPED) {
        timeline = new Timeline(new KeyFrame(Duration.seconds(5), 1), event -> Platform.runLater(this::updateTimer));
        timeline.setCycleCount(Animation.INDEFINITE);
        timeline.play();
    }
}

1 usage
private void updateTimer() {
    User user = App.getApp().getUser();
    user.setSeconds(user.getSeconds() + 1);
    if (user.getSeconds() == 60) {
        user.setSeconds(0);
        user.setMinutes(user.getMinutes() + 1);
        if (user.getMinutes() == 60) {
            user.setMinutes(0);
            user.setHour(user.getHour() + 1);
        }
    }
    timeLabel.setText(formatTime());
}

1 usage
private void stopTimer() {
    if (timeline != null && timeline.getStatus() == Animation.Status.RUNNING) {
        timeline.stop();
    }
}

2 usages
public String formatTime() {
    User user = App.getApp().getUser();
    return String.format("%02d:%02d:%02d", user.getHour(), user.getMinutes(), user.getSeconds());
}

```

Рисунок 6 - Класс QuestionController.