

КУРСОВОЙ ПРОЕКТ

По дисциплине: МДК 01.01 Разработка программных модулей

Тема: Разработка системы классов для приложения «Кадровое агентство» на Go

Специальность 09.02.07 «Информационные системы и программирование»

|  |  |  |
| --- | --- | --- |
| Выполнил студент(ка) группы 302ИС-22 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | М.Д.Трусов |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Л.Б.Гусятинер |

Москва 2024

**Оглавление**

Введение3

1.Теоретическая часть 3

1.1**Описание предметной области** 4

1.2. Описание существующих разработок 4

1.2.1.Workday HCM5

1.2.2. Spencer Stuart 6

2.Проектная часть7

2.1.Диаграмма прецедентов8

**2.2.** **Выбор инструментов**9

**2.3. Проектирование сценария**10

**2.4. Диаграмма классов**12

**2.5Описание главного модуля**14

**2.6 Описание тестовых наборов модулей**20

**2.7 Отладка** 26

3. Эксплуатационная часть 26

3.1. Руководство оператора26

**Заключение**34

Список литературы и интернет-источников34

Приложение 1.Код программы35

Введение

Целью данного курсового проекта является разработка приложения для кадрового агентства, предназначенного для упрощения и автоматизации работы с кандидатами и вакансиями. Эта тема является актуальной, поскольку современный рынок труда каждый день расширяется и требует новых решений

Разработка данного приложения позволит существенно облегчить работу сотрудников кадрового агентства. Оно обеспечит удобный интерфейс для ведения базы данных соискателей и вакансий, а также для поиска подходящих кандидатов под конкретные запросы. Интуитивно понятный интерфейс приложения сделает его доступным даже для пользователей без специальной подготовки.

В первой части будет рассмотрена предметная область данной темы, а

также несколько продуктов по данной теме.

Во второй части будут рассмотрены инструменты и модули, которые

были разработаны, структура программной части и листинги ключевых частей

программных модулей.

В третьей части будет рассмотрено руководство для пользователей.

В заключительной части будет приведен To-do лист с планами по

доработки программы, а также сделаны общие выводы о получившемся

проекте.

1.1. Описание предметной области

Кадровое агентство – это организация, основная задача которой заключается в подборе специалистов для различных компаний и предприятий, а также в помощи соискателям в трудоустройстве. Эта сфера является важным элементом современного рынка труда, поскольку эффективная работа кадрового агентства способствует развитию бизнеса и снижению уровня безработицы.

Работа кадрового агентства связана с необходимостью оперативной обработки значительных объемов информации, включая резюме кандидатов, запросы работодателей, данные об открытых вакансиях и результатах собеседований. Автоматизация этих процессов позволяет повысить качество обслуживания клиентов и соискателей, а также снизить затраты времени на выполнение рутинных задач.

1.2. Описание существующих разработок

Современное кадровое агентство – это сложная структура, где требуется точность и оперативность во всех аспектах деятельности. Своевременный подбор персонала, учет изменений в базах данных, взаимодействие с клиентами и кандидатами – все это критически важно для успешного функционирования агентства. Неправильная организация или отсутствие автоматизированных систем могут привести к значительным временным и финансовым потерям.

На рынке существует несколько видов кадровых агентств, которые отличаются своей специализацией:

Агентства, деятельность которых нацелена на подбор высшего руководящего состава и эксклюзивных специалистов (Executive Search Agency) и компании по подбору персонала среднего и низшего уровней (Recruiting Agency).

К первому виду следует отнести компании, которые предлагают клиентам широкий спектр услуг, характеризующихся развитой структурой и имеющих известное имя на рынке кадровых услуг не только региона, но и России в целом. Эти компании владеют профессиональными специалистами, которые способны заниматься подбором эксклюзивного персонала.

В последнее время динамично развивается ряд молодых агентств, которые успели занять рыночную нишу в сфере кадровых услуг и развиться на волне общего развития рынка. Эти компании занимаются поиском эксклюзивных специалистов и специалистов низшего уровня. Среди их услуг можно выделить не только подбор персонала, но и такие функции, как оценка и обучение персонала, HR-консалтинг, лизинг персонала сферы рекрутмента и IT, мониторинг рынка труда, проведение тренингов как для субъектов хозяйствования, так и для людей, ищущих работу.

**1.2.1.Workday HCM**

Workday HCM — это облачная платформа управления человеческим капиталом, предназначенная для автоматизации HR-процессов, анализа данных о сотрудниках и повышения эффективности управления персоналом. Решение активно используется крупными компаниями по всему миру благодаря своей гибкости, интуитивно понятному интерфейсу и широкому функционалу.

Преимущества Workday HCM

Единая платформа: Все HR-данные хранятся в одной системе, что упрощает управление и анализ.

Интуитивный интерфейс: Простота использования даже для пользователей без технической подготовки.

Масштабируемость: Подходит как для малого бизнеса, так и для крупных корпораций с тысячами сотрудников.

Гибкость: Возможность настройки под нужды компании.

Безопасность: Высокий уровень защиты данных благодаря современным технологиям шифрования и управления доступом.

Интеграция: Возможность соединения с другими корпоративными системами, такими как ERP или CRM.

**1.2.2. Spencer Stuart**

Spencer Stuart — это международное агентство, специализирующееся на подборе высшего руководящего состава и эксклюзивных специалистов для крупных и международных компаний.

**Преимущества Spencer Stuart:**

**Глубокая отраслевая экспертиза**: Компания имеет опыт работы в самых различных отраслях, что позволяет предлагать решения для каждой конкретной сферы. Это делает их подход более персонализированным и эффективным.

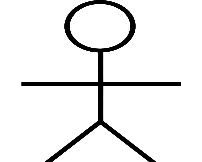
**Использование инновационных методов:** Spencer Stuart активно использует современные аналитические инструменты и подходы для оценки кандидатов, что позволяет значительно повысить точность подбора.

**Конфиденциальность и надежность:** Компания обеспечивает высокие стандарты конфиденциальности, что особенно важно при подборе топ-менеджеров и кандидатов для ключевых позиций

**Гибкость подхода**: Spencer Stuart использует индивидуальный подход к каждому клиенту и предлагает решения, которые максимально соответствуют потребностям бизнеса.

2.Проектная часть

2.1.Диаграмма прецедентов



запуск программы

2.2. Выбор инструментов

При выборе инструментов было проведено сравнение по критериям, представленных в таблице 1. Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1. Критерии выбора инструмента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| критерий | Участие в корпоративном проекте | Простота сопровождения | Наличие библиоте | Наличие документации на русском языке | Скорость разработки |
| Важность критерия | очень высокая | высокая | очень высокая | очень высокая | высокая |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий.

Таблица 2. Оценка языков программирования.

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий Язык программирования | Go | C++ | Python |
| Участие в корпоративном проекте | 9 | 10 | 5 |
| Простота сопровождения | 10 | 7 | 10 |
| Наличие библиотек | 10 | 7 | 8 |
| Наличие документации на русском языке | 1 | 8 | 6 |
| Скорость разработки | 10 | 6 | 9 |
| Итого баллов | 40 | 38 | 38 |

По результатам сравнения был выбран язык программирования Go

2.3. Проектирование сценария

В данном разделе приведен сценарий использования программы

Выберите действие

запуск программы

Авторизоваться

Добавить кандидата

Добавить компанию

Введите имя пользователя

Зарегистрироваться

Введите имя пользователя

Введите ФИО кандидата

Введите название компании

Введите пароль

Введите возраст кандидата

Компания добавлена!

Авторизация успешна!

Произошла ошибка: пользователь не найден

Введите пароль

Регистрация успешна!

Вакансия добавлена

требуемые навыки

Произошла ошибка

Ввод символа

Введите зарплату

требуемый опыт работы

Произошла ошибка

Ввод символа

Добавить вакансию

Введите ID компании

Введите название вакансии

Ошибка неверный ввод

Ввод символа

Кандидат добавлен!

Введите навыки кандидата

Введите email кандидата

неверный пароль

Завершение работы

Выход

Вывод всех вакансий

Показать все вакансии

Найденные вакансии

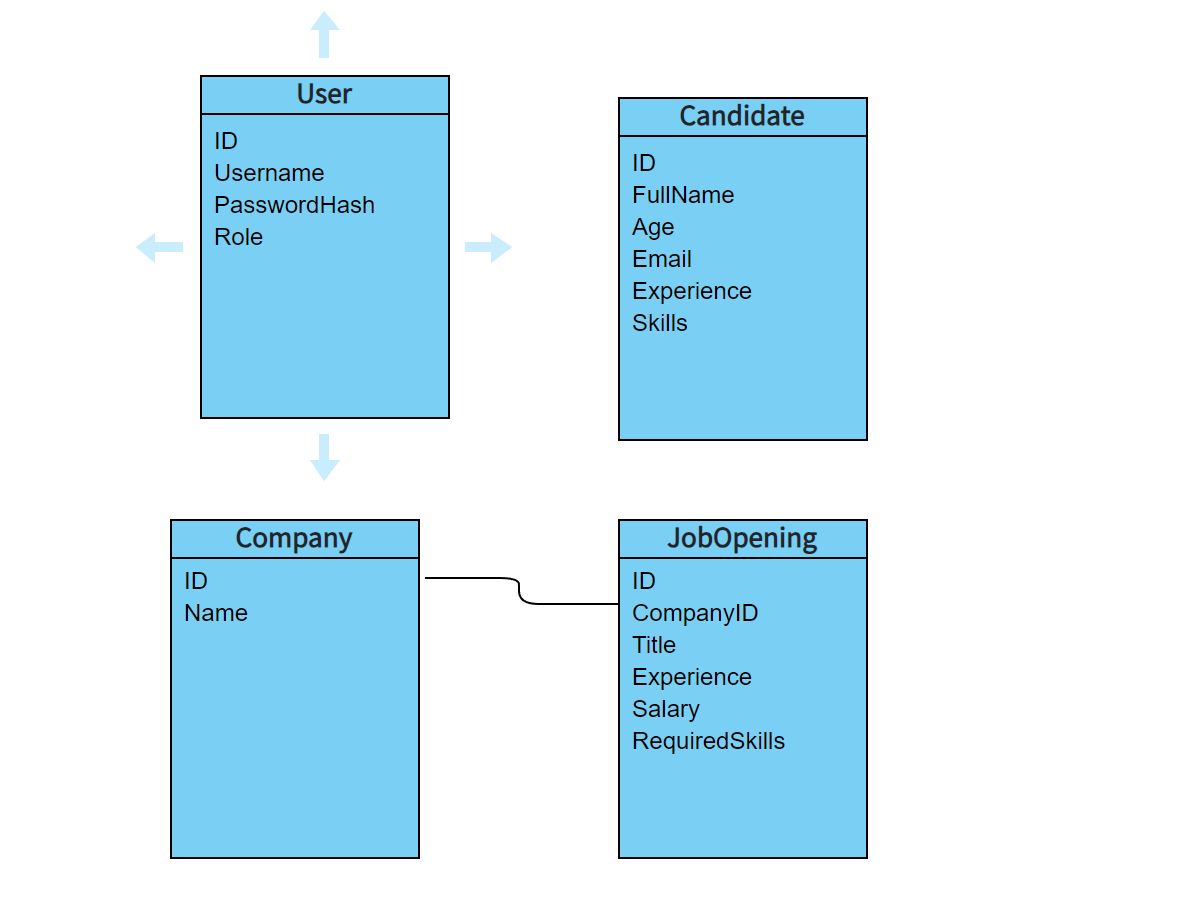
Найти вакансии по навыку

Найденные кандидаты

Найти кандидатов по навыку

2.4 Диаграмма классов

В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.



Класс User

Содержит публичные поля ID, Username, PasswordHash и Role.

Используется для хранения информации о пользователях.

Методы:

hashPassword() — хеширует пароль.

checkPasswordHash() — проверяет соответствие пароля хешу.

registerUser() — регистрирует нового пользователя.

loginUser() — авторизует пользователя.

Класс Company

Содержит публичные поля ID и Name.

Используется для хранения информации о компаниях.

Методы:

addCompany() — добавляет новую компанию.

Класс Candidate

Содержит публичные поля ID, FullName, Age, Email, Experience, Skills.

Используется для хранения информации о кандидатах.

Методы:

addCandidate() — добавляет информацию о новом кандидате.

findCandidatesBySkill() — ищет кандидатов по навыкам.

Класс JobOpening

Содержит публичные поля ID, CompanyID, Title, Experience, Salary, RequiredSkills.

Используется для хранения информации о вакансиях.

Методы:

addJobOpening() — добавляет новую вакансию.

listAllJobOpenings() — выводит список всех вакансий.

findJobOpeningsBySkill() — ищет вакансии по навыкам.

Класс DBControl

Содержит методы для взаимодействия с базой данных.

Методы:

createTables() — создает таблицы в базе данных, если они не существуют.

handleError() — обрабатывает ошибки базы данных.

Класс Application

Объединяет логику работы всех остальных классов.

Методы:

main() — запускает приложение, предоставляя интерфейс для взаимодействия с пользователем.

Методы взаимодействия:

getInput() — получает пользовательский ввод.

getIntInput() — получает числовой пользовательский ввод.

getFloatInput() — получает ввод с плавающей точкой.

getStringArrayInput() — получает массив строк.

Взаимодействие между классами

Классы User, Candidate, Company, и JobOpening используются для работы с соответствующими данными.

Класс DBControl обеспечивает работу с базой данных для всех остальных классов.

Класс Application предоставляет пользовательский интерфейс и вызывает методы остальных классов для выполнения задач.

2.5Описание главного модуля

в главный модуль импортируются библиотеки :database/sql , fmt, github.com/lib/pq , log

а так же пакеты других модулей: struct\_kursovaya/internal/handlers, struct\_kursovaya/pkg/utils

главный модуль является точкой входа в программу,в нем релизованы вызовы всех функций,открытие базы данных и логика меню,с помощью которого пользователь может взаимодействовать с программой

db, err := sql.Open("postgres", "user=postgres password=\*\*\*\*\*. dbname=KURSOVAYA sslmode=disable")

if err != nil {

log.Fatal(err)

}

defer db.Close()

err = createTables(db)

handleError(err)

if err != nil {

return

}

for {

fmt.Println("\nВыберите действие:")

fmt.Println("1. Зарегистрироваться")

fmt.Println("2. Авторизоваться")

fmt.Println("3. Добавить компанию")

fmt.Println("4. Добавить кандидата")

fmt.Println("5. Добавить вакансию")

fmt.Println("6. Найти кандидатов по навыку")

fmt.Println("7. Найти вакансии по навыку")

fmt.Println("8. Показать все вакансии")

fmt.Println("9. Выйти")

choice, err := getIntInput("Введите номер действия: ")

handleError(err)

if err != nil {

continue

}

switch choice {

case 1: // Регистрация пользователя

username := getInput("Введите имя пользователя: ")

password := getInput("Введите пароль: ")

err := registerUser(db, username, password)

handleError(err)

if err == nil {

fmt.Println("Регистрация успешна!")

}

case 2: // Авторизация пользователя

username := getInput("Введите имя пользователя: ")

password := getInput("Введите пароль: ")

userID, role, err := loginUser(db, username, password)

handleError(err)

if err == nil {

fmt.Printf("Авторизация успешна! ID пользователя: %d, Роль: %s\n", userID, role)

}

case 3: // Добавление компании

companyName := getInput("Введите название компании: ")

err := addCompany(db, companyName)

handleError(err)

if err == nil {

fmt.Println("Компания успешно добавлена!")

}

case 4: // Добавление кандидата

candidate := Candidate{}

candidate.FullName = getInput("Введите ФИО кандидата: ")

candidate.Age, err = getIntInput("Введите возраст кандидата: ")

handleError(err)

if err != nil {

continue

}

candidate.Email = getInput("Введите email кандидата: ")

candidate.Experience = getInput("Введите опыт работы кандидата: ")

candidate.Skills, err = getStringArrayInput("Введите навыки кандидата (через запятую): ")

handleError(err)

if err != nil {

continue

}

err = addCandidate(db, candidate)

handleError(err)

if err == nil {

fmt.Println("Кандидат успешно добавлен!")

}

case 5: // Добавление вакансии

jobOpening := JobOpening{}

jobOpening.Title = getInput("Введите название вакансии: ")

jobOpening.CompanyID, err = getIntInput("Введите ID компании: ")

handleError(err)

if err != nil {

continue

}

jobOpening.Experience = getInput("Введите требуемый опыт работы: ")

jobOpening.Salary, err = getFloatInput("Введите зарплату: ")

handleError(err)

if err != nil {

continue

}

jobOpening.RequiredSkills, err = getStringArrayInput("Введите требуемые навыки (через запятую): ")

handleError(err)

if err != nil {

continue

}

err = addJobOpening(db, jobOpening)

handleError(err)

if err == nil {

fmt.Println("Вакансия успешно добавлена!")

}

case 6: // Поиск кандидатов по навыку

skill := getInput("Введите навык для поиска кандидатов: ")

candidates, err := findCandidatesBySkill(db, skill)

handleError(err)

if err == nil {

fmt.Println("Найденные кандидаты:")

for \_, c := range candidates {

fmt.Printf("ID: %d, ФИО: %s, Навыки: %v\n", c.ID, c.FullName, c.Skills)

}

}

case 7: // Поиск вакансий по навыку

skill := getInput("Введите навык для поиска вакансий: ")

jobOpenings, err := findJobOpeningsBySkill(db, skill)

handleError(err)

if err == nil {

fmt.Println("Найденные вакансии:")

for \_, j := range jobOpenings {

fmt.Printf("ID: %d, Название: %s, Требуемые навыки: %v\n", j.ID, j.Title, j.RequiredSkills)

}

}

case 8: // Показать все вакансии

err := listAllJobOpenings(db)

handleError(err)

if err != nil {

fmt.Println("Ошибка при выводе вакансий:", err)

}

case 9: // Выход из программы

fmt.Println("Выход из программы.")

return

default: // Неверный выбор

fmt.Println("Неверный выбор действия. Попробуйте снова.")

}

}

}

2.6 Описание тестовых наборов модулей

В этом разделе будут продемонстрированы результаты тестирования «черного ящика».

Тест 1

Цель: Проверить корректность регистрации нового пользователя.

Тестовые данные:

Ввод имени пользователя: new\_user.

Ввод пароля: password123.

Ожидаемый результат: Сообщение "Регистрация успешна!".

Результат теста:



Тест 2

Цель: Убедиться, что пользователи могут корректно входить в систему.

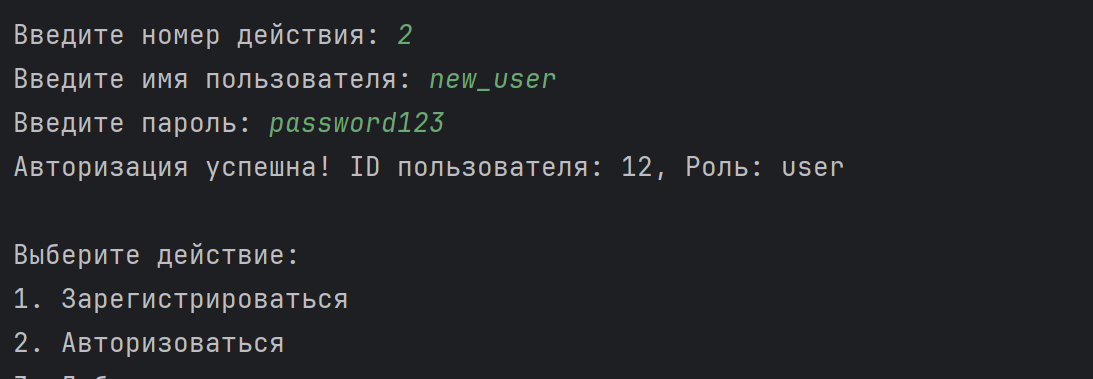
Тестовые данные:

Имя пользователя: new\_user.

Пароль: password123.

Ожидаемый результат: Сообщение "Авторизация успешна!" и вывод ID пользователя и его роли.

Результат теста



Тест 3

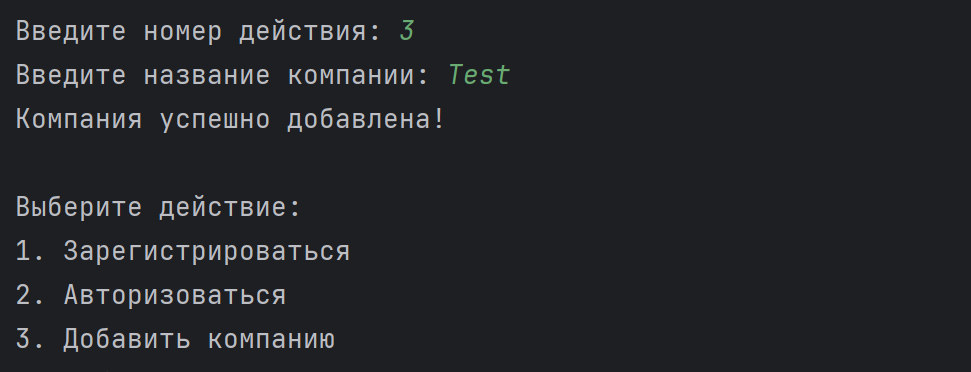
Цель: Проверить добавление новой компании в базу данных.

Тестовые данные:

Название компании:Test.

Ожидаемый результат: Сообщение "Компания успешно добавлена!".

Результаты теста



Тест 4

Цель: Проверить добавление нового кандидата.

Тестовые данные:

ФИО: Трусов Михаил Дмитриевич.

Возраст: 18.

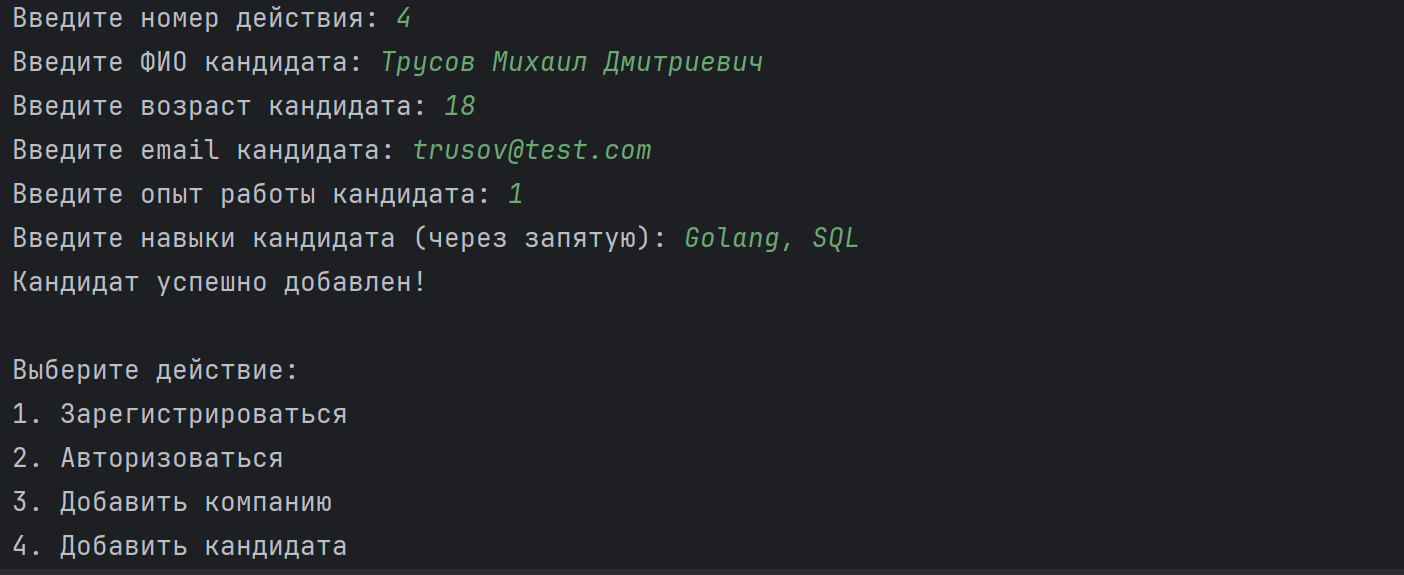
Email: trusov@test.com.

Опыт: 1 год.

Навыки: Golang, SQL.

Ожидаемый результат: Сообщение "Кандидат успешно добавлен!".

Результаты теста



Тест 5

Цель: Создать вакансию

Тестовые данные:

**Название вакансии**: "Backend Developer"

**ID компании**: 1

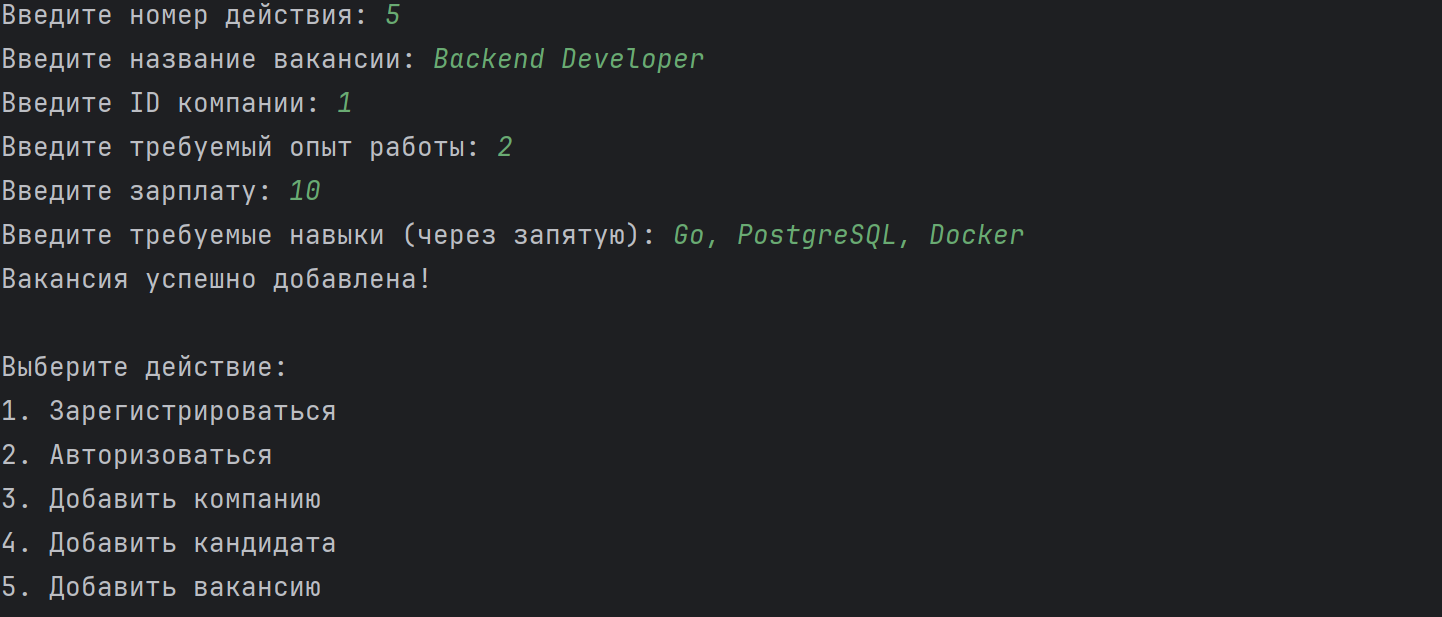
**Требуемый опыт работы**: "2"

**Зарплата**: 10

**Требуемые навыки**: ["Go", "PostgreSQL", "Docker"]

Ожидаемый результат: создание новой вакансии

Результаты теста



Тест 6

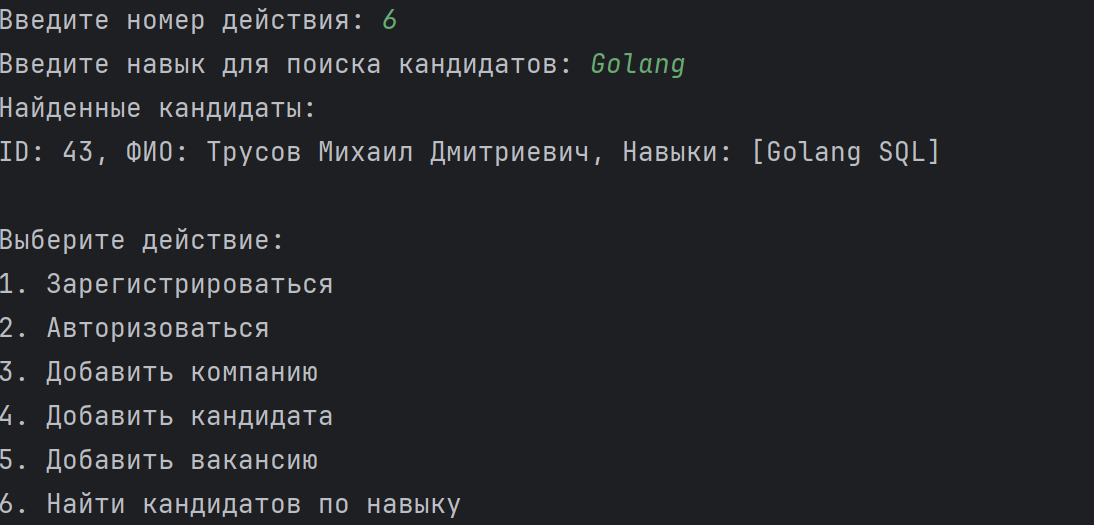
Цель: Проверить поиск кандидатов с указанным навыком.

Тестовые данные:

Навык: Golang.

Ожидаемый результат: Вывод списка кандидатов, обладающих этим навыком.

Результаты теста



Тест 7

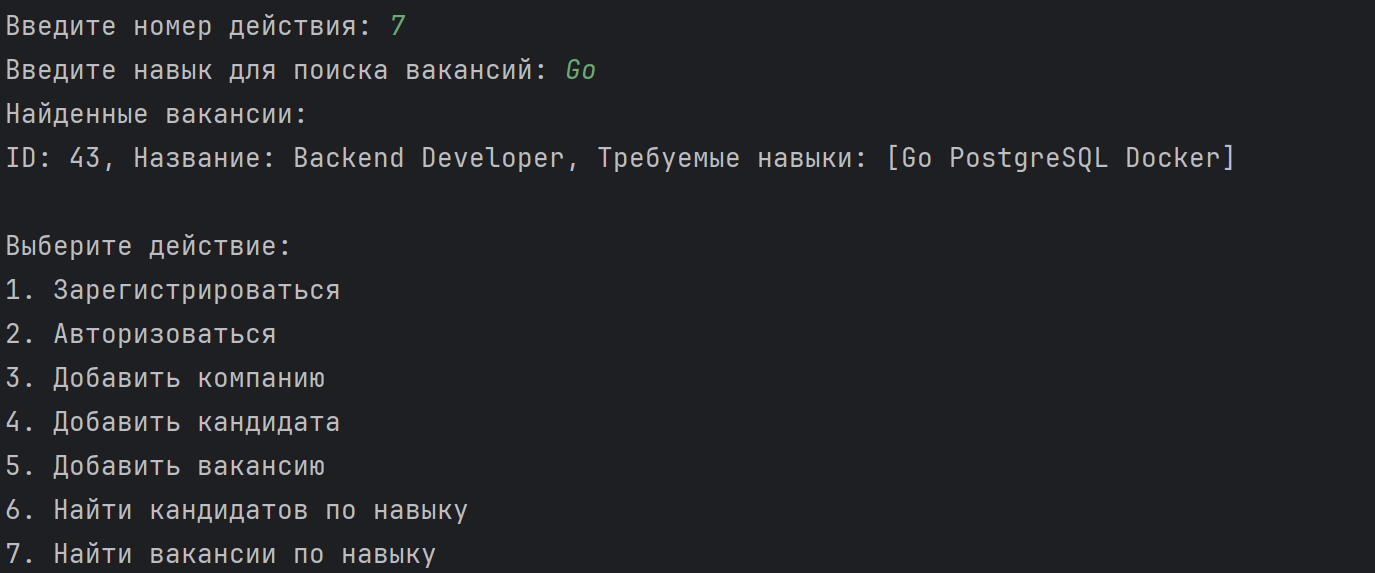
Цель: Проверить корректность поиска вакансий по указанному навыку.

Тестовые данные:

Навык: Go.

Ожидаемый результат: Вывод списка вакансий, требующих этот навык.

Результаты теста



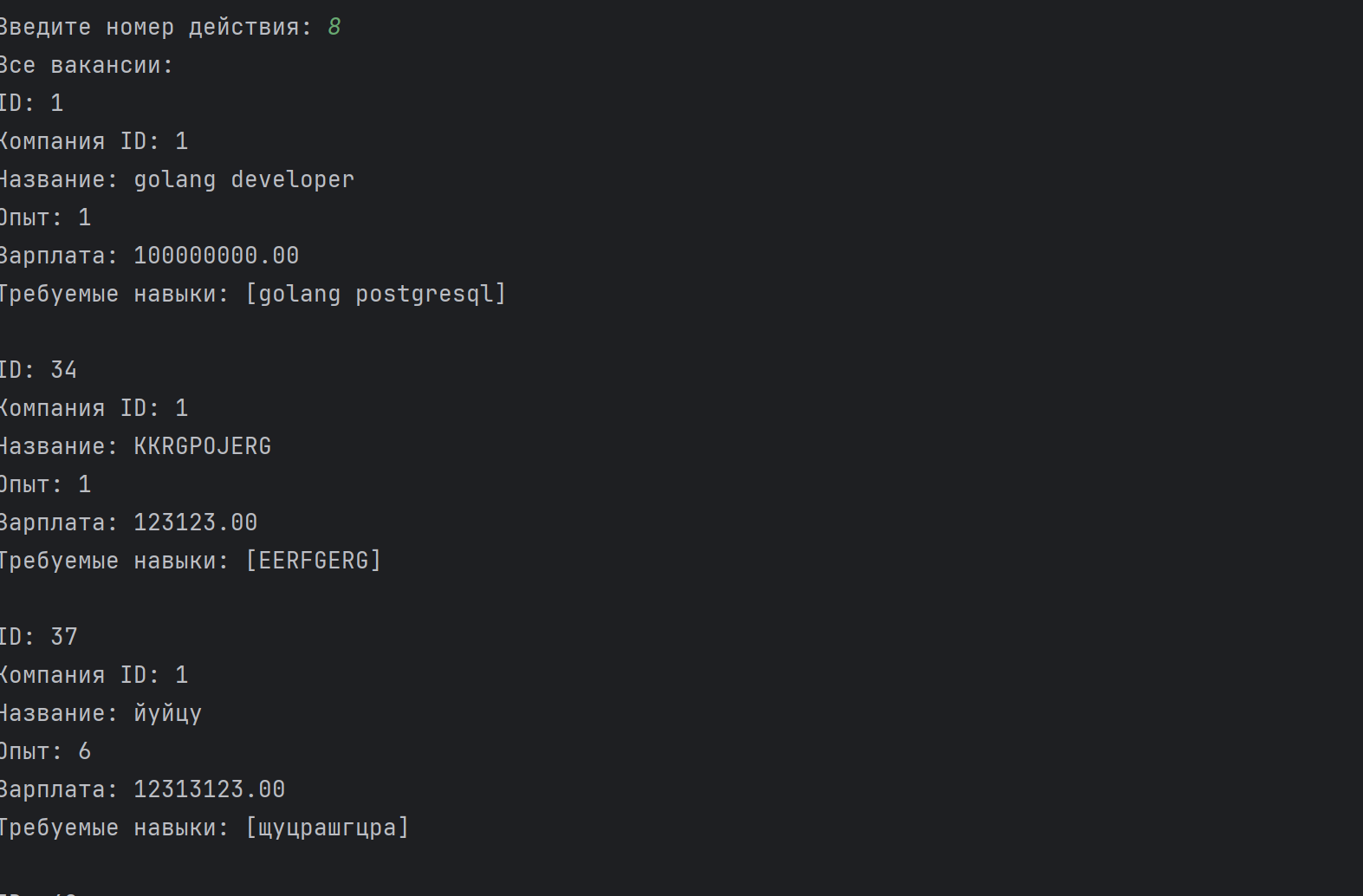
Тест 8

Цель: Убедиться, что программа корректно выводит все вакансии.

Тестовые данные:

Команда: 8.

Ожидаемый результат: Вывод списка всех доступных вакансий.



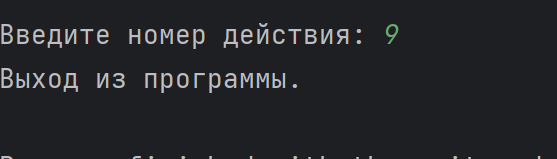
Тест 9

Цель: Убедиться, что программа корректно завершается

Тестовые данные:

Команда: 9.

Ожидаемый результат:Завершение работы программы

.

2.7 Отладка

В данной программе багов и ошибок не обнаружено.

3. Эксплуатационная часть

3.1. Руководство оператора

АННОТАЦИЯ

В данном программном документе приведено руководство оператора по применению и эксплуатации программы.

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации. В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.). В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды. Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 1), ГОСТ 19.103-77 2), ГОСТ 19.104-78\* 3), ГОСТ 19.105-78\* 4), ГОСТ 19.106-78\* 5), ГОСТ 19.505-79\* 6), ГОСТ 19.604-78\* 7)). 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом

1. Назначение программы 1.1. Функциональное назначение программы Специальное программное обеспечение используется для управления кандидатами, работодателями и их вакансиями . 1.2. Эксплуатационное назначение программы Специальное программное обеспечение может эксплуатироваться на объектах любого масштаба в сфере кадрового агентства для облегчения работы персонала.
   1. Состав функций

1.3.1 Функция регистрации.

эта функция позволяет пользователю зарегистрироваться в системе

1.3.2 Функция авторизации

эта функция позволяет пользователю авторизоваться в системе под своим логином и паролем

1.3.3 Функция Добавления компании

эта функция позволяет пользователю создать компанию

1.3.4 Функция Добавления кандидата

эта функция позволяет пользователю создать кандидата

1.3.5 Функция Добавления вакансии

эта функция позволяет пользователю создать вакансию

1.3.6 Функция Нахождения кандидатов по навыку

эта функция позволяет пользователю найти кандидатов по навыку

1.3.7 Функция Нахождения вакансий по навыку

эта функция позволяет пользователю найти вакансии по навыку

1.3.8 Функция Показа всех вакансий

эта функция позволяет пользователю посмотреть все вакансии

1.3.8 Функция Выхода из приложения

эта функция позволяет пользователю завершить работу приложения

2. Условия выполнения программы

2.1. Минимальный состав аппаратных средств ОС: Windows 10 Процессор: Как минимум 1 ГГц или SoC. ОЗУ: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем). 34 Место на жестком диске: 16 ГБ (для 32-разрядных систем) или 20 ГБ (для 64-разрядных систем). Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0. Дисплей: 800 x 600. 2.2. Минимальный состав программных средств Дополнительные программные средства не требуются.

2.3. Требование к персоналу (пользователю) Конечный пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

1. Подготовка окружения

Перед запуском программы выполните следующие действия:

Убедитесь, что на вашем компьютере установлен PostgreSQL, и вы настроили базу данных с именем KURSOVAYA.

Создайте пользователя базы данных с именем postgres и паролем

Проверьте, что драйвер lib/pq (PostgreSQL driver) установлен. Если нет, выполните команду:

go get github.com/lib/pq

2. Настройка программы

Скачайте или скопируйте файл программы в каталог на вашем компьютере.

Убедитесь, что каталог программы имеет доступ на чтение и запись.

3. Запуск программы

Запустите программу из терминала командой:

go run main.go

При первом запуске программа автоматически создаст необходимые таблицы в базе данных:

users — для хранения данных пользователей.

companies — для списка компаний.

candidates — для кандидатов.

job\_openings — для вакансий.

4. Основные функции программы

После запуска программы вам будет предложено выбрать одно из следующих действий:

Зарегистрироваться

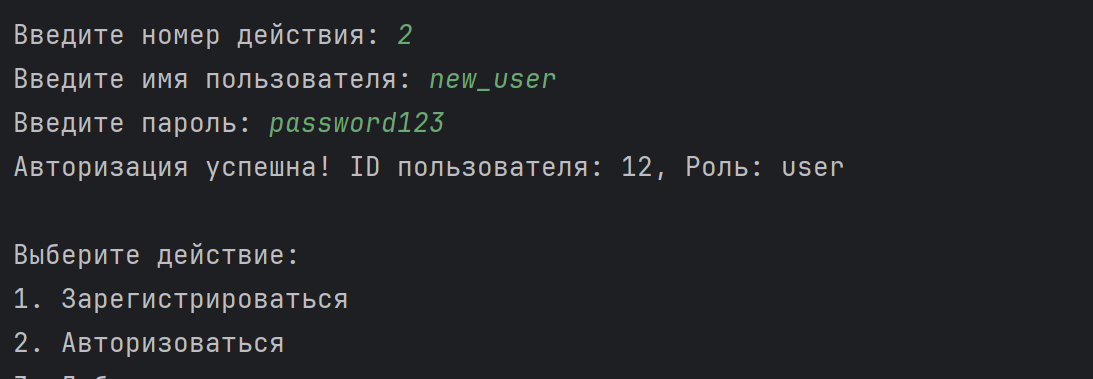
Введите имя пользователя и пароль. Пароль будет зашифрован и сохранён в базе данных.

Если пользователь с таким именем уже существует, программа сообщит об этом.



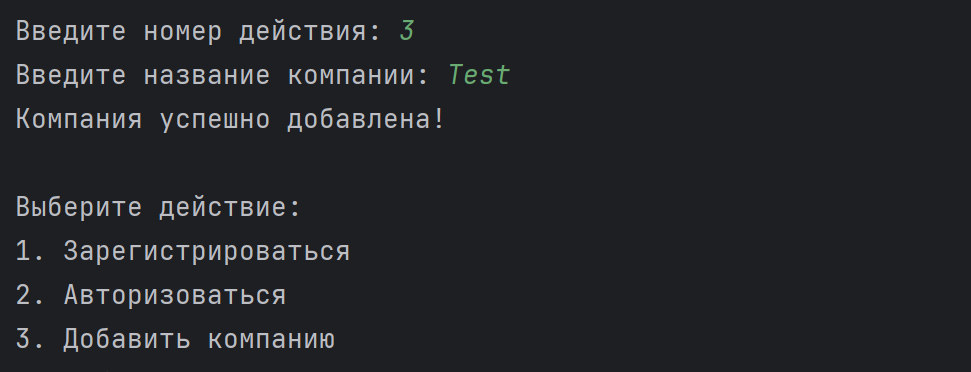
Авторизоваться

Введите имя пользователя и пароль. В случае успешной авторизации вы получите информацию об ID пользователя и его роли.



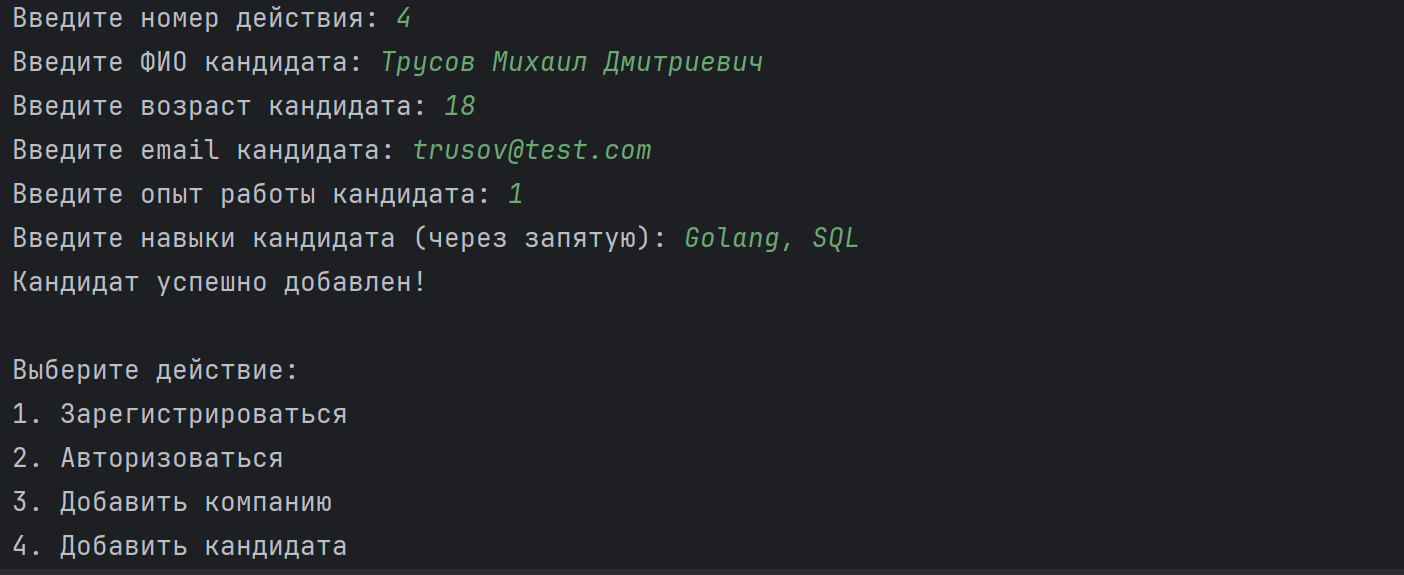
Добавить компанию

Введите название компании. Оно будет добавлено в таблицу companies.



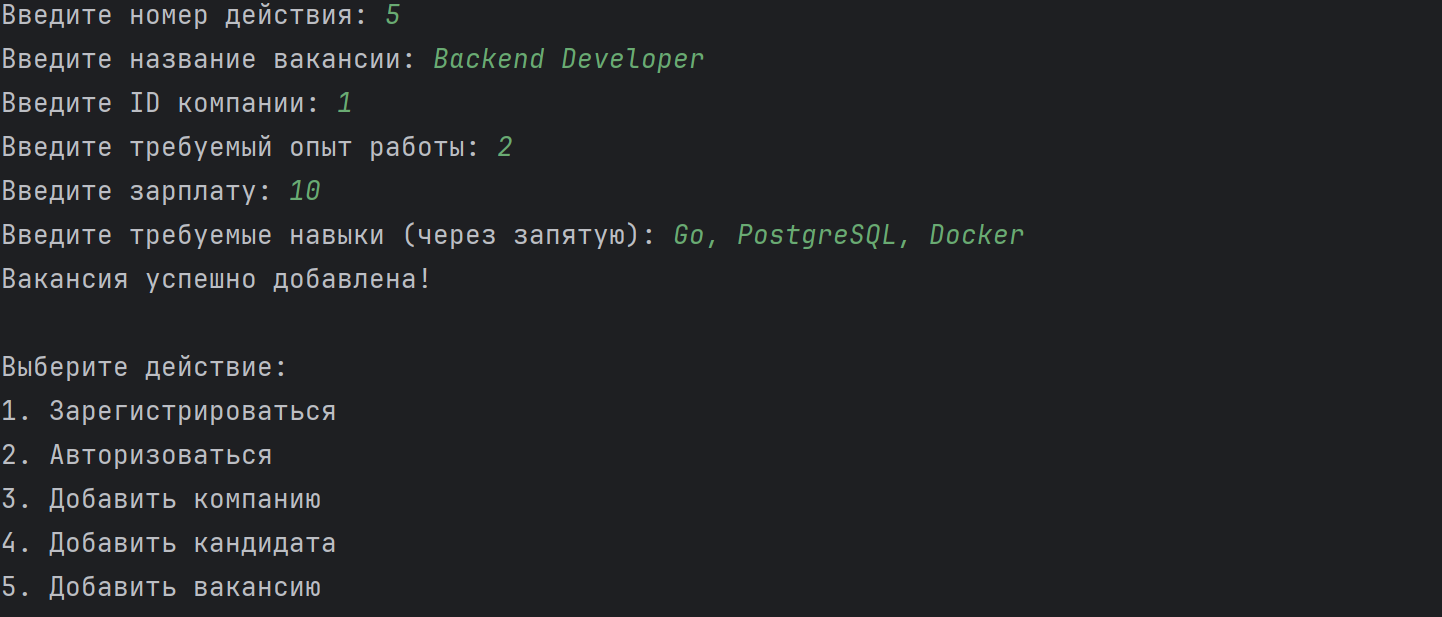
Добавить кандидата

Введите данные кандидата: ФИО, возраст, email, опыт работы, навыки (через запятую). Эти данные будут добавлены в таблицу candidates.



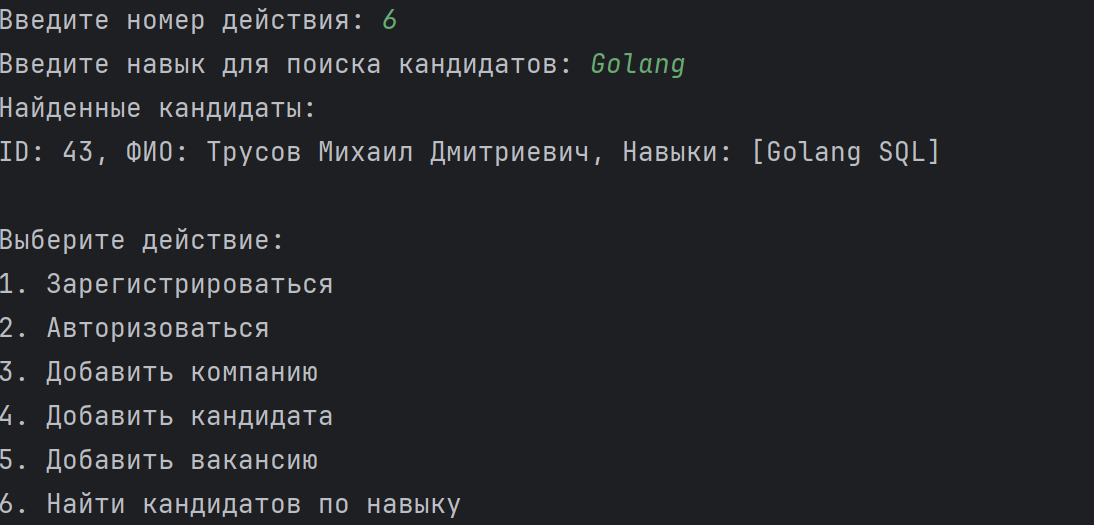
Добавить вакансию

Введите данные вакансии: название, ID компании, требуемый опыт работы, зарплату, навыки (через запятую). Вакансия будет добавлена в таблицу job\_openings.



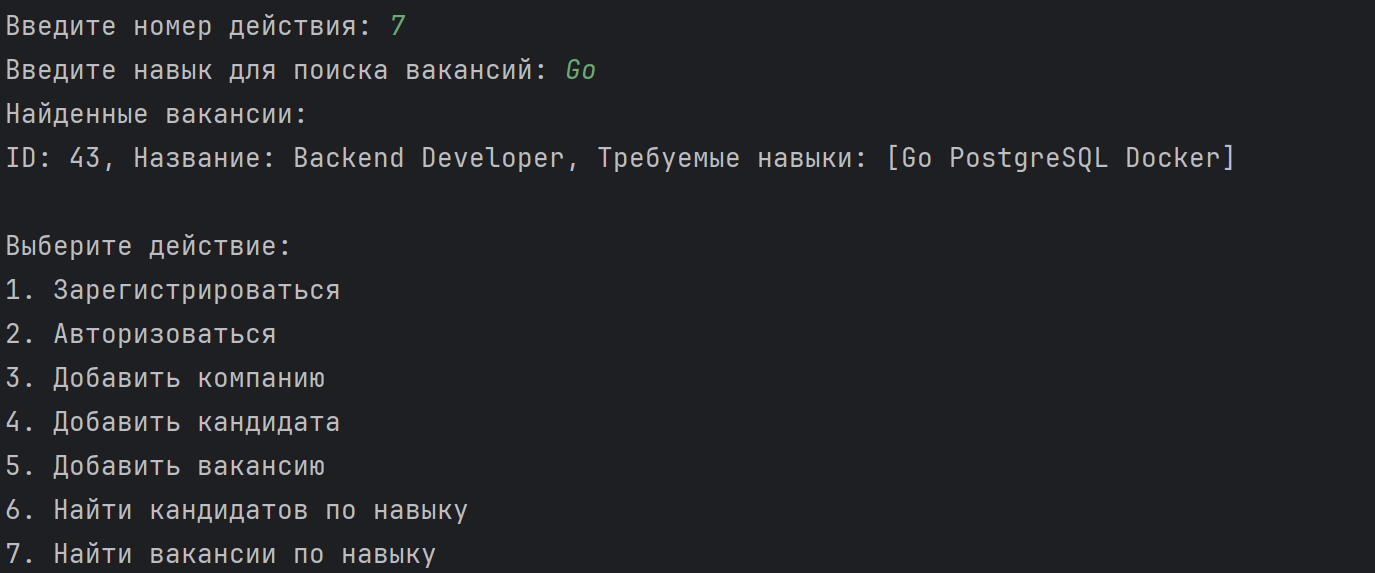
Найти кандидатов по навыку

Введите навык, по которому нужно найти кандидатов. Программа выведет список кандидатов, обладающих указанным навыком.



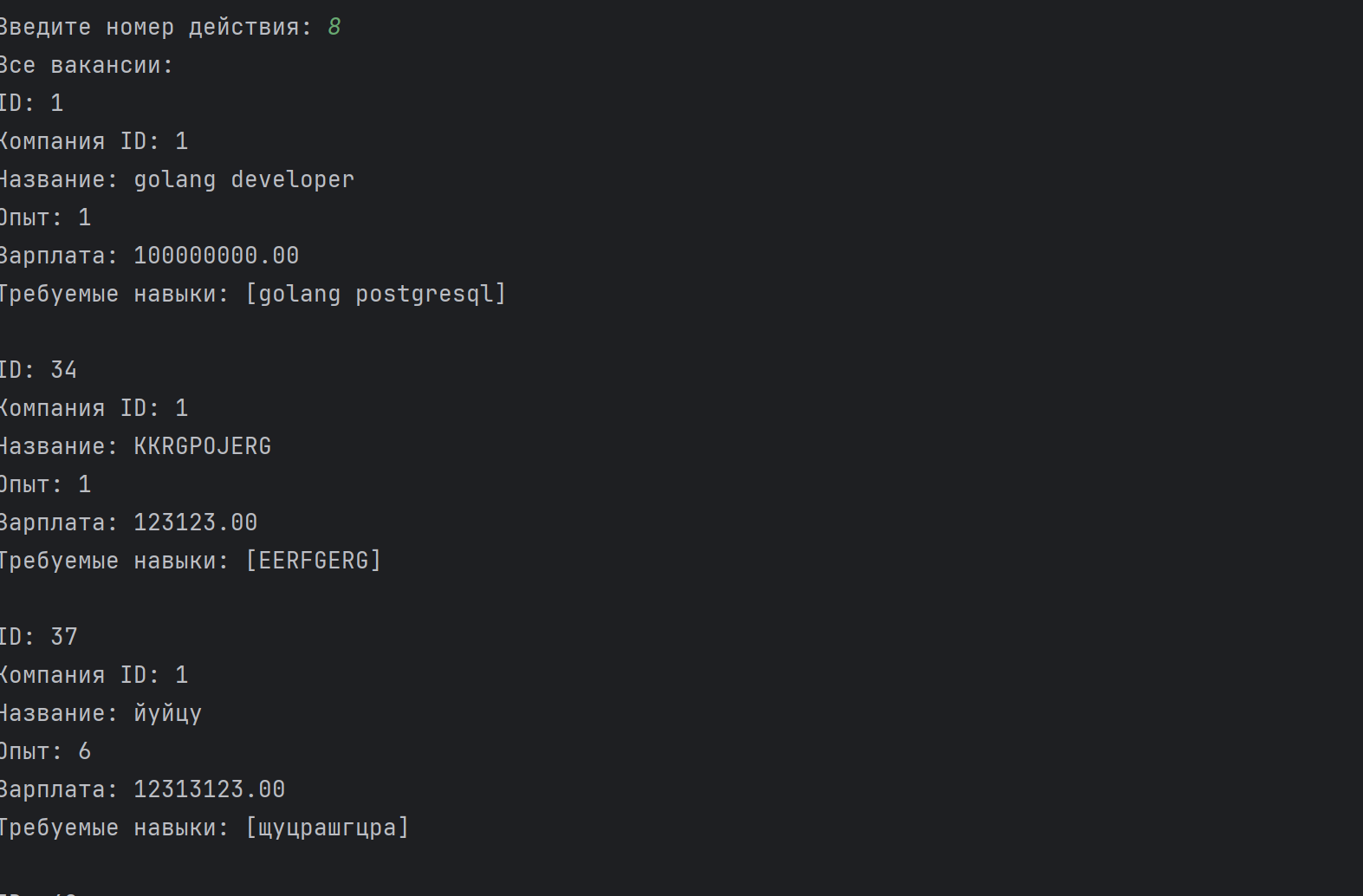
Найти вакансии по навыку

Введите навык, по которому нужно найти вакансии. Программа выведет список вакансий, в которых требуется указанный навык.



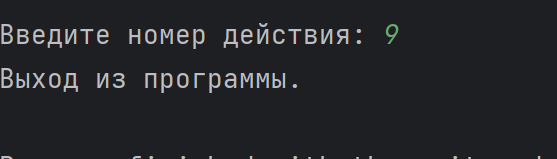
Показать все вакансии

Программа выведет список всех доступных вакансий.



Выйти

Завершение работы программы.



Заключение

В результате выполнения курсового проекта была разработана программа для упрощения взаимодействия между работодателями и кандидатами. Основные возможности включают регистрацию пользователей, добавление компаний, создание вакансий, поиск кандидатов и вакансий по навыкам.

В процессе работы изучены новые библиотеки и сериализация данных в формате JSONB, а также приобретены навыки создания функционала для взаимодействия с базой данных.

Также планируется продолжать работу над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены ниже.

To-do лист:

1. Создание новых функций: вывод всех вакансий, вывод всех компаний
2. Создание графического интерфейса программы
3. Добавить больше обработчиков ошибок

Список литературы и интернет-источников

1. Официальная документация Golang

<https://go.dev/doc/>

1. Подключение базы данных

<https://github.com/jackc/pgx>

1. Работа с базой данных Postgresql

<https://habr.com/ru/companies/avito/articles/461935/>

1. Статья по видам кадровых агенств

<https://eck.ru/blog/autstaffing/vidy-kadrovyh-agentstv/>

1. Официальный сайт SpencerStuart

<https://www.spencerstuart.com>

1. Официальный сайт Workday HCM

<https://www.workday.com>

1. Библиотека bcrypt

<https://pkg.go.dev/golang.org/x/crypto>

Приложение 1.Код программы

package main  
  
import (  
 "bufio"  
 "database/sql"  
 "encoding/json"  
 "errors"  
 "fmt"  
 "golang.org/x/crypto/bcrypt"  
 "log"  
 "os"  
 "strconv"  
 "strings"  
  
 \_ "github.com/lib/pq"  
)  
  
type User struct {  
 ID int `db:"id"`  
 Username string `db:"username"`  
 PasswordHash string `db:"password\_hash"`  
 Role string `db:"role"`  
}  
  
type Candidate struct {  
 ID int `db:"id"`  
 FullName string `db:"full\_name"`  
 Age int `db:"age"`  
 Email string `db:"email"`  
 Experience string `db:"experience"`  
 Skills []string `db:"skills"`  
}  
  
type JobOpening struct {  
 ID int `db:"id"`  
 CompanyID int `db:"company\_id"`  
 Title string `db:"title"`  
 Experience string `db:"experience"`  
 Salary float64 `db:"salary"`  
 RequiredSkills []string `db:"required\_skills"`  
}  
  
type Company struct {  
 ID int `db:"id"`  
 Name string `db:"name"`  
}  
  
func hashPassword(password string) (string, error) {  
 bytes, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.*DefaultCost*)  
 if err != nil {  
 return "", err  
 }  
 return string(bytes), nil  
}  
  
func checkPasswordHash(password, hash string) bool {  
 err := bcrypt.CompareHashAndPassword([]byte(hash), []byte(password))  
 return err == nil  
}  
  
func registerUser(db \*sql.DB, username, password string) error {  
 if username == "" || password == "" {  
 return errors.New("имя пользователя и пароль не могут быть пустыми")  
 }  
  
 row := db.QueryRow("SELECT 1 FROM users WHERE username = $1", username)  
 var exists int  
 err := row.Scan(&exists)  
 if err == nil && exists == 1 {  
 return errors.New("пользователь с таким именем уже существует")  
 } else if err != nil && err != sql.ErrNoRows {  
 return fmt.Errorf("ошибка проверки существования пользователя: %w", err)  
 }  
  
 hashedPassword, err := hashPassword(password)  
 if err != nil {  
 return fmt.Errorf("ошибка хеширования пароля: %w", err)  
 }  
  
 stmt, err := db.Prepare("INSERT INTO users (username, password\_hash) VALUES ($1, $2)")  
 if err != nil {  
 return fmt.Errorf("ошибка подготовки запроса: %w", err)  
 }  
 defer stmt.Close()  
  
 \_, err = stmt.Exec(username, hashedPassword)  
 if err != nil {  
 return fmt.Errorf("ошибка регистрации пользователя: %w", err)  
 }  
 return nil  
}  
  
func loginUser(db \*sql.DB, username, password string) (int, string, error) {  
 stmt, err := db.Prepare("SELECT id, password\_hash, role FROM users WHERE username = $1")  
 if err != nil {  
 return 0, "", fmt.Errorf("Ошибка подготовки запроса: %w", err)  
 }  
 defer stmt.Close()  
  
 var user User  
 err = stmt.QueryRow(username).Scan(&user.ID, &user.PasswordHash, &user.Role)  
 if err != nil {  
 if err == sql.ErrNoRows {  
 return 0, "", errors.New("пользователь не найден")  
 }  
 return 0, "", fmt.Errorf("ошибка авторизации: %w", err)  
 }  
  
 if !checkPasswordHash(password, user.PasswordHash) {  
 return 0, "", errors.New("неверный пароль")  
 }  
  
 return user.ID, user.Role, nil  
}  
  
func getInput(prompt string) string {  
 reader := bufio.NewReader(os.Stdin)  
 fmt.Print(prompt)  
 input, \_ := reader.ReadString('\n')  
 return strings.TrimSpace(input)  
}  
  
func getIntInput(prompt string) (int, error) {  
 input := getInput(prompt)  
 num, err := strconv.Atoi(input)  
 if err != nil {  
 return 0, fmt.Errorf("неверный ввод целого числа: %w", err)  
 }  
 return num, nil  
}  
  
func getFloatInput(prompt string) (float64, error) {  
 input := getInput(prompt)  
 num, err := strconv.ParseFloat(input, 64)  
 if err != nil {  
 return 0, fmt.Errorf("неверный ввод вещественного числа: %w", err)  
 }  
 return num, nil  
}  
  
func getStringArrayInput(prompt string) ([]string, error) {  
 input := getInput(prompt)  
 if input == "" {  
 return []string{}, nil  
 }  
 skills := strings.Split(input, ",")  
 for i, skill := range skills {  
 skills[i] = strings.TrimSpace(skill)  
 }  
 return skills, nil  
  
}  
  
func addCompany(db \*sql.DB, companyName string) error {  
 if companyName == "" {  
 return errors.New("имя компании не может быть пустым")  
 }  
 stmt, err := db.Prepare("INSERT INTO companies (name) VALUES ($1)")  
 if err != nil {  
 return fmt.Errorf("ошибка подготовки запроса: %w", err)  
 }  
 defer stmt.Close()  
  
 \_, err = stmt.Exec(companyName)  
 if err != nil {  
 return fmt.Errorf("ошибка добавления компании: %w", err)  
 }  
 return nil  
}  
  
func addCandidate(db \*sql.DB, candidate Candidate) error {  
 if candidate.FullName == "" || candidate.Age <= 0 {  
 return errors.New("не все обязательные поля заполнены для кандидата")  
 }  
  
 skillsJSON, err := json.Marshal(candidate.Skills)  
 if err != nil {  
 return fmt.Errorf("ошибка сериализации навыков: %w", err)  
 }  
  
 stmt, err := db.Prepare("INSERT INTO candidates (full\_name, age, email, experience, skills) VALUES ($1, $2, $3, $4, $5)")  
 if err != nil {  
 return fmt.Errorf("ошибка подготовки запроса: %w", err)  
 }  
 defer stmt.Close()  
  
 \_, err = stmt.Exec(candidate.FullName, candidate.Age, candidate.Email, candidate.Experience, skillsJSON)  
 if err != nil {  
 return fmt.Errorf("ошибка добавления кандидата: %w", err)  
 }  
 return nil  
}  
  
func addJobOpening(db \*sql.DB, jobOpening JobOpening) error {  
 if jobOpening.Title == "" || jobOpening.CompanyID <= 0 || jobOpening.Salary <= 0 {  
 return errors.New("не все обязательные поля заполнены для вакансии")  
 }  
  
 requiredSkillsJSON, err := json.Marshal(jobOpening.RequiredSkills)  
 if err != nil {  
 return fmt.Errorf("ошибка сериализации навыков: %w", err)  
 }  
  
 stmt, err := db.Prepare("INSERT INTO job\_openings (company\_id, title, experience, salary, required\_skills) VALUES ($1, $2, $3, $4, $5)")  
 if err != nil {  
 return fmt.Errorf("ошибка подготовки запроса: %w", err)  
 }  
 defer stmt.Close()  
  
 \_, err = stmt.Exec(jobOpening.CompanyID, jobOpening.Title, jobOpening.Experience, jobOpening.Salary, requiredSkillsJSON)  
 if err != nil {  
 return fmt.Errorf("ошибка добавления вакансии: %w", err)  
 }  
 return nil  
}  
  
func findCandidatesBySkill(db \*sql.DB, skill string) ([]Candidate, error) {  
 var candidates []Candidate  
 rows, err := db.Query("SELECT id, full\_name, age, email, experience, skills FROM candidates WHERE skills @> $1::jsonb", `["`+skill+`"]`)  
 if err != nil {  
 return nil, fmt.Errorf("ошибка запроса к базе данных: %w", err)  
 }  
 defer rows.Close()  
  
 for rows.Next() {  
 var candidate Candidate  
 var skillsJSON []byte  
 err := rows.Scan(&candidate.ID, &candidate.FullName, &candidate.Age, &candidate.Email, &candidate.Experience, &skillsJSON)  
 if err != nil {  
 return nil, fmt.Errorf("ошибка сканирования строки: %w", err)  
 }  
 json.Unmarshal(skillsJSON, &candidate.Skills)  
 candidates = append(candidates, candidate)  
 }  
  
 if err := rows.Err(); err != nil {  
 return nil, fmt.Errorf("ошибка чтения строк: %w", err)  
 }  
  
 return candidates, nil  
}  
func listAllJobOpenings(db \*sql.DB) error {  
 rows, err := db.Query("SELECT id, company\_id, title, experience, salary, required\_skills FROM job\_openings")  
 if err != nil {  
 return fmt.Errorf("ошибка запроса к базе данных: %w", err)  
 }  
 defer rows.Close()  
  
 fmt.Println("Все вакансии:")  
 for rows.Next() {  
 var jobOpening JobOpening  
 var requiredSkillsJSON []byte  
 err := rows.Scan(&jobOpening.ID, &jobOpening.CompanyID, &jobOpening.Title, &jobOpening.Experience, &jobOpening.Salary, &requiredSkillsJSON)  
 if err != nil {  
 return fmt.Errorf("ошибка сканирования строки: %w", err)  
 }  
 json.Unmarshal(requiredSkillsJSON, &jobOpening.RequiredSkills)  
 fmt.Printf("ID: %d\nКомпания ID: %d\nНазвание: %s\nОпыт: %s\nЗарплата: %.2f\nТребуемые навыки: %v\n\n",  
 jobOpening.ID, jobOpening.CompanyID, jobOpening.Title, jobOpening.Experience, jobOpening.Salary, jobOpening.RequiredSkills)  
 }  
  
 if err := rows.Err(); err != nil {  
 return fmt.Errorf("ошибка чтения строк: %w", err)  
 }  
  
 return nil  
}  
  
func findJobOpeningsBySkill(db \*sql.DB, skill string) ([]JobOpening, error) {  
 var jobOpenings []JobOpening  
 rows, err := db.Query("SELECT id, company\_id, title, experience, salary, required\_skills FROM job\_openings WHERE required\_skills @> $1::jsonb", `["`+skill+`"]`)  
 if err != nil {  
 return nil, fmt.Errorf("ошибка запроса к базе данных: %w", err)  
 }  
 defer rows.Close()  
  
 for rows.Next() {  
 var jobOpening JobOpening  
 var requiredSkillsJSON []byte  
 err := rows.Scan(&jobOpening.ID, &jobOpening.CompanyID, &jobOpening.Title, &jobOpening.Experience, &jobOpening.Salary, &requiredSkillsJSON)  
 if err != nil {  
 return nil, fmt.Errorf("ошибка сканирования строки: %w", err)  
 }  
 json.Unmarshal(requiredSkillsJSON, &jobOpening.RequiredSkills)  
 jobOpenings = append(jobOpenings, jobOpening)  
 }  
  
 if err := rows.Err(); err != nil {  
 return nil, fmt.Errorf("ошибка чтения строк: %w", err)  
 }  
  
 return jobOpenings, nil  
}  
  
func createTables(db \*sql.DB) error {  
 \_, err := db.Exec(`  
 CREATE TABLE IF NOT EXISTS users (  
 id SERIAL PRIMARY KEY,  
 username TEXT UNIQUE NOT NULL,  
 password\_hash TEXT NOT NULL,  
 role TEXT NOT NULL DEFAULT 'user'  
 );  
  
 CREATE TABLE IF NOT EXISTS companies (  
 id SERIAL PRIMARY KEY,  
 name TEXT UNIQUE NOT NULL  
 );  
  
 CREATE TABLE IF NOT EXISTS candidates (  
 id SERIAL PRIMARY KEY,  
 full\_name TEXT NOT NULL,  
 age INTEGER NOT NULL,  
 email TEXT NOT NULL,  
 experience TEXT,  
 skills JSONB  
 );  
  
 CREATE TABLE IF NOT EXISTS job\_openings (  
 id SERIAL PRIMARY KEY,  
 company\_id INTEGER REFERENCES companies(id) ON DELETE CASCADE,  
 title TEXT NOT NULL,  
 experience TEXT,  
 salary NUMERIC(10,2) NOT NULL,  
 required\_skills JSONB  
 );  
`)  
 if err != nil {  
 return fmt.Errorf("ошибка создания таблиц: %w", err)  
 }  
 return nil  
}  
  
func handleError(err error) {  
 if err != nil {  
 fmt.Println("Произошла ошибка:", err)  
 }  
}  
func main() {  
 db, err := sql.Open("postgres", "user=postgres password= dbname=KURSOVAYA sslmode=disable")  
 if err != nil {  
 log.Fatal(err)  
 }  
 defer db.Close()  
  
 err = createTables(db)  
 handleError(err)  
 if err != nil {  
 return  
 }  
  
 for {  
 fmt.Println("\nВыберите действие:")  
 fmt.Println("1. Зарегистрироваться")  
 fmt.Println("2. Авторизоваться")  
 fmt.Println("3. Добавить компанию")  
 fmt.Println("4. Добавить кандидата")  
 fmt.Println("5. Добавить вакансию")  
 fmt.Println("6. Найти кандидатов по навыку")  
 fmt.Println("7. Найти вакансии по навыку")  
 fmt.Println("8. Показать все вакансии")  
 fmt.Println("9. Выйти")  
  
 choice, err := getIntInput("Введите номер действия: ")  
 handleError(err)  
 if err != nil {  
 continue  
 }  
  
 switch choice {  
 case 1:  
 username := getInput("Введите имя пользователя: ")  
 password := getInput("Введите пароль: ")  
 err := registerUser(db, username, password)  
 handleError(err)  
 if err == nil {  
 fmt.Println("Регистрация успешна!")  
 }  
 case 2:  
 username := getInput("Введите имя пользователя: ")  
 password := getInput("Введите пароль: ")  
 userID, role, err := loginUser(db, username, password)  
 handleError(err)  
 if err == nil {  
 fmt.Printf("Авторизация успешна! ID пользователя: %d, Роль: %s\n", userID, role)  
 }  
 case 3:  
 companyName := getInput("Введите название компании: ")  
 err := addCompany(db, companyName)  
 handleError(err)  
 if err == nil {  
 fmt.Println("Компания успешно добавлена!")  
 }  
 case 4:  
 candidate := Candidate{}  
 candidate.FullName = getInput("Введите ФИО кандидата: ")  
 candidate.Age, err = getIntInput("Введите возраст кандидата: ")  
 handleError(err)  
 if err != nil {  
 continue  
 }  
 candidate.Email = getInput("Введите email кандидата: ")  
 candidate.Experience = getInput("Введите опыт работы кандидата: ")  
 candidate.Skills, err = getStringArrayInput("Введите навыки кандидата (через запятую): ")  
 handleError(err)  
 if err != nil {  
 continue  
 }  
 err = addCandidate(db, candidate)  
 handleError(err)  
 if err == nil {  
 fmt.Println("Кандидат успешно добавлен!")  
 }  
 case 5:  
 jobOpening := JobOpening{}  
 jobOpening.Title = getInput("Введите название вакансии: ")  
 jobOpening.CompanyID, err = getIntInput("Введите ID компании: ")  
 handleError(err)  
 if err != nil {  
 continue  
 }  
 jobOpening.Experience = getInput("Введите требуемый опыт работы: ")  
 jobOpening.Salary, err = getFloatInput("Введите зарплату: ")  
 handleError(err)  
 if err != nil {  
 continue  
 }  
 jobOpening.RequiredSkills, err = getStringArrayInput("Введите требуемые навыки (через запятую): ")  
 handleError(err)  
 if err != nil {  
 continue  
 }  
 err = addJobOpening(db, jobOpening)  
 handleError(err)  
 if err == nil {  
 fmt.Println("Вакансия успешно добавлена!")  
 }  
 case 6:  
 skill := getInput("Введите навык для поиска кандидатов: ")  
 candidates, err := findCandidatesBySkill(db, skill)  
 handleError(err)  
 if err == nil {  
 fmt.Println("Найденные кандидаты:")  
 for \_, c := range candidates {  
 fmt.Printf("ID: %d, ФИО: %s, Навыки: %v\n", c.ID, c.FullName, c.Skills)  
 }  
 }  
 case 7:  
 skill := getInput("Введите навык для поиска вакансий: ")  
 jobOpenings, err := findJobOpeningsBySkill(db, skill)  
 handleError(err)  
 if err == nil {  
 fmt.Println("Найденные вакансии:")  
 for \_, j := range jobOpenings {  
 fmt.Printf("ID: %d, Название: %s, Требуемые навыки: %v\n", j.ID, j.Title, j.RequiredSkills)  
 }  
 }  
 case 8:  
 err := listAllJobOpenings(db)  
 handleError(err)  
 if err != nil {  
 fmt.Println("Ошибка при выводе вакансий:", err)  
 }  
 case 9:  
 fmt.Println("Выход из программы.")  
 return  
 default:  
 fmt.Println("Неверный выбор действия. Попробуйте снова.")  
 }  
 }  
}