



ДЕЛАЙ: Полная инструкция по созданию Telegram HR Bot на Windows

Автор: Manus AI

Версия: 1.0

Дата: 2025

Время выполнения: 2-4 часа



Содержание

1. [Введение и обзор проекта](#)
 2. [Подготовка Windows к разработке](#)
 3. [Установка и настройка Python](#)
 4. [Установка и настройка Git](#)
 5. [Установка и настройка PostgreSQL](#)
 6. [Создание Telegram бота](#)
 7. [Клонирование и настройка проекта](#)
 8. [Настройка базы данных](#)
 9. [Запуск и тестирование бота](#)
 10. [Настройка GitHub синхронизации](#)
 11. [Подготовка к деплою на Яндекс.Облако](#)
 12. [Устранение проблем](#)
 13. [Дополнительные возможности](#)
-

Введение и обзор проекта

Добро пожаловать в самую подробную инструкцию по созданию профессионального Telegram HR Bot на Windows! Эта инструкция написана так, чтобы даже человек без опыта программирования смог успешно создать и запустить полнофункционального бота для найма сотрудников и поиска работы.

Что мы будем создавать

Наш Telegram HR Bot - это современное решение для автоматизации процессов найма и поиска работы. Бот предоставляет две основные роли пользователей:

Для работодателей: - Создание и управление вакансиями с детальными описаниями - Просмотр и обработка откликов от кандидатов - Статистика и аналитика по вакансиям - Быстрые действия для эффективной работы

Для соискателей: - Расширенный поиск вакансий с множественными фильтрами - Отклики на вакансии с персонализированными сопроводительными письмами - Умные уведомления о новых вакансиях по подписке - Управление профилем и историей откликов

Технологический стек

Проект построен на современных и надежных технологиях:

- **Backend:** Python 3.11+ с Flask фреймворком
- **База данных:** PostgreSQL 15+ для надежного хранения данных
- **Telegram API:** pyTelegramBotAPI для взаимодействия с Telegram
- **ORM:** SQLAlchemy для работы с базой данных
- **Планировщик:** Schedule для автоматических уведомлений
- **Контейнеризация:** Docker для упрощения деплоя
- **Версионирование:** Git с синхронизацией на GitHub
- **Облачный деплой:** Яндекс.Облако для продакшен среды

Архитектура системы

Система спроектирована с учетом принципов масштабируемости и поддерживаемости. Основные компоненты включают:

1. **Telegram Bot Handler** - обрабатывает все входящие сообщения и команды
2. **Database Models** - модели данных для пользователей, вакансий, откликов и подписок
3. **Business Logic** - логика создания вакансий, поиска, уведомлений
4. **Notification Scheduler** - планировщик для автоматической отправки уведомлений
5. **REST API** - дополнительные API endpoints для статистики и интеграций

Ожидаемый результат

По завершении этой инструкции у вас будет:

- Полностью функциональный Telegram HR Bot
- Настроенная PostgreSQL база данных
- Синхронизация с GitHub репозиторием
- Готовность к деплою на Яндекс.Облако
- Понимание архитектуры и возможности дальнейшего развития

Подготовка Windows к разработке

Прежде чем начать установку основных компонентов, необходимо подготовить операционную систему Windows для разработки. Этот раздел критически важен для успешного выполнения всех последующих шагов.

Проверка версии Windows

Для комфортной работы рекомендуется использовать Windows 10 версии 1903 или новее, либо Windows 11. Проверить версию можно следующим образом:

1. Нажмите `Win + R`

2. Введите `winver` и нажмите Enter
3. В открывшемся окне проверьте версию

Если у вас более старая версия Windows, рекомендуется обновиться до поддерживаемой версии для обеспечения совместимости всех инструментов.

Включение Windows Subsystem for Linux (WSL) - Опционально

Хотя мы будем работать в нативной Windows среде, WSL может быть полезен для некоторых операций. Для его включения:

1. Откройте PowerShell от имени администратора
2. Выполните команду: `wsl --install`
3. Перезагрузите компьютер при необходимости

Настройка Windows Terminal

Windows Terminal значительно улучшает опыт работы с командной строкой:

1. Откройте Microsoft Store
2. Найдите и установите "Windows Terminal"
3. Запустите Windows Terminal
4. Настройте профиль по умолчанию на PowerShell или Command Prompt

Настройка политик выполнения PowerShell

Для корректной работы некоторых скриптов необходимо настроить политики выполнения:

1. Откройте PowerShell от имени администратора
2. Выполните: `Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`
3. Подтвердите изменения, введя `Y`

Установка Chocolatey (Опционально)

Chocolatey - это пакетный менеджер для Windows, который упрощает установку программ:

1. Откройте PowerShell от имени администратора
2. Выполните команду:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object  
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps
```

1. Перезапустите PowerShell
2. Проверьте установку: `choco --version`

Создание рабочей директории

Создайте структуру папок для проекта:

1. Откройте Проводник Windows
2. Перейдите на рабочий стол (Desktop)
3. Создайте папку `TelegramHRBot`
4. Внутри создайте подпапки:
5. `Projects` - для исходного кода
6. `Tools` - для инструментов разработки
7. `Documentation` - для документации

Настройка переменных окружения

Подготовьте переменные окружения для удобной работы:

1. Нажмите `win + x` и выберите "Система"
2. Нажмите "Дополнительные параметры системы"
3. Нажмите "Переменные среды"
4. В разделе "Переменные пользователя" нажмите "Создать"

5. Добавьте переменную `HRBOT_HOME` со значением пути к папке проекта



Установка и настройка Python

Python является основным языком программирования для нашего проекта. Правильная установка и настройка Python критически важна для успешной работы всей системы.

Выбор версии Python

Для нашего проекта требуется Python версии 3.11 или новее. Эта версия обеспечивает оптимальную производительность и поддержку всех необходимых библиотек. Не используйте Python 3.9 или более старые версии, так как некоторые зависимости могут работать некорректно.

Загрузка Python

1. Откройте веб-браузер и перейдите на официальный сайт Python:
<https://www.python.org/>
2. Нажмите на кнопку "Downloads" в верхнем меню
3. Система автоматически определит вашу операционную систему и предложит подходящую версию
4. Убедитесь, что предлагаемая версия 3.11 или новее
5. Нажмите на кнопку загрузки (обычно это большая желтая кнопка)

Установка Python

После загрузки установочного файла выполните следующие шаги:

1. **Запуск установщика:** Найдите загруженный файл (обычно в папке "Загрузки") и запустите его двойным щелчком
2. **Важно:** Обязательно поставьте галочку "Add Python to PATH" в нижней части окна установщика
3. Выберите "Install Now" для стандартной установки

4. Если появится запрос UAC (Контроль учетных записей), нажмите "Да"
5. Дождитесь завершения установки (обычно 2-5 минут)
6. В конце установки нажмите "Close"

Проверка установки Python

Проверим, что Python установлен корректно:

1. Откройте Command Prompt или PowerShell:
2. Нажмите `Win + R`
3. Введите `cmd` и нажмите Enter
4. Введите команду: `python --version`
5. Вы должны увидеть что-то вроде: `Python 3.11.x`
6. Также проверьте pip: `pip --version`
7. Вы должны увидеть информацию о версии pip

Если команды не работают, это означает, что Python не добавлен в PATH. В этом случае:

1. Найдите папку установки Python (обычно `C:\Users\[ваше_имя]\AppData\Local\Programs\Python\Python311\`)
2. Добавьте эту папку и подпапку `Scripts` в переменную PATH через "Переменные среды"

Обновление pip

pip - это менеджер пакетов Python. Обновим его до последней версии:

1. Откройте Command Prompt или PowerShell
2. Выполните команду: `python -m pip install --upgrade pip`
3. Дождитесь завершения обновления

Установка виртуального окружения

Виртуальные окружения позволяют изолировать зависимости проектов друг от друга:

1. Установите virtualenv: `pip install virtualenv`
2. Проверьте установку: `virtualenv --version`

Настройка Python для разработки

Для удобной разработки рекомендуется настроить несколько дополнительных параметров:

1. **Кодировка по умолчанию:** Создайте переменную окружения `PYTHONIOENCODING` со значением `utf-8`
2. **Отключение буферизации:** Создайте переменную окружения `PYTHONUNBUFFERED` со значением `1`

Установка дополнительных инструментов Python

Установим несколько полезных инструментов для разработки:

```
pip install --upgrade setuptools wheel
pip install black flake8 pytest
```

Эти инструменты помогут в форматировании кода, проверке стиля и тестировании.



Установка и настройка Git

Git - это система контроля версий, которая позволяет отслеживать изменения в коде и синхронизировать работу с GitHub. Правильная настройка Git является основой для эффективной разработки и деплоя.

Загрузка Git для Windows

1. Откройте веб-браузер и перейдите на официальный сайт Git: <https://git-scm.com/>
2. Нажмите на кнопку "Download for Windows"
3. Загрузка должна начаться автоматически
4. Если загрузка не началась, нажмите на прямую ссылку для загрузки

Установка Git

Процесс установки Git включает множество опций. Рекомендуемые настройки:

1. **Запуск установщика:** Запустите загруженный файл от имени администратора
2. **Лицензия:** Прочитайте и примите лицензионное соглашение
3. **Папка установки:** Оставьте папку по умолчанию (`C:\Program Files\Git`)
4. **Компоненты:** Выберите следующие компоненты:
 5. Git Bash Here
 6. Git GUI Here
 7. Git LFS (Large File Support)
 8. Associate .git* configuration files with the default text editor
 9. Associate .sh files to be run with Bash
10. **Папка в меню Пуск:** Оставьте по умолчанию
11. **Редактор по умолчанию:** Выберите "Use Visual Studio Code as Git's default editor" (если у вас установлен VS Code) или "Use Notepad++ as Git's default editor"
12. **Имя ветки по умолчанию:** Выберите "Override the default branch name for new repositories" и введите "main"
13. **PATH environment:** Выберите "Git from the command line and also from 3rd-party software"
14. **SSH executable:** Выберите "Use bundled OpenSSH"
15. **HTTPS transport backend:** Выберите "Use the OpenSSL library"

16. **Line ending conversions:** Выберите "Checkout Windows-style, commit Unix-style line endings"
17. **Terminal emulator:** Выберите "Use MinTTY (the default terminal of MSYS2)"
18. **Default behavior of git pull:** Выберите "Default (fast-forward or merge)"
19. **Credential helper:** Выберите "Git Credential Manager"
20. **Extra options:** Включите "Enable file system caching" и "Enable symbolic links"
21. **Experimental options:** Можете оставить пустыми
22. Нажмите "Install" и дождитесь завершения установки

Проверка установки Git

1. Откройте Command Prompt, PowerShell или Git Bash
2. Введите команду: `git --version`
3. Вы должны увидеть версию Git, например: `git version 2.42.0.windows.1`

Первоначальная настройка Git

Настроим Git с вашими данными:

1. Откройте Git Bash или Command Prompt
2. Настройте имя пользователя:

```
git config --global user.name "Ваше Имя"
```

1. Настройте email:

```
git config --global user.email "your.email@example.com"
```

1. Настройте редактор по умолчанию (если не настроили при установке):

```
git config --global core.editor "notepad"
```

1. Настройте автоматическое преобразование окончаний строк:

```
git config --global core.autocrlf true
```

Настройка SSH ключей для GitHub

SSH ключи обеспечивают безопасное соединение с GitHub без необходимости вводить пароль каждый раз:

1. Проверка существующих ключей:

```
ls -al ~/.ssh
```

Если папка не существует или пуста, переходите к следующему шагу.

1. Генерация нового SSH ключа:

```
ssh-keygen -t ed25519 -C "your.email@example.com"
```

Замените email на ваш реальный email.

1. **Сохранение ключа:** Нажмите Enter для сохранения в папке по умолчанию

2. **Пароль для ключа:** Можете оставить пустым или задать пароль для дополнительной безопасности

3. Добавление ключа в ssh-agent:

```
eval "$(ssh-agent -s)"  
ssh-add ~/.ssh/id_ed25519
```

1. Копирование публичного ключа:

```
cat ~/.ssh/id_ed25519.pub
```

Скопируйте весь вывод команды.

Добавление SSH ключа в GitHub

1. Войдите в свой аккаунт GitHub (или создайте новый на <https://github.com/>)
2. Нажмите на ваш аватар в правом верхнем углу
3. Выберите "Settings"
4. В левом меню выберите "SSH and GPG keys"
5. Нажмите "New SSH key"

6. Введите название ключа (например, "Windows PC")
7. Вставьте скопированный публичный ключ в поле "Key"
8. Нажмите "Add SSH key"

Проверка соединения с GitHub

Проверим, что SSH соединение работает:

```
ssh -T git@github.com
```

Вы должны увидеть сообщение: "Hi [username]! You've successfully authenticated, but GitHub does not provide shell access."

Установка и настройка PostgreSQL

PostgreSQL - это мощная объектно-реляционная система управления базами данных, которая будет хранить все данные нашего HR бота. Правильная установка и настройка PostgreSQL критически важна для стабильной работы приложения.

Загрузка PostgreSQL

1. Откройте веб-браузер и перейдите на официальный сайт PostgreSQL:
<https://www.postgresql.org/>
2. Нажмите на кнопку "Download"
3. Выберите "Windows"
4. Нажмите "Download the installer" от EnterpriseDB
5. Выберите версию 15.x или новее для Windows x86-64
6. Дождитесь завершения загрузки

Установка PostgreSQL

Процесс установки PostgreSQL требует внимательности к деталям:

1. **Запуск установщика:** Запустите загруженный файл от имени администратора
2. **Приветствие:** Нажмите "Next" в приветственном окне
3. **Папка установки:** Оставьте папку по умолчанию (`C:\Program Files\PostgreSQL\15`) или выберите другую
4. **Компоненты для установки:** Убедитесь, что выбраны:
 5. PostgreSQL Server
 6. pgAdmin 4 (графический интерфейс)
 7. Stack Builder (для дополнительных компонентов)
 8. Command Line Tools
9. **Папка для данных:** Оставьте по умолчанию (`C:\Program Files\PostgreSQL\15\data`)
10. **Пароль суперпользователя:** Введите надежный пароль для пользователя `postgres` . **ВАЖНО:** Запомните этот пароль, он понадобится позже!
11. **Порт:** Оставьте порт по умолчанию `5432`
12. **Локаль:** Выберите "Russian, Russia" или оставьте "Default locale"
13. **Сводка:** Проверьте настройки и нажмите "Next"
14. **Установка:** Нажмите "Next" и дождитесь завершения установки (может занять 5-10 минут)
15. **Stack Builder:** Можете снять галочку, если не планируете устанавливать дополнительные компоненты
16. **Завершение:** Нажмите "Finish"

Проверка установки PostgreSQL

1. **Проверка службы:**
 2. Нажмите `Win + R` , введите `services.msc`
 3. Найдите службу "postgresql-x64-15" (или аналогичную)
 4. Убедитесь, что статус "Выполняется"
5. **Проверка через командную строку:**

6. Откройте Command Prompt

7. Перейдите в папку bin PostgreSQL: `cmd cd "C:\Program Files\PostgreSQL\15\bin"`

8. Проверьте версию: `cmd psql --version`

9. **Проверка подключения:** `cmd psql -U postgres -h localhost` Введите пароль, который вы задали при установке.

Настройка PostgreSQL для разработки

Создание пользователя для проекта

1. Подключитесь к PostgreSQL как суперпользователь:

```
psql -U postgres -h localhost
```

1. Создайте нового пользователя:

```
CREATE USER hr_bot_user WITH PASSWORD 'secure_password_here';
```

1. Дайте пользователю права на создание баз данных:

```
ALTER USER hr_bot_user CREATEDB;
```

1. Выйдите из psql:

```
\q
```

Создание базы данных для проекта

1. Создайте базу данных:

```
createdb -U hr_bot_user -h localhost telegram_hr_bot
```

1. Проверьте создание базы данных:

```
psql -U hr_bot_user -h localhost -d telegram_hr_bot
```

1. Если подключение успешно, выйдите:

\q

Настройка pgAdmin 4

pgAdmin 4 - это графический интерфейс для управления PostgreSQL:

1. **Запуск pgAdmin:** Найдите pgAdmin 4 в меню Пуск и запустите
2. **Первый запуск:** При первом запуске потребуется задать мастер-пароль для pgAdmin
3. **Добавление сервера:**
 4. Щелкните правой кнопкой на "Servers" в левой панели
 5. Выберите "Create" → "Server"
 6. На вкладке "General" введите имя: "Local PostgreSQL"
 7. На вкладке "Connection":
 - Host name/address: localhost
 - Port: 5432
 - Maintenance database: postgres
 - Username: postgres
 - Password: ваш пароль суперпользователя
 8. Нажмите "Save"
9. **Проверка подключения:** В левой панели должен появиться сервер "Local PostgreSQL"

Настройка переменных окружения для PostgreSQL

Добавим PostgreSQL в PATH для удобства работы:

1. Нажмите `win + x` и выберите "Система"
2. Нажмите "Дополнительные параметры системы"
3. Нажмите "Переменные среды"

4. В разделе "Переменные пользователя" найдите переменную "Path" и нажмите "Изменить"
5. Нажмите "Создать" и добавьте путь: `C:\Program Files\PostgreSQL\15\bin`
6. Нажмите "ОК" во всех окнах
7. Перезапустите Command Prompt для применения изменений

Настройка автозапуска PostgreSQL

Убедимся, что PostgreSQL запускается автоматически при загрузке Windows:

1. Откройте "Службы" (win + R, введите `services.msc`)
2. Найдите службу "postgresql-x64-15"
3. Щелкните правой кнопкой и выберите "Свойства"
4. Установите "Тип запуска" в "Автоматически"
5. Нажмите "ОК"

Создание резервной копии конфигурации

Создадим резервную копию важных файлов конфигурации:

1. Перейдите в папку данных PostgreSQL: `C:\Program Files\PostgreSQL\15\data`
2. Скопируйте файлы `postgresql.conf` и `pg_hba.conf` в безопасное место
3. Эти файлы понадобятся для восстановления настроек в случае проблем



Создание Telegram бота

Создание Telegram бота - это первый шаг к запуску нашего HR бота. Telegram предоставляет простой и удобный способ создания ботов через специального бота @BotFather.

Регистрация в Telegram

Если у вас еще нет аккаунта Telegram:

1. Скачайте Telegram для Windows с официального сайта:
<https://desktop.telegram.org/>
2. Установите приложение и зарегистрируйтесь, используя ваш номер телефона
3. Подтвердите регистрацию через SMS код

Создание бота через BotFather

BotFather - это официальный бот Telegram для создания и управления другими ботами:

1. Поиск BotFather:

2. Откройте Telegram
3. В поиске введите `@BotFather`
4. Выберите официального BotFather (с синей галочкой)

5. Начало диалога:

6. Нажмите "Start" или отправьте команду `/start`
7. BotFather покажет список доступных команд

8. Создание нового бота:

9. Отправьте команду `/newbot`
10. BotFather попросит ввести имя для вашего бота
11. Введите: `HR Bot` для найма (или любое другое имя)

12. Выбор username:

13. BotFather попросит ввести username (должен заканчиваться на "bot")
14. Введите что-то вроде: `your_company_hr_bot` (замените на уникальное имя)
15. Если имя занято, попробуйте другие варианты

16. Получение токена:

17. После успешного создания BotFather отправит вам токен

18. Токен выглядит примерно так: `1234567890:ABCdefGHIjklMNOpqrSTUVwxyz`
19. **ВАЖНО:** Сохраните этот токен в безопасном месте! Он понадобится для настройки бота

Настройка бота

Настроим основные параметры бота:

1. **Описание бота:**
2. Отправьте команду `/setdescription`
3. Выберите вашего бота из списка
4. Введите описание: `Профессиональный бот для найма сотрудников и поиска работы. Помогает работодателям находить кандидатов, а соискателям - подходящие вакансии.`
5. **Краткое описание:**
6. Отправьте команду `/setabouttext`
7. Выберите вашего бота
8. Введите: `HR Bot - ваш помощник в мире трудоустройства!`
9. **Команды бота:**
10. Отправьте команду `/setcommands`
11. Выберите вашего бота
12. Введите список команд: `start - Начать работу с ботом menu - Главное меню quick - Быстрые действия profile - Управление профилем jobs - Поиск вакансий myjobs - Мои вакансии (для работодателей) applications - Мои отклики subscribe - Подписки на уведомления stats - Статистика help - Помощь`
13. **Изображение бота** (опционально):
14. Отправьте команду `/setuserpic`
15. Выберите вашего бота

16. Загрузите изображение (рекомендуемый размер: 512x512 пикселей)

Тестирование бота

Проверим, что бот создан корректно:

1. Найдите вашего бота в Telegram по username
2. Нажмите "Start"
3. Пока бот не отвечает - это нормально, так как мы еще не запустили код
4. Убедитесь, что описание и команды отображаются корректно

Безопасность токена

Токен бота - это секретная информация, которая дает полный доступ к управлению ботом:

1. **Никогда не публикуйте токен** в открытых репозиториях или чатах
2. **Используйте переменные окружения** для хранения токена
3. **Регулярно проверяйте** активность бота через BotFather
4. **При компрометации** немедленно создайте новый токен через команду
`/revoke`



Клонирование и настройка проекта

Теперь, когда все основные инструменты установлены, пора клонировать наш проект и настроить его для работы на вашем компьютере.

Создание GitHub аккаунта

Если у вас еще нет аккаунта GitHub:

1. Перейдите на <https://github.com/>
2. Нажмите "Sign up"
3. Заполните форму регистрации
4. Подтвердите email адрес

5. Выберите бесплатный план

Создание репозитория на GitHub

1. **Войдите в GitHub** и нажмите зеленую кнопку "New" или "Create repository"
2. **Название репозитория:** `telegram-hr-bot`
3. **Описание:** `Professional Telegram bot for HR and job search`
4. **Видимость:** Выберите "Private" для приватного репозитория или "Public" для открытого
5. **Инициализация:** Поставьте галочку "Add a README file"
6. **Gitignore:** Выберите "Python" из выпадающего списка
7. **Лицензия:** Выберите "MIT License" (опционально)
8. Нажмите "Create repository"

Клонирование проекта

Теперь клонируем проект на ваш компьютер:

1. **Получение URL репозитория:**
2. На странице вашего репозитория нажмите зеленую кнопку "Code"
3. Выберите вкладку "SSH" (если настроили SSH ключи) или "HTTPS"
4. Скопируйте URL
5. **Клонирование:**
6. Откройте Git Bash или Command Prompt
7. Перейдите на рабочий стол: `bash cd Desktop`
8. Клонировать репозиторий: `bash git clone git@github.com:ваш_username/telegram-hr-bot.git` (замените URL на ваш)
9. **Переход в папку проекта:** `bash cd telegram-hr-bot`

Загрузка исходного кода

Поскольку мы создали пустой репозиторий, нужно добавить исходный код нашего HR бота. Есть несколько способов:

Способ 1: Загрузка архива (рекомендуется)

1. Скачайте архив с исходным кодом (будет предоставлен отдельно)
2. Распакуйте архив в папку проекта
3. Убедитесь, что структура папок выглядит так: telegram-hr-bot/ ├── src/ |
├── bot/ | ├── models/ | └── main.py ├── requirements.txt ├──
Dockerfile ├── docker-compose.yml ├── .env.example └── README.md

Способ 2: Создание файлов вручную

Если архив недоступен, создайте структуру проекта вручную:

1. **Создание папок:** `bash mkdir src src/bot src/models mkdir logs uploads`
2. **Создание основных файлов:**
3. Скопируйте содержимое файлов из предоставленного кода
4. Создайте файлы по одному, используя текстовый редактор

Настройка виртуального окружения

Виртуальное окружение изолирует зависимости проекта:

1. **Создание виртуального окружения:** `bash python -m venv venv`
2. **Активация виртуального окружения:**
3. В Command Prompt: `cmd venv\Scripts\activate`
4. В PowerShell: `powershell venv\Scripts\Activate.ps1`
5. В Git Bash: `bash source venv/Scripts/activate`
6. **Проверка активации:** В начале строки командной строки должно появиться `(venv)`

Установка зависимостей

Установим все необходимые Python библиотеки:

1. **Обновление pip:** `bash python -m pip install --upgrade pip`
2. **Установка зависимостей:** `bash pip install -r requirements.txt`
3. **Проверка установки:** `bash pip list` Вы должны увидеть список установленных пакетов, включая:
4. Flask
5. pyTelegramBotAPI
6. psycopg2-binary
7. SQLAlchemy
8. python-dotenv
9. schedule

Настройка переменных окружения

Создадим файл с настройками для нашего бота:

1. **Копирование примера:** `bash copy .env.example .env`
2. **Редактирование .env файла:** Откройте файл `.env` в текстовом редакторе и заполните:

```
```.env # Telegram Bot Configuration
TELEGRAM_BOT_TOKEN=ваш_токен_от_BotFather

Database Configuration POSTGRES_HOST=localhost POSTGRES_PORT=5432
POSTGRES_DB=telegram_hr_bot POSTGRES_USER=hr_bot_user
POSTGRES_PASSWORD=secure_password_here

Flask Configuration FLASK_SECRET_KEY=ваш_секретный_ключ_для_flask
FLASK_ENV=development FLASK_DEBUG=True

Notification Configuration NOTIFICATION_ENABLED=true

Other Configuration MAX_CONTENT_LENGTH=16777216 ```.env
```

1. **Генерация секретного ключа Flask:** `python python -c "import secrets; print(secrets.token_hex(32))"` Скопируйте результат в `FLASK_SECRET_KEY`

## Проверка настройки проекта

Убедимся, что все настроено корректно:

1. **Проверка структуры проекта:** `bash dir /s` (или `ls -la` в Git Bash)
  2. **Проверка виртуального окружения:** `bash python --version pip --version`
  3. **Проверка переменных окружения:** `python python -c "from dotenv import load_dotenv; load_dotenv(); import os; print('Token loaded:', bool(os.getenv('TELEGRAM_BOT_TOKEN')))"`
- 

## Настройка базы данных

Настройка базы данных - критически важный этап, который обеспечивает корректное хранение и обработку всех данных нашего HR бота.

### Проверка подключения к PostgreSQL

Убедимся, что PostgreSQL работает и доступен:

1. **Проверка службы PostgreSQL:**
2. Откройте "Службы" (`Win + R`, введите `services.msc`)
3. Найдите "postgresql-x64-15" и убедитесь, что статус "Выполняется"
4. **Проверка подключения через psql:** `bash psql -U hr_bot_user -h localhost -d telegram_hr_bot` Если подключение успешно, введите `\q` для выхода
5. **Проверка через Python:** `python python -c "import psycopg2; conn = psycopg2.connect(host='localhost', database='telegram_hr_bot', user='hr_bot_user', password='secure_password_here'); print('Connection successful'); conn.close()"`

## Инициализация базы данных

Создадим структуру таблиц для нашего бота:

1. **Переход в папку проекта:** `bash cd Desktop/telegram-hr-bot`
2. **Активация виртуального окружения** (если не активировано): `bash venv\Scripts\activate`
3. **Запуск скрипта инициализации:** `python python -c " from src.main import app, db with app.app_context(): db.create_all() print('Database tables created successfully!') "`

## Проверка созданных таблиц

Убедимся, что все таблицы созданы корректно:

1. **Подключение к базе данных:** `bash psql -U hr_bot_user -h localhost -d telegram_hr_bot`
2. **Просмотр списка таблиц:** `sql \dt` Вы должны увидеть таблицы:
3. `users` - пользователи бота
4. `jobs` - вакансии
5. `applications` - отклики на вакансии
6. `subscriptions` - подписки на уведомления
7. **Просмотр структуры таблицы пользователей:** `sql \d users`
8. **Выход из psql:** `sql \q`

## Создание тестовых данных

Добавим несколько тестовых записей для проверки работы:

1. **Создание скрипта с тестовыми данными:** Создайте файл `test_data.py` в корне проекта: `python from src.main import app, db from src.models.user import User from src.models.job import Job from datetime import datetime`



```
with app.app_context(): # Создание тестового пользователя-работодателя
 employer = User(telegram_id=123456789, username='test_employer',
first_name='Тест', last_name='Работодатель', role='employer', is_active=True)
 db.session.add(employer)
```

```
 # Создание тестового пользователя-соискателя
 candidate = User(
 telegram_id=987654321,
 username='test_candidate',
 first_name='Тест',
 last_name='Соискатель',
 role='candidate',
 is_active=True
)
 db.session.add(candidate)

 # Создание тестовой вакансии
 job = Job(
 title='Python разработчик',
 description='Требуется опытный Python разработчик для работы над
интересными проектами.',
 company='ТестКомпания',
 location='Москва',
 salary_min=100000,
 salary_max=150000,
 employment_type='full_time',
 experience_level='middle',
 skills='Python, Flask, PostgreSQL',
 employer_id=123456789,
 is_active=True
)
 db.session.add(job)

 db.session.commit()
 print('Test data created successfully!')
```

...

1. **Запуск скрипта:** `bash python test_data.py`

## Настройка индексов для производительности

Создадим индексы для ускорения поиска:

1. **Подключение к базе данных:** `bash psql -U hr_bot_user -h localhost -d telegram_hr_bot`

2. **Создание индексов:** ```sql -- Индексы для таблицы jobs CREATE INDEX idx_jobs_location ON jobs(location); CREATE INDEX idx_jobs_employment_type ON jobs(employment_type); CREATE INDEX idx_jobs_experience_level ON`

```
jobs(experience_level); CREATE INDEX idx_jobs_salary_range ON
jobs(salary_min, salary_max); CREATE INDEX idx_jobs_created_at ON
jobs(created_at); CREATE INDEX idx_jobs_is_active ON jobs(is_active);
```

```
-- Индексы для таблицы users CREATE INDEX idx_users_telegram_id ON
users(telegram_id); CREATE INDEX idx_users_role ON users(role); CREATE INDEX
idx_users_is_active ON users(is_active);
```

```
-- Индексы для таблицы applications CREATE INDEX idx_applications_job_id ON
applications(job_id); CREATE INDEX idx_applications_candidate_id ON
applications(candidate_id); CREATE INDEX idx_applications_status ON
applications(status); CREATE INDEX idx_applications_created_at ON
applications(created_at);
```

```
-- Индексы для таблицы subscriptions CREATE INDEX idx_subscriptions_user_id ON
subscriptions(user_id); CREATE INDEX idx_subscriptions_is_active ON
subscriptions(is_active); ````
```

1. **Проверка созданных индексов:** `sql \di`

2. **Выход из psql:** `sql \q`

## Настройка резервного копирования

Настроим автоматическое резервное копирование базы данных:

1. **Создание папки для резервных копий:** `bash mkdir backups`

2. **Создание скрипта резервного копирования:** Создайте файл `backup_db.bat` : ````batch @echo off set PGPASSWORD=secure\_password\_here  
set BACKUP\_DIR=%~dp0backups set  
DATE=%date:~-4,4%%date:~-10,2%%date:~-7,2% set  
TIME=%time:~-0,2%%time:~-3,2%%time:~-6,2% set  
FILENAME=telegram\_hr\_bot\_%DATE%\_%TIME%.sql

```
"C:\Program Files\PostgreSQL\15\bin\pg_dump" -U hr_bot_user -h localhost
telegram_hr_bot > "%BACKUP_DIR%\%FILENAME%"
```

```
echo Backup created: %FILENAME% pause ````
```

1. **Тестирование резервного копирования:** `bash backup_db.bat`

# Мониторинг базы данных

Настроим базовый мониторинг для отслеживания состояния базы данных:

1. **Создание скрипта мониторинга:** Создайте файл `monitor_db.py`: ``python  
import psycopg2 from datetime import datetime import os from dotenv import  
load\_dotenv

load\_dotenv()

```
def check_database_health(): try: conn = psycopg2.connect(
host=os.getenv('POSTGRES_HOST'), database=os.getenv('POSTGRES_DB'),
user=os.getenv('POSTGRES_USER'), password=os.getenv('POSTGRES_PASSWORD'))
```

```
 cursor = conn.cursor()

 # Проверка подключения
 cursor.execute('SELECT version();')
 version = cursor.fetchone()
 print(f"PostgreSQL version: {version[0]}")

 # Статистика таблиц
 cursor.execute("""
 SELECT schemaname, tablename, n_tup_ins, n_tup_upd, n_tup_del
 FROM pg_stat_user_tables;
 """)

 print("\nTable statistics:")
 for row in cursor.fetchall():
 print(f" {row[1]}: {row[2]} inserts, {row[3]} updates, {row[4]}
deletes")

 # Размер базы данных
 cursor.execute("""
 SELECT pg_size_pretty(pg_database_size(current_database()));
 """)
 size = cursor.fetchone()
 print(f"\nDatabase size: {size[0]}")

 cursor.close()
 conn.close()

 print(f"\nDatabase health check completed at {datetime.now()}")
 return True

 except Exception as e:
 print(f"Database health check failed: {e}")
 return False
```

```
if name == "main": check_database_health() ``
```

1. **Запуск мониторинга:** `bash python monitor_db.py`

---

## Запуск и тестирование бота

---

Теперь, когда все компоненты настроены, пора запустить нашего HR бота и протестировать его функциональность.

### Предварительная проверка

Перед запуском убедимся, что все готово:

1. **Проверка переменных окружения:**

```
bash python -c " from dotenv import load_dotenv import os load_dotenv() print('Bot token:', 'OK' if os.getenv('TELEGRAM_BOT_TOKEN') else 'MISSING') print('DB config:', 'OK' if all([os.getenv('POSTGRES_HOST'), os.getenv('POSTGRES_DB')]) else 'MISSING') "
```
2. **Проверка подключения к базе данных:**

```
bash python -c " from src.main import app, db with app.app_context(): try: db.engine.execute('SELECT 1') print('Database connection: OK') except Exception as e: print(f'Database connection: FAILED - {e}') "
```
3. **Проверка импортов:**

```
bash python -c " try: from src.bot.telegram_bot import TelegramHRBot print('Bot imports: OK') except Exception as e: print(f'Bot imports: FAILED - {e}') "
```

### Первый запуск бота

Запустим бота в режиме разработки:

1. **Активация виртуального окружения:**

```
bash venv\Scripts\activate
```
2. **Запуск бота:**

```
bash python src/main.py
```
3. **Ожидаемый вывод:**

```
INFO:root:Инициализация Flask приложения...
INFO:root:Подключение к базе данных... INFO:root:Инициализация
Telegram-бота... INFO:root:Запуск планировщика уведомлений...
INFO:root:Запуск Telegram-бота... INFO:telebot:Starting polling.
```

4. Если бот запустился успешно, вы увидите сообщение о том, что polling начался

## Тестирование основных функций

Протестируем бота через Telegram:

### Тест 1: Базовые команды

1. Найдите вашего бота в Telegram
2. Отправьте `/start`
3. Ожидаемый результат: Приветственное сообщение с выбором роли
4. Выберите роль (работодатель или соискатель)
5. Отправьте `/menu`
6. Ожидаемый результат: Главное меню с кнопками

### Тест 2: Функции работодателя

1. Выберите роль "Работодатель"
2. Отправьте `/quick`
3. Нажмите "Создать вакансию"
4. Следуйте инструкциям бота для создания тестовой вакансии
5. Проверьте, что вакансия сохранилась: `/myjobs`

### Тест 3: Функции соискателя

1. Создайте второй аккаунт Telegram или попросите друга помочь
2. Выберите роль "Соискатель"
3. Отправьте `/jobs`
4. Проверьте поиск вакансий
5. Попробуйте откликнуться на вакансию

### Тест 4: Система уведомлений

1. Как соискатель, отправьте `/subscribe`

2. Создайте подписку на уведомления
3. Как работодатель, создайте новую вакансию
4. Проверьте, что соискатель получил уведомление

## Отладка и логирование

Если что-то работает не так, как ожидается:

1. **Проверка логов:**
2. Логи выводятся в консоль
3. Также сохраняются в папку `logs/`
4. Откройте файл `logs/bot.log` для детального анализа
5. **Увеличение уровня логирования:** В файле `src/main.py` измените: `python logging.basicConfig(level=logging.DEBUG)`
6. **Проверка базы данных:** `bash python monitor_db.py`
7. **Проверка конкретной таблицы:** `bash psql -U hr_bot_user -h localhost -d telegram_hr_bot -c "SELECT * FROM users LIMIT 5;"`

## Остановка бота

Для корректной остановки бота:

1. В консоли нажмите `ctrl + c`
2. Дождитесь сообщения о корректном завершении
3. Если бот не останавливается, нажмите `ctrl + c` еще раз

## Автоматический перезапуск

Для автоматического перезапуска при изменениях в коде:

1. Установите watchdog: `bash pip install watchdog`
2. Создайте скрипт `auto_restart.py`: `python import time import subprocess import sys from watchdog.observers import Observer from watchdog.events`

```
import FileSystemEventHandler
```

```
class BotRestartHandler(FileSystemEventHandler):
 def init(self):
 self.process = None
 self.start_bot()
```

```
 def start_bot(self):
 if self.process:
 self.process.terminate()
 self.process.wait()

 print("Starting bot...")
 self.process = subprocess.Popen([sys.executable, "src/main.py"])

 def on_modified(self, event):
 if event.src_path.endswith('.py'):
 print(f"File {event.src_path} modified, restarting bot...")
 self.start_bot()
```

```
if name == "main":
 event_handler = BotRestartHandler()
 observer = Observer()
 observer.schedule(event_handler, "src", recursive=True)
 observer.start()
```

```
 try:
 while True:
 time.sleep(1)
 except KeyboardInterrupt:
 observer.stop()
 if event_handler.process:
 event_handler.process.terminate()

 observer.join()
```

```
...
```

1. Запуск с автоперезапуском: `bash python auto_restart.py`



## Настройка GitHub синхронизации

Настройка синхронизации с GitHub позволит вам сохранять изменения в коде, работать в команде и автоматически деплоить обновления.

### Подготовка локального репозитория

Убедимся, что Git настроен корректно:

1. Проверка статуса Git: `bash git status`

2. **Добавление всех файлов:** `bash git add .`

3. **Проверка файлов для коммита:** `bash git status`

4. **Создание .gitignore** (если не существует): ````bash echo "# Python pycache/  
.py[cod] $py.class .so .Python build/ develop-eggs/ dist/ downloads/ eggs/ .eggs/  
lib/ lib64/ parts/ sdist/ var/ wheels/ .egg-info/ .installed.cfg *.egg`

# Virtual Environment venv/ env/ ENV/

# Environment Variables .env .env.local .env.production

# Database .db.sqlite3

# Logs logs/ \*.log

# Uploads uploads/

# IDE .vscode/ .idea/ .swp .swo

# OS .DS\_Store Thumbs.db

# Backups backups/ \*.bak" > .gitignore ```

## Первый коммит

Создадим первый коммит с нашим кодом:

1. **Добавление файлов:** `bash git add .`

2. **Создание коммита:** ````bash git commit -m "Initial commit: Telegram HR Bot  
with PostgreSQL`

Features: - Complete bot functionality for employers and candidates - PostgreSQL database integration - Advanced search and filtering - Notification system with subscriptions - Docker support for deployment - Comprehensive documentation" ```

1. **Проверка коммита:** `bash git log --oneline`

## Настройка удаленного репозитория

Подключим локальный репозиторий к GitHub:



1. **Добавление удаленного репозитория:** `bash git remote add origin git@github.com:ваш_username/telegram-hr-bot.git`
2. **Проверка удаленных репозитория:** `bash git remote -v`
3. **Отправка кода на GitHub:** `bash git push -u origin main`

## Настройка веток для разработки

Создадим структуру веток для организованной разработки:

1. **Создание ветки разработки:** `bash git checkout -b develop git push -u origin develop`
2. **Создание ветки для новых функций:** `bash git checkout -b feature/notifications-enhancement`
3. **Возврат к основной ветке:** `bash git checkout main`

## Настройка GitHub Actions для CI/CD

Создадим автоматический пайплайн для тестирования и деплоя:

1. **Создание папки для GitHub Actions:** `bash mkdir -p .github/workflows`
2. **Создание файла CI/CD пайплайна:** Создайте файл `.github/workflows/ci-cd.yml`: ```yaml name: CI/CD Pipeline`

on: push: branches: [ main, develop ] pull\_request: branches: [ main ]

jobs: test: runs-on: ubuntu-latest

```

services:
 postgres:
 image: postgres:15
 env:
 POSTGRES_PASSWORD: test_password
 POSTGRES_DB: test_db
 options: >-
 --health-cmd pg_isready
 --health-interval 10s
 --health-timeout 5s
 --health-retries 5
 ports:
 - 5432:5432

steps:
- uses: actions/checkout@v3

- name: Set up Python
 uses: actions/setup-python@v4
 with:
 python-version: '3.11'

- name: Install dependencies
 run: |
 python -m pip install --upgrade pip
 pip install -r requirements.txt
 pip install pytest pytest-cov

- name: Run tests
 env:
 POSTGRES_HOST: localhost
 POSTGRES_PORT: 5432
 POSTGRES_DB: test_db
 POSTGRES_USER: postgres
 POSTGRES_PASSWORD: test_password
 TELEGRAM_BOT_TOKEN: test_token
 FLASK_SECRET_KEY: test_secret_key
 run: |
 pytest tests/ --cov=src --cov-report=xml

- name: Upload coverage to Codecov
 uses: codecov/codecov-action@v3
 with:
 file: ./coverage.xml

build:
 needs: test
 runs-on: ubuntu-latest
 if: github.ref == 'refs/heads/main'

steps:
- uses: actions/checkout@v3

- name: Set up Docker Buildx
 uses: docker/setup-buildx-action@v2

- name: Login to Container Registry
 uses: docker/login-action@v2
 with:
 registry: cr.yandex
 username: json_key

```

```

 password: ${ secrets.YC_SERVICE_ACCOUNT_KEY }}

- name: Build and push Docker image
 uses: docker/build-push-action@v4
 with:
 context: .
 push: true
 tags: |
 cr.yandex/${ secrets.YC_REGISTRY_ID }}/hr-bot:latest
 cr.yandex/${ secrets.YC_REGISTRY_ID }}/hr-bot:${ secrets.github.sha }}

deploy:
 needs: build
 runs-on: ubuntu-latest
 if: github.ref == 'refs/heads/main'

 steps:
 - name: Deploy to Yandex Cloud
 uses: appleboy/ssh-action@v0.1.5
 with:
 host: ${ secrets.SERVER_HOST }}
 username: ${ secrets.SERVER_USER }}
 key: ${ secrets.SSH_PRIVATE_KEY }}
 script: |
 cd /opt/hr-bot
 sudo docker-compose pull
 sudo docker-compose up -d
 sudo docker system prune -f

```

...

## Настройка секретов GitHub

Добавим секретные переменные в GitHub:

1. **Переход в настройки репозитория:**
2. Откройте ваш репозиторий на GitHub
3. Нажмите "Settings"
4. В левом меню выберите "Secrets and variables" → "Actions"
5. **Добавление секретов:** Нажмите "New repository secret" и добавьте:
6. `YC_SERVICE_ACCOUNT_KEY` : Содержимое файла key.json от Яндекс.Облако
7. `YC_REGISTRY_ID` : ID Container Registry в Яндекс.Облако
8. `SERVER_HOST` : IP адрес вашего сервера
9. `SERVER_USER` : Имя пользователя на сервере (например, hr-bot)

10. `SSH_PRIVATE_KEY` : Приватный SSH ключ для доступа к серверу

## Создание Pull Request шаблона

Создадим шаблон для Pull Request:

1. **Создание папки для шаблонов:** `bash mkdir -p .github/PULL_REQUEST_TEMPLATE`

2. **Создание шаблона:** Создайте файл `.github/PULL_REQUEST_TEMPLATE/pull_request_template.md`: ````markdown`  
`## Описание изменений`

Краткое описание того, что было изменено и почему.

`## Тип изменений`

- ☐ Исправление ошибки (bug fix)
- ☐ Новая функциональность (feature)
- ☐ Критическое изменение (breaking change)
- ☐ Обновление документации
- ☐ Рефакторинг кода
- ☐ Улучшение производительности

`## Тестирование`

- ☐ Код протестирован локально
- ☐ Добавлены новые тесты
- ☐ Все существующие тесты проходят
- ☐ Протестировано в Telegram

`## Чеклист`

- ☐ Код соответствует стандартам проекта
- ☐ Добавлена документация для новых функций
- ☐ Обновлен CHANGELOG.md
- ☐ Проверена совместимость с базой данных

## Скриншоты (если применимо)

Добавьте скриншоты интерфейса бота, если изменения затрагивают UI.

## Дополнительная информация

Любая дополнительная информация, которая может быть полезна для ревьюера. ``

## Настройка автоматической синхронизации

Создадим скрипт для автоматической синхронизации изменений:

1. **Создание скрипта sync.bat:** `` batch @echo off echo Синхронизация с GitHub...

```
git add .
```

```
set /p commit_message="Введите сообщение коммита: "
```

```
git commit -m "%commit_message%"
```

```
git push origin main
```

```
echo Синхронизация завершена! pause ``
```

1. **Создание скрипта для быстрого обновления:** `` batch @echo off echo  
Получение обновлений с GitHub...

```
git fetch origin git pull origin main
```

```
echo Активация виртуального окружения... call venv\Scripts\activate
```

```
echo Обновление зависимостей... pip install -r requirements.txt
```

```
echo Обновление завершено! pause ``
```

---

## Подготовка к деплою на Яндекс.Облако

---

Подготовим проект для развертывания в продакшен среде на Яндекс.Облако.

## Установка Yandex Cloud CLI

### 1. Загрузка CLI:

2. Перейдите на <https://cloud.yandex.ru/docs/cli/quickstart>

3. Скачайте установщик для Windows

4. Запустите установщик и следуйте инструкциям

5. **Инициализация CLI:** `bash yc init` Следуйте инструкциям для авторизации и настройки.

6. **Проверка установки:** `bash yc config list`

## Подготовка Docker образа

Оптимизируем Dockerfile для продакшена:

1. **Создание .dockerignore:** `.git .gitignore README.md Dockerfile .dockerignore venv/ __pycache__/ *.pyc *.pyo *.pyd .Python env/ pip-log.txt pip-delete-this-directory.txt .tox .coverage .coverage.* .cache nosetests.xml coverage.xml *.cover *.log .idea/ .vscode/ .DS_Store .env backups/ logs/ uploads/`

2. **Оптимизация Dockerfile:** Обновите Dockerfile для продакшена:  
````dockerfile # Multi-stage build для уменьшения размера образа FROM python:3.11-slim as builder`

```
# Установка системных зависимостей для сборки RUN apt-get update && apt-get install -y \ gcc \ libpq-dev \ && rm -rf /var/lib/apt/lists/*
```

```
# Создание виртуального окружения RUN python -m venv /opt/venv ENV PATH="/opt/venv/bin:$PATH"
```

```
# Копирование и установка зависимостей COPY requirements.txt . RUN pip install --no-cache-dir --upgrade pip && \ pip install --no-cache-dir -r requirements.txt
```

```
# Продакшен образ FROM python:3.11-slim
```

```
# Установка только runtime зависимостей RUN apt-get update && apt-get install -y \ libpq5 \ curl \ && rm -rf /var/lib/apt/lists/* \ && useradd --create-home --shell /bin/bash app
```

```
# Копирование виртуального окружения из builder COPY --from=builder /opt/venv
/opt/venv ENV PATH="/opt/venv/bin:$PATH"

# Установка рабочей директории WORKDIR /app

# Копирование кода приложения COPY src/ ./src/ COPY .env.production .env

# Создание необходимых директорий RUN mkdir -p logs uploads && \ chown -R
app:app /app

# Переключение на пользователя приложения USER app

# Настройка переменных окружения ENV PYTHONPATH=/app ENV
FLASK_APP=src/main.py ENV FLASK_ENV=production ENV PYTHONUNBUFFERED=1

# Проверка здоровья HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --
retries=3 \ CMD curl -f http://localhost:5000/api/health || exit 1

# Открытие порта EXPOSE 5000

# Команда запуска CMD ["python", "src/main.py"] ````
```

Создание продакшен конфигурации

1. Создание .env.production: ````env # Production Environment Configuration

```
# Telegram Bot TELEGRAM_BOT_TOKEN=${TELEGRAM_BOT_TOKEN}

# Database (Managed PostgreSQL) POSTGRES_HOST=${POSTGRES_HOST}
POSTGRES_PORT=6432 POSTGRES_DB=telegram_hr_bot
POSTGRES_USER=${POSTGRES_USER}
POSTGRES_PASSWORD=${POSTGRES_PASSWORD} POSTGRES_SSLMODE=require

# Flask FLASK_ENV=production FLASK_DEBUG=False
FLASK_SECRET_KEY=${FLASK_SECRET_KEY}

# Security SESSION_COOKIE_SECURE=True SESSION_COOKIE_HTTPONLY=True
SESSION_COOKIE_SAMESITE=Lax

# Logging LOG_LEVEL=INFO LOG_TO_FILE=True

# Performance WORKERS=4 MAX_CONTENT_LENGTH=16777216
```

```
# Monitoring ENABLE_METRICS=True HEALTH_CHECK_ENABLED=True
```

```
# Notifications NOTIFICATION_ENABLED=True NOTIFICATION_BATCH_SIZE=100 ``
```

1. Создание **docker-compose.production.yml**: ``yaml version: '3.8'

```
services: app: image: cr.yandex/${YC_REGISTRY_ID}/hr-bot:latest container_name:
hr_bot_app restart: unless-stopped env_file: .env.production ports: - "5000:5000"
volumes: - ./logs:/app/logs - ./uploads:/app/uploads healthcheck: test: ["CMD", "curl",
"-f", "http://localhost:5000/api/health"] interval: 30s timeout: 10s retries: 3
start_period: 40s logging: driver: "json-file" options: max-size: "10m" max-file: "3"
```

```
nginx:
  image: nginx:alpine
  container_name: hr_bot_nginx
  restart: unless-stopped
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
    - ./ssl:/etc/nginx/ssl:ro
    - ./logs/nginx:/var/log/nginx
  depends_on:
    app:
      condition: service_healthy
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"
```

```
``
```

Создание Nginx конфигурации

Создайте файл `nginx.conf`:


```

events {
    worker_connections 1024;
}

http {
    upstream app {
        server app:5000;
    }

    # Rate limiting
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;

    # Logging
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;
    error_log /var/log/nginx/error.log warn;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css application/json application/javascript
    text/xml application/xml application/xml+rss text/javascript;

    server {
        listen 80;
        server_name _;

        # Security headers
        add_header X-Frame-Options DENY;
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";
        add_header Strict-Transport-Security "max-age=31536000;
includeSubDomains" always;

        # Rate limiting
        limit_req zone=api burst=20 nodelay;

        # Health check endpoint
        location /health {
            access_log off;
            return 200 "healthy\n";
            add_header Content-Type text/plain;
        }

        # Main application
        location / {
            proxy_pass http://app;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;

            # Timeouts
            proxy_connect_timeout 30s;
            proxy_send_timeout 30s;
            proxy_read_timeout 30s;
        }
    }
}

```

```

    # Buffer settings
    proxy_buffering on;
    proxy_buffer_size 4k;
    proxy_buffers 8 4k;
}

# Static files (if any)
location /static/ {
    alias /app/static/;
    expires 1y;
    add_header Cache-Control "public, immutable";
}
}

```

Создание скриптов деплоя

1. **Скрипт сборки образа** (`build.bat`): ```batch @echo off echo Сборка Docker образа для продакшена...

```
set /p version="Введите версию (например, 1.0.0): "
```

```
docker build -t hr-bot:%version% . docker tag hr-bot:%version% hr-bot:latest
```

```
echo Образ собран: hr-bot:%version% pause ```
```

1. **Скрипт деплоя** (`deploy.bat`): ```batch @echo off echo Деплой на Яндекс.Облако...

```
echo Проверка конфигурации Yandex Cloud CLI... yc config list
```

```
echo Сборка и загрузка образа... docker build -t cr.yandex/%YC_REGISTRY_ID%/hr-bot:latest . docker push cr.yandex/%YC_REGISTRY_ID%/hr-bot:latest
```

```
echo Деплой на сервер... ssh hr-bot@%SERVER_HOST% "cd /opt/hr-bot && sudo docker-compose -f docker-compose.production.yml pull && sudo docker-compose -f docker-compose.production.yml up -d"
```

```
echo Деплой завершен! pause ```
```

Мониторинг и логирование

1. **Создание скрипта мониторинга** (`monitor.py`): ```python import requests
import time import logging from datetime import datetime

```
# Настройка логирования logging.basicConfig( level=logging.INFO, format='%  
(asctime)s          -          %(levelname)s          -          %(message)s',          handlers=[  
logging.FileHandler('monitoring.log'), logging.StreamHandler() ] )
```

```
def check_health(url): try: response = requests.get(f"{url}/api/health", timeout=10) if  
response.status_code == 200: logging.info(f"Health check OK: {response.text.strip()}")  
return True else: logging.error(f"Health check failed: HTTP {response.status_code}")  
return False except Exception as e: logging.error(f"Health check error: {e}") return  
False
```









```
def monitor_service(url, interval=60): logging.info(f"Starting monitoring for {url}")
```

```
while True:  
    if not check_health(url):  
        # Отправка уведомления об ошибке  
        logging.critical("Service is DOWN!")  
        # Здесь можно добавить отправку уведомления в Telegram  
  
    time.sleep(interval)
```

```
if name == "main": SERVICE_URL = "https://your-domain.com" # Замените на ваш  
домен monitor_service(SERVICE_URL) ``
```

Финальная проверка готовности

Перед деплоем убедитесь, что:

1. **Все файлы созданы:**
2.  Dockerfile оптимизирован
3.  .env.production настроен
4.  docker-compose.production.yml создан
5.  nginx.conf настроен
6.  GitHub Actions настроены
7. **Секреты настроены:**
8.  GitHub Secrets добавлены
9.  Yandex Cloud CLI настроен
10.  SSH ключи настроены

11. Тестирование:

- 12. ☒ Локальный Docker образ собирается
- 13. ☒ Бот работает в продакшен режиме
- 14. ☒ База данных подключается

Теперь ваш проект готов к деплою на Яндекс.Облако! Следуйте инструкциям в файле `yandex_cloud_deploy.md` для завершения процесса развертывания.

Устранение проблем

В этом разделе собраны решения наиболее частых проблем, с которыми вы можете столкнуться при настройке и запуске HR бота.

Проблемы с Python

Проблема: "python не является внутренней или внешней командой"

Причина: Python не добавлен в PATH или установлен некорректно.

Решение: 1. Переустановите Python, обязательно поставив галочку "Add Python to PATH" 2. Или добавьте Python в PATH вручную: - Найдите папку установки Python (обычно `C:\Users\[имя]\AppData\Local\Programs\Python\Python311\`) - Добавьте эту папку и подпапку `Scripts` в переменную PATH

Проблема: "pip install завершается с ошибкой"

Причина: Проблемы с сетью, правами доступа или версией pip.

Решение:

```
# Обновите pip
python -m pip install --upgrade pip

# Используйте другой индекс пакетов
pip install --index-url https://pypi.org/simple/ package_name

# Установка с правами администратора (только если необходимо)
pip install --user package_name
```

Проблема: "ModuleNotFoundError" при импорте

Причина: Модуль не установлен или виртуальное окружение не активировано.

Решение:

```
# Активируйте виртуальное окружение
venv\Scripts\activate

# Установите недостающий модуль
pip install module_name

# Проверьте установленные пакеты
pip list
```

Проблемы с PostgreSQL

Проблема: "psql: error: connection to server failed"

Причина: PostgreSQL не запущен или неверные параметры подключения.

Решение: 1. Проверьте статус службы PostgreSQL: - Win + R → services.msc - Найдите "postgresql-x64-15" и убедитесь, что статус "Выполняется"

1. Проверьте параметры подключения в .env файле
2. Попробуйте подключиться напрямую: `bash psql -U hr_bot_user -h localhost -d telegram_hr_bot`

Проблема: "FATAL: password authentication failed"

Причина: Неверный пароль или пользователь не существует.

Решение: 1. Подключитесь как суперпользователь: `bash psql -U postgres -h localhost`

1. Сбросьте пароль пользователя: `sql ALTER USER hr_bot_user PASSWORD 'новый_пароль' ;`
2. Обновите пароль в .env файле

Проблема: "database does not exist"

Причина: База данных не создана.

Решение:

```
# Создайте базу данных
createdb -U hr_bot_user -h localhost telegram_hr_bot

# Или через psql
psql -U postgres -h localhost
CREATE DATABASE telegram_hr_bot OWNER hr_bot_user;
```

Проблемы с Telegram Bot

Проблема: "Unauthorized" при запуске бота

Причина: Неверный токен бота.

Решение: 1. Проверьте токен в `.env` файле 2. Убедитесь, что токен скопирован полностью без лишних пробелов 3. Создайте новый токен через @BotFather если необходимо

Проблема: Бот не отвечает на сообщения

Причина: Ошибки в коде, проблемы с базой данных или сетью.

Решение: 1. Проверьте логи в консоли и файле `logs/bot.log` 2. Убедитесь, что база данных доступна 3. Проверьте интернет-соединение 4. Перезапустите бота

Проблема: "Conflict: terminated by other getUpdates request"

Причина: Несколько экземпляров бота запущено одновременно.

Решение: 1. Остановите все экземпляры бота 2. Подождите 1-2 минуты 3. Запустите только один экземпляр

Проблемы с Git и GitHub

Проблема: "Permission denied (publickey)"

Причина: SSH ключ не настроен или неверно сконфигурирован.

Решение: 1. Проверьте SSH ключ: `bash ssh -T git@github.com`

1. Если ключа нет, создайте новый: `bash ssh-keygen -t ed25519 -C "your.email@example.com"`

2. Добавьте ключ в GitHub через веб-интерфейс

Проблема: "fatal: not a git repository"

Причина: Git не инициализирован в папке проекта.

Решение:

```
git init
git remote add origin git@github.com:username/repo.git
```

Проблемы с Docker

Проблема: "docker: command not found"

Причина: Docker не установлен или не добавлен в PATH.

Решение: 1. Установите Docker Desktop для Windows 2. Перезапустите компьютер 3. Убедитесь, что Docker запущен

Проблема: "Cannot connect to the Docker daemon"

Причина: Docker daemon не запущен.

Решение: 1. Запустите Docker Desktop 2. Подождите полной загрузки 3. Проверьте статус: `docker version`

Проблемы с сетью и портами

Проблема: "Port 5000 is already in use"

Причина: Порт занят другим приложением.

Решение: 1. Найдите процесс, использующий порт: `cmd netstat -ano | findstr :5000`

1. Завершите процесс: `cmd taskkill /PID [номер_процесса] /F`

2. Или измените порт в настройках приложения

Проблемы с производительностью

Проблема: Медленная работа бота

Причина: Неоптимизированные запросы к базе данных или недостаток ресурсов.

Решение: 1. Проверьте индексы в базе данных 2. Оптимизируйте SQL запросы 3. Увеличьте ресурсы сервера 4. Используйте кэширование

Проблема: Высокое потребление памяти

Причина: Утечки памяти или неэффективный код.

Решение: 1. Мониторьте использование памяти: `python import psutil`
`print(f"Memory usage: {psutil.virtual_memory().percent}%")`

1. Используйте профилировщик памяти
2. Оптимизируйте код и закрывайте соединения

Диагностические команды

Полезные команды для диагностики проблем:

```
# Проверка версий
python --version
pip --version
git --version
psql --version

# Проверка сервисов
netstat -an | findstr :5432 # PostgreSQL
netstat -an | findstr :5000 # Flask app

# Проверка процессов
tasklist | findstr python
tasklist | findstr postgres

# Проверка дискового пространства
dir C:\ /-c

# Проверка переменных окружения
echo %PATH%
set | findstr PYTHON
```


Логирование для отладки

Включите детальное логирование для диагностики:

```
import logging

# В начале main.py добавьте:
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('debug.log'),
        logging.StreamHandler()
    ]
)
```



Дополнительные возможности

В этом разделе описаны дополнительные функции и улучшения, которые можно добавить к вашему HR боту для расширения его возможностей.

Интеграция с внешними сервисами

Интеграция с HeadHunter API

Добавьте возможность импорта вакансий с HeadHunter:

```

import requests

class HHIntegration:
    def __init__(self):
        self.base_url = "https://api.hh.ru"

    def search_vacancies(self, text, area=1): # area=1 для Москвы
        url = f"{self.base_url}/vacancies"
        params = {
            'text': text,
            'area': area,
            'per_page': 20
        }

        response = requests.get(url, params=params)
        return response.json()

    def import_vacancy(self, hh_vacancy_id):
        url = f"{self.base_url}/vacancies/{hh_vacancy_id}"
        response = requests.get(url)
        vacancy_data = response.json()

        # Преобразование данных HH в формат нашей базы
        return {
            'title': vacancy_data['name'],
            'description': vacancy_data['description'],
            'company': vacancy_data['employer']['name'],
            'location': vacancy_data['area']['name'],
            'salary_min': vacancy_data['salary']['from'] if
vacancy_data['salary'] else None,
            'salary_max': vacancy_data['salary']['to'] if
vacancy_data['salary'] else None,
        }

```

Интеграция с Telegram Payments

Добавьте возможность платных услуг:

```

from telebot.types import LabeledPrice, InlineKeyboardMarkup,
InlineKeyboardButton

def create_premium_subscription_invoice(bot, chat_id):
    prices = [LabeledPrice(label='Премиум подписка на месяц', amount=99900)] #
    в копейках

    bot.send_invoice(
        chat_id,
        title='Премиум подписка HR Bot',
        description='Получите доступ к расширенным функциям: приоритетная
поддержка, дополнительные фильтры поиска, аналитика',
        provider_token='YOUR_PAYMENT_PROVIDER_TOKEN',
        currency='rub',
        prices=prices,
        start_parameter='premium-subscription',
        invoice_payload='premium_subscription_payload'
    )

@bot.pre_checkout_query_handler(func=lambda query: True)
def checkout(pre_checkout_query):
    bot.answer_pre_checkout_query(pre_checkout_query.id, ok=True)

@bot.message_handler(content_types=['successful_payment'])
def successful_payment(message):
    # Активация премиум подписки
    user = User.query.filter_by(telegram_id=message.from_user.id).first()
    user.is_premium = True
    user.premium_until = datetime.now() + timedelta(days=30)
    db.session.commit()

    bot.send_message(message.chat.id, "Спасибо за покупку! Премиум функции
активированы.")

```

Аналитика и отчеты

Система аналитики для работодателей

```
class AnalyticsService:
    def __init__(self, db):
        self.db = db

    def get_employer_stats(self, employer_id):
        # Статистика по вакансиям
        total_jobs = Job.query.filter_by(employer_id=employer_id).count()
        active_jobs = Job.query.filter_by(employer_id=employer_id,
is_active=True).count()

        # Статистика по откликам
        applications = db.session.query(Application).join(Job).filter(
            Job.employer_id == employer_id
        ).all()

        total_applications = len(applications)
        new_applications = len([a for a in applications if a.status == 'new'])

        # Популярные навыки
        skills_stats = self.get_popular_skills(employer_id)

        return {
            'total_jobs': total_jobs,
            'active_jobs': active_jobs,
            'total_applications': total_applications,
            'new_applications': new_applications,
            'popular_skills': skills_stats
        }

    def generate_report(self, employer_id, period='month'):
        stats = self.get_employer_stats(employer_id)

        report = f"""
📊 Отчет за {period}

📋 Вакансии:
• Всего: {stats['total_jobs']}
• Активных: {stats['active_jobs']}

👥 Отклики:
• Всего: {stats['total_applications']}
• Новых: {stats['new_applications']}

🔥 Популярные навыки:
{chr(10).join([f"• {skill}: {count}" for skill, count in
stats['popular_skills'][:5]])}
"""

        return report
```

Экспорт данных в Excel

```
import pandas as pd
from openpyxl import Workbook
from openpyxl.styles import Font, PatternFill

def export_applications_to_excel(job_id):
    applications = Application.query.filter_by(job_id=job_id).all()

    data = []
    for app in applications:
        data.append({
            'Дата отклика': app.created_at.strftime('%d.%m.%Y %H:%M'),
            'Кандидат': f"{app.candidate.first_name} {app.candidate.last_name}",
            'Username': app.candidate.username,
            'Сопроводительное письмо': app.cover_letter,
            'Статус': app.status,
            'Контакты': app.contact_info
        })

    df = pd.DataFrame(data)

    # Создание Excel файла с форматированием
    wb = Workbook()
    ws = wb.active
    ws.title = "Отклики на вакансию"

    # Заголовки
    headers = list(df.columns)
    for col, header in enumerate(headers, 1):
        cell = ws.cell(row=1, column=col, value=header)
        cell.font = Font(bold=True)
        cell.fill = PatternFill(start_color="CCCCCC", end_color="CCCCCC",
                                fill_type="solid")

    # Данные
    for row, record in enumerate(data, 2):
        for col, value in enumerate(record.values(), 1):
            ws.cell(row=row, column=col, value=value)

    # Автоширина колонок
    for column in ws.columns:
        max_length = 0
        column_letter = column[0].column_letter
        for cell in column:
            try:
                if len(str(cell.value)) > max_length:
                    max_length = len(str(cell.value))
            except:
                pass
        adjusted_width = min(max_length + 2, 50)
        ws.column_dimensions[column_letter].width = adjusted_width

    filename =
    f"applications_job_{job_id}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.xlsx"
    wb.save(filename)

    return filename
```

Машинное обучение и ИИ

Рекомендательная система

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

class RecommendationEngine:
    def __init__(self):
        self.vectorizer = TfidfVectorizer(stop_words='english',
max_features=1000)
        self.job_vectors = None
        self.jobs_data = None

    def train(self, jobs):
        # Подготовка текстовых данных
        job_texts = []
        self.jobs_data = []

        for job in jobs:
            text = f"{job.title} {job.description} {job.skills} {job.location}"
            job_texts.append(text)
            self.jobs_data.append({
                'id': job.id,
                'title': job.title,
                'company': job.company,
                'location': job.location
            })

        # Векторизация
        self.job_vectors = self.vectorizer.fit_transform(job_texts)

    def recommend_jobs(self, user_profile, num_recommendations=5):
        # Создание профиля пользователя
        user_text = f"{user_profile.get('skills', '')}
{user_profile.get('experience', '')} {user_profile.get('interests', '')}"
        user_vector = self.vectorizer.transform([user_text])

        # Вычисление сходства
        similarities = cosine_similarity(user_vector,
self.job_vectors).flatten()

        # Получение топ рекомендаций
        top_indices = np.argsort(similarities)[::-1][:num_recommendations]

        recommendations = []
        for idx in top_indices:
            job_data = self.jobs_data[idx]
            job_data['similarity'] = similarities[idx]
            recommendations.append(job_data)

        return recommendations
```

Автоматическая категоризация вакансий

```
import re
from collections import Counter

class JobCategorizer:
    def __init__(self):
        self.categories = {
            'IT': ['python', 'java', 'javascript', 'react', 'node.js', 'sql',
                  'git', 'docker'],
            'Marketing': ['seo', 'smm', 'контекстная реклама', 'аналитика',
                          'google analytics'],
            'Sales': ['продажи', 'менеджер по продажам', 'клиенты', 'crm',
                     'переговоры'],
            'Design': ['дизайн', 'photoshop', 'figma', 'ui/ux', 'графический
дизайн'],
            'Finance': ['финансы', 'бухгалтерия', '1с', 'налоги', 'отчетность']
        }

    def categorize_job(self, job_text):
        job_text_lower = job_text.lower()
        category_scores = {}

        for category, keywords in self.categories.items():
            score = 0
            for keyword in keywords:
                if keyword in job_text_lower:
                    score += 1
            category_scores[category] = score

        # Возвращаем категорию с наибольшим счетом
        if max(category_scores.values()) > 0:
            return max(category_scores, key=category_scores.get)
        else:
            return 'Другое'

    def extract_skills(self, job_text):
        # Простое извлечение навыков на основе ключевых слов
        all_skills = []
        for skills_list in self.categories.values():
            all_skills.extend(skills_list)

        found_skills = []
        job_text_lower = job_text.lower()

        for skill in all_skills:
            if skill in job_text_lower:
                found_skills.append(skill)

        return found_skills
```

Интеграция с мессенджерами

Поддержка WhatsApp Business API

```
import requests

class WhatsAppIntegration:
    def __init__(self, access_token, phone_number_id):
        self.access_token = access_token
        self.phone_number_id = phone_number_id
        self.base_url = "https://graph.facebook.com/v17.0"

    def send_message(self, to_phone, message):
        url = f"{self.base_url}/{self.phone_number_id}/messages"

        headers = {
            'Authorization': f'Bearer {self.access_token}',
            'Content-Type': 'application/json'
        }

        data = {
            'messaging_product': 'whatsapp',
            'to': to_phone,
            'text': {'body': message}
        }

        response = requests.post(url, headers=headers, json=data)
        return response.json()

    def send_job_notification(self, phone, job):
        message = f"""
🔔 Новая вакансия!

📋 {job.title}
🏢 {job.company}
📍 {job.location}
💰 {job.salary_min}-{job.salary_max} руб.

Подробности: {job.description[:100]}...
"""

        return self.send_message(phone, message)
```


Расширенная безопасность

Система антиспама

```
from datetime import datetime, timedelta
import redis

class AntiSpamSystem:
    def __init__(self, redis_client):
        self.redis = redis_client
        self.limits = {
            'messages_per_minute': 10,
            'jobs_per_hour': 5,
            'applications_per_day': 20
        }

    def check_rate_limit(self, user_id, action_type):
        key = f"rate_limit:{user_id}:{action_type}"
        current_time = datetime.now()

        if action_type == 'messages':
            window = 60 # секунд
            limit = self.limits['messages_per_minute']
        elif action_type == 'jobs':
            window = 3600 # секунд
            limit = self.limits['jobs_per_hour']
        elif action_type == 'applications':
            window = 86400 # секунд
            limit = self.limits['applications_per_day']

        # Проверка текущего количества действий
        current_count = self.redis.get(key)
        if current_count is None:
            current_count = 0
        else:
            current_count = int(current_count)

        if current_count >= limit:
            return False, f"Превышен лимит: {limit} {action_type} в указанный период"

        # Увеличение счетчика
        pipe = self.redis.pipeline()
        pipe.incr(key)
        pipe.expire(key, window)
        pipe.execute()

        return True, "OK"

    def is_spam_content(self, text):
        spam_patterns = [
            r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',
            r'@[a-zA-Z0-9_]+',
            r'telegram.me',
            r't.me',
            r'whatsapp',
            r'viber'
        ]
```

```
for pattern in spam_patterns:
    if re.search(pattern, text, re.IGNORECASE):
        return True

return False
```

Мониторинг и метрики

Система метрик с Prometheus

```
from prometheus_client import Counter, Histogram, Gauge, start_http_server
import time

# Метрики
message_counter = Counter('bot_messages_total', 'Total messages processed',
['message_type'])
response_time = Histogram('bot_response_time_seconds', 'Response time for bot
operations')
active_users = Gauge('bot_active_users', 'Number of active users')
job_applications = Counter('job_applications_total', 'Total job applications',
['status'])

class MetricsCollector:
    def __init__(self):
        # Запуск HTTP сервера для метрик
        start_http_server(8000)

    def record_message(self, message_type):
        message_counter.labels(message_type=message_type).inc()

    def record_response_time(self, duration):
        response_time.observe(duration)

    def update_active_users(self, count):
        active_users.set(count)

    def record_application(self, status):
        job_applications.labels(status=status).inc()

# Декоратор для измерения времени выполнения
def measure_time(metrics_collector):
    def decorator(func):
        def wrapper(*args, **kwargs):
            start_time = time.time()
            result = func(*args, **kwargs)
            duration = time.time() - start_time
            metrics_collector.record_response_time(duration)
            return result
        return wrapper
    return decorator
```

Многоязычная поддержка

Система интернационализации

```
import json
import os

class I18n:
    def __init__(self, default_language='ru'):
        self.default_language = default_language
        self.translations = {}
        self.load_translations()

    def load_translations(self):
        translations_dir = 'translations'
        for filename in os.listdir(translations_dir):
            if filename.endswith('.json'):
                lang_code = filename[:-5] # убираем .json
                with open(os.path.join(translations_dir, filename), 'r',
encoding='utf-8') as f:
                    self.translations[lang_code] = json.load(f)

    def get_user_language(self, user_id):
        # Получение языка пользователя из базы данных
        user = User.query.filter_by(telegram_id=user_id).first()
        return user.language if user and user.language else
self.default_language

    def t(self, key, user_id=None, **kwargs):
        language = self.get_user_language(user_id) if user_id else
self.default_language

        if language not in self.translations:
            language = self.default_language

        translation = self.translations[language].get(key, key)

        # Подстановка параметров
        for param, value in kwargs.items():
            translation = translation.replace(f'{{{param}}}', str(value))

        return translation

# Пример файла translations/en.json
"""
{
    "welcome_message": "Welcome to HR Bot! Choose your role:",
    "employer_role": "Employer",
    "candidate_role": "Job Seeker",
    "job_created": "Job '{title}' has been created successfully!",
    "application_sent": "Your application has been sent to the employer."
}
"""

# Пример файла translations/ru.json
"""
{
    "welcome_message": "Добро пожаловать в HR Bot! Выберите вашу роль:",
    "employer_role": "Работодатель",
```

```
"candidate_role": "Соискатель",  
"job_created": "Вакансия '{title}' успешно создана!",  
"application_sent": "Ваш отклик отправлен работодателю."  
}  
"""
```

Заключение

Эти дополнительные возможности значительно расширяют функциональность вашего HR бота и делают его более конкурентоспособным на рынке. Вы можете реализовывать их поэтапно, начиная с наиболее важных для ваших пользователей функций.

Помните, что каждое новое дополнение требует тщательного тестирования и может повлиять на производительность системы. Рекомендуется внедрять новые функции постепенно и мониторить их влияние на общую работу бота.

Поддержка и сообщество

Если у вас возникли вопросы или проблемы при следовании этой инструкции:

1. **Проверьте раздел "Устранение проблем"** - там собраны решения наиболее частых проблем
2. **Изучите логи** - они содержат детальную информацию об ошибках
3. **Обратитесь к документации** используемых технологий
4. **Создайте Issue** в GitHub репозитории проекта

Удачи в создании вашего HR бота! 🚀

Документ создан Manus AI специально для максимально подробного и понятного внедрения Telegram HR Bot на Windows.