Санкт–Петербургский государственный университет

Криворучко Денис Игоревич

Отчет по учебной практике

Разработка Telegram-бота для автоматизации анализа конкурсных списков при поступлении

Уровень образования: бакалавриат
Направление 02.03.03 «Математическое обеспечение и администрирование информационных систем»
Основная образовательная программа СВ.5006.2019 «Математическое обеспечение и администрирование информационных систем»

Научный руководитель:
доцент каф. СП, к.т.н. Т.А. Брыксин

Санкт-Петербург
2020 г.

Содержание

Глава 1. Введение

Поступление - важная ступень в жизни каждого выпускника 11-го класса, успешно окончившего школу. Через это прошла огромная часть населения России. Некоторым данный этап предстоит преодолеть в будущем. Так, на сегодняшний день в университетах страны обучается приблизительно 4 миллионов человек [1]. А около 500 тысяч стали студентами в 2020 году [1].

Но сам процесс поступления является очень сложным и запутанным. Например, абитуриент бакалавриата может подать документы в 3 вуза, на 5 направлений в каждом, которые, в свою очередь, делятся на образовательные программы. Итого, человек имеет право участвовать в 20 или даже в 25 конкурсах. Причем ситуация в ранжированных списках меняется очень динамично. Одни люди приносят документы, другие забирают, чтобы подать в другое место.

Таким образом, возникает проблема анализа своего положения в конкурсе на поступление. Из-за постоянного обновления участников, абитуриенты вынуждены непрерывно мониторить таблицы поступающих, что является непростым занятием, если мы говорим о десятках списков, за которыми нужно следить.

Это мотивирует на создание разнообразных программ, призванных решить некоторые проблемы, связанные с поступлением в университет. Цель данного проекта — разработка приложения, способного помочь абитуриентам в анализе конкурсных списков путем автоматизации некоторых рутинных действий.

В ходе разработки были поставлены следующие задачи:

1) разработка архитектуры приложения;
2) проектирование базы данных;
3) реализация модуля взаимодействия с Telegram API;
4) реализация модуля-посредника, связывающего остальные модули;
5) реализация модуля получения данных с Web-сайтов университетов;

6) реализация обработки данных, полученных с сайтов университетов;

7) реализация модуля взаимодействия с базой данных;

8) логгирование;

9) тестирование;

10) отправка приложения на хостинг

Практическая значимость проекта состоит в решении одной из основных проблем многих абитуриентов в момент поступления: невозможности анализа большого количества конкурсных списков. Это позволяет будущим студентам трезво оценивать текущую ситуацию, при этом не отвлекаясь на постоянный мониторинг сайтов, на которых размещают актуальную информацию о приеме.

Глава 2. Обзор существующих решений

Для решения подобного рода проблем уже создаются некоторые специализированные инструменты.

Первое, что стоит отметить: сейчас абитуриентам не нужно лично приходить в приемную комиссию университета или каким-то образом связываться с ее представителями для того, чтобы узнать, какое место он сейчас занимает в конкурсных списках. Вся актуальная информация публикуется на сайте конкретного вуза и обновляется с адекватной частотой. Это, действительно, намного упростило процесс поступления. Сейчас абитуриент может оценивать свои шансы сразу в нескольких таблицах, при этом не выходя из собственного дома. Однако же, тут все еще нужно активное участие абитуриента на протяжении всей приемной кампании.

Вторым средством, которым пользуются абитуриенты, являются всевозможные агрегаторы вузов. Они собирают все нужные данные нескольких университетов в одном месте, позволяя анализировать ситуацию на всех интересующих образовательных программах, при этом используя только приложение-агрегатор. Одним из наиболее популярных таких инструментов является сайт Admlist [2]. Действительно, это значительно уменьшает сложность мониторинга изменений ранжированных списков поступающих, но не решает ее. Во-первых, данный инструмент предназначается не для этой цели, решает немного другие задачи. Во-вторых, все еще абитуриент должен сам заходить на страницу с нужной таблицей и следить за своим положением в ней. А значит, это не решает поставленной цели — автоматизации данного процесса.

Как можно заметить, все популярные инструменты, предназначенные для упрощения процесса поступления, либо не решают проблему со сложностью одновременного анализа большого количества постоянно обновляемых конкурсных списков, возможно, по причине того, что предназначены не совсем для этого, либо решают ее лишь частично. Данный проект, в свою очередь, полностью автоматизирует процесс выявления изменений в таблице, позволяя абитуриенту не следить за ней на протяжении всей приемной кампании.

Глава 3. Обзор используемых инструментов

При создании приложения использован язык программирования Pyhton. Были применены библиотеки для работы с Web-страницами, Telegram API и базами данных, логгирования и тестирования. Поговорим подробнее про каждую технологию.

Для взаимодействия с Telegram API использована библиотека TelebotAPI. Сначала для обмена данными с Telegram использовались http-запросы. Но потом стало понятно, что готовые решения для взаимодействия с API гораздо удобнее. Поэтому в качестве обертки был применен готовый инструмент - TelebotAPI, одно из самых популярных и простых в использовании решений. С помощью данной библиотеки осуществлено общение с пользователем.

При обработке web-сайтов использован пакет Requests$_HTML.requests.$, , Jav

Полученный html-код преобразовывался в словарь с помощью BeautifulSoup. Это позволило быстро и просто доставать всю необходимую информацию из данных, которые удалось взять с web-страниц университетов. Данный пакет является одним из самых надежных и популярных [3], поэтому был выбран именно он.

Для хранения данных был выбран PostgreSQL в совокупности с библиотекой psycopg2, наверное, самой популярной для работы с этой базой данных [4]. В качестве аналога была рассмотрен MySQL. Среди преимуществ такого инструмента можно выделить легкость в освоении, но эта база данных предназначена скорее для web-приложений. В случае нашего проекта, больше подходит PostgreSQL, в силу своей скорости [5].

Логгирование осуществлено с помощью пакета Logging. Вся важная информация об исключениях вносилась в отдельный файл, что обеспечило стабильность работы программы и возможность быстрого устранения возникающих проблем. Библиотека обладает широкой популярностью у разработчиков и большинство оценивают ее возможности достаточно высоко [6], поэтому выбор пал на Logging.

Unit-тестирование проведено с помощью стандартного пакета unittest. В качестве альтернатив рассматривались еще две популярные библиотеки:

nose и pytest. Но данные аналоги используются для крупномасштабных проектов [7] и требуют добавления большого числа зависимостей. Выбранный инструмент, в свою очередь, обладает достаточной функциональностью и не требует установки каких-либо модулей. В связи с этим был выбран именно этот unittest.

Для интеграционного тестирования использован инструмент GithubActions. Другим возможным аналогом был Jenkins. Но, во-первых, при использовании Github, намного удобнее пользоваться GithubActions. А, во-вторых, Jenkins предназначен для крупных коммерческих продуктов, каким не является данное приложение.

В организации совместной работы помог тасктрекер Trello для размещения и отслеживания задач. Выбор пал на этот сервис, так как он прост в освоении и там есть очень удобная функция: при возникновении изменений доски, на почту приходит уведомление об этом. Для параллельной разработки использован Github. Участники уже имели опыт работы с инструментом, и поэтому такое решение показалось наиболее оптимальным.

Для хостинга было выбран Heroku, главным образом, из-за возможности бесплатного использования данного сервиса.

Глава 4. Основная часть

4.1 Разработка архитектуры приложения

С самого начала разработки архитектуры стояла задача сделать приложение гибким и расширяемым, чтобы можно было легко добавлять новые возможности. В связи с этим было принято решение разделить программу на 5 модулей. В отдельные модули вынесены взаимодействие с пользователем, получение данных с сайтов, работа с базой данных, контроль взаимодействия всех остальных модулей, а также модуль, позволяющий связывать модели между собой. Таким образом, для обновления какой-то части продукта, достаточно провести изменения конкретного модуля. Это позволяет избежать конфликтов с остальными частями программы. Диаграмму модулей можно увидеть в приложении.

В проекте также используются несколько моделей: класс Student содержит всю информацию о пользователях. Интерфейс University и его реализации отвечают за взаимодействие с сайтами конкретных вузов и агрегирование необходимых для этого данных. Интерфейс Program нужен для того, чтобы хранить выбранные студентом образовательные программы и получать доступ к ним на сайте вуза. Общий интерфейс ConcreteUniversityStudent предназначен для удобства взаимодействия с информацией конкретного вуза и перехода от абстракции к чему-то более конкретному. Диаграмму классов можно увидеть в приложении.

4.2 Проектирование базы данных

В базе данных мы имеем одну таблицу с общей информацией об абитуриенте. Также здесь есть отдельные таблицы для каждого вуза, где мы храним информацию о выбранных образовательных программах. Такая архитектура базы позволяет легко добавлять новые университеты и пользователей, а также изменять или удалять старых.

4.3 Реализация модуля взаимодействия с Telegram API

Взаимодействие с пользователем через TelegramAPI вынесено в отдельный модуль. Здесь реализованы только методы обработки запроса пользователя и ответа сервера. Основная логика работы находится в посреднике, куда из данного модуля передаются все запросы пользователя.

При работе данного модуля используются длинные опросы сервера вместо коротких, чтобы меньше нагружать его. Интерфейс выбора вузов реализован через список, который отображается для пользователя. Но образовательные программы, направления и так далее, вводятся с клавиатуры из-за невозможности создать лист, где были бы перечислены все возможные варианты.

4.4 Реализация модуля-посредника, связывающего остальные модули

Для того, чтобы все компоненты приложения могли друг с другом взаимодействовать, был введен модуль-посредник, который позволяет решить эту задачу. Такой подход упростил архитектуру и уменьшил количество связей между модулями. Благодаря этому, каждый из них может быть изменен по-отдельности, без необходимости корректировки остальных.

Тут реализованы все основные методы: создание и изменение пользователя, получение информации о текущем положении при отправке запроса, сообщение о смещении, в случае, если абитуриент подписан на оповещения, а также добавление или удаление подписки на уведомления.

Особого внимания заслуживает метод notify, который оповещает абитуриента о смещении. Он находится в отдельном потоке, раз в час собирает информацию о положении подписанных на уведомления абитуриентов и сообщает им в случае, если их позиция изменилась в каком-то из списков.

4.5 Реализация модуля получения данных с Web-сайтов университетов

Для получения актуальной информации о текущей конкурсной ситуации был создан отдельный модуль. Здесь реализовано получение html-кода. Для этого с помощью библиотеки Requests$_HTMLget, , javascript - ., html - .$

В ходе разработки данного модуля также возникла еще одна проблема: невозможность доступа к сайтам некоторых вузов. Это связано с тем, что в момент разработки этого проекта, волна поступления не была активна, соответственно, некоторые сайты были сняты с хостинга. Решение данной проблемы было найдено в использовании инструмента WebArchive [8]. Он хранит историю некоторых часто посещаемых сайтов. Оттуда и была взята необходимая для парсинга информация.

4.6 Реализация обработки данных, полученных с сайтов вузов

Сайт каждого вуза имеет свою структуру. Такие различия влекут за собой необходимость наличия уникальных моделей для абитуриента. Так, для получения информации с сайта СПБГУ потребуется только ФИО, название образовательной программы, форма обучения и форма оплаты. А вот с университетом РАНХИГС все сложнее: тут нужно знать ФИО абитуриента, филиал, факультет, направление, образовательную программу и форму обучения. Именно по этой причине было принято решение ввести общий интерфейс для всех вузов, имеющий уникальную реализацию в каждом конкретном случае.

По окончании работы, все методы обработки информации с сайта возвращают для каждой образовательной программы номер абитуриента в соответствующем списке или бросают исключение, если что-то пошло не так.

4.7 Реализация модуля взаимодействия с базой данных

Для того, чтобы взаимодействовать с базой данных, создан отдельный модуль. Тут реализованы все необходимые методы для сохранения, изменения и удаления информации. Для того, чтобы не использовать чистый sql, применяется пакет, который позволяет легко и эффективно проводить все операции с информацией. Для ускорения работы, все методы обращения к базе являются асинхронными.

4.8 Логгирование

Для выявления непредвиденного поведения программы, все возможные исключения логгируются. Для этого создан отдельный файл, куда записывается информация о возникшей проблеме.

4.9 Тестирование

Для обеспечения стабильной работы программы и ее надежности, основные методы были покрыты тестами, что позволило на каждой итерации разработки приложения быть уверенным, что функциональность, созданная ранее, работает корректно.

4.10 Отправка приложения на хостинг

Приложение отправлялось на хостинг Heroku для того, чтобы протестировать его работоспособность, но за неактуальностью в текущий момент было снято оттуда. Планируется, что продукт снова будет загружен на сервер, когда начнется очередная волна поступления и проект станет актуален.

Глава 5. Заключение

Подводя итог, можно констатировать, что по результатам работы была успешно разработана архитектура, спроектирована база данных. Реализован модуль взаимодействия с Telegram API, модуль-посредник, связывающий остальные модули. Для получения данных с Web-сайтов университетов, также реализован отдельный модуль взаимодействия с базой данных. Решена задача обработки полученных данных. Для основной функциональности были написаны юнит-тесты и проведено интеграционное тестирование приложения. Все возникающие исключения были залоггированы.

Приложение планируется снова выложить на хост, когда это будет актуально.

Список литературы

[1] Анализ статистики численности сутдентов и поступивших в этом году. URL: https://aif.ru/society/education/chislennost$_s tudentov_{ik} olichestvo_m est_{vv} uzah_{vr} os$

[2] Сайт Admlist. URL: https://aif.ru/society/education/chislennost$_s tudentov_{ik} olichestv$

[3] Обзор и пример использования библиотеки Beautifulsoup. URL: https://habr.com/ru/sandbox/132503/ (дата обращения 25.09.2020).

[4] Обзор и пример использования библиотеки psycopg2. URL: https://khashtamov.com/ru/postgresql-python-psycopg2/ (дата обращения 28.10.2020).

[5] Сравнение реляционных баз данных. URL: https://mcs.mail.ru/blog/postgresql-ili-mysql-kakaya-iz-etih-relyacionnyh-subd/ (дата обращения 19.10.2020).

[6] Посвящение в логгирование с помомощью библиотеки Logging. URL: https://webdevblog.ru/logging-v-python/ (дата обращения 08.12.2020).

[7] Погружение в тестирование на python. URL: https://ru.hexlet.io/courses/advanced$_p ython/lessons/python_t esting_i ntroduction/th$

[8] Сайт Webarchive. URL: http://web.archive.org/ (дата обращения 04.10.2020).

Приложение

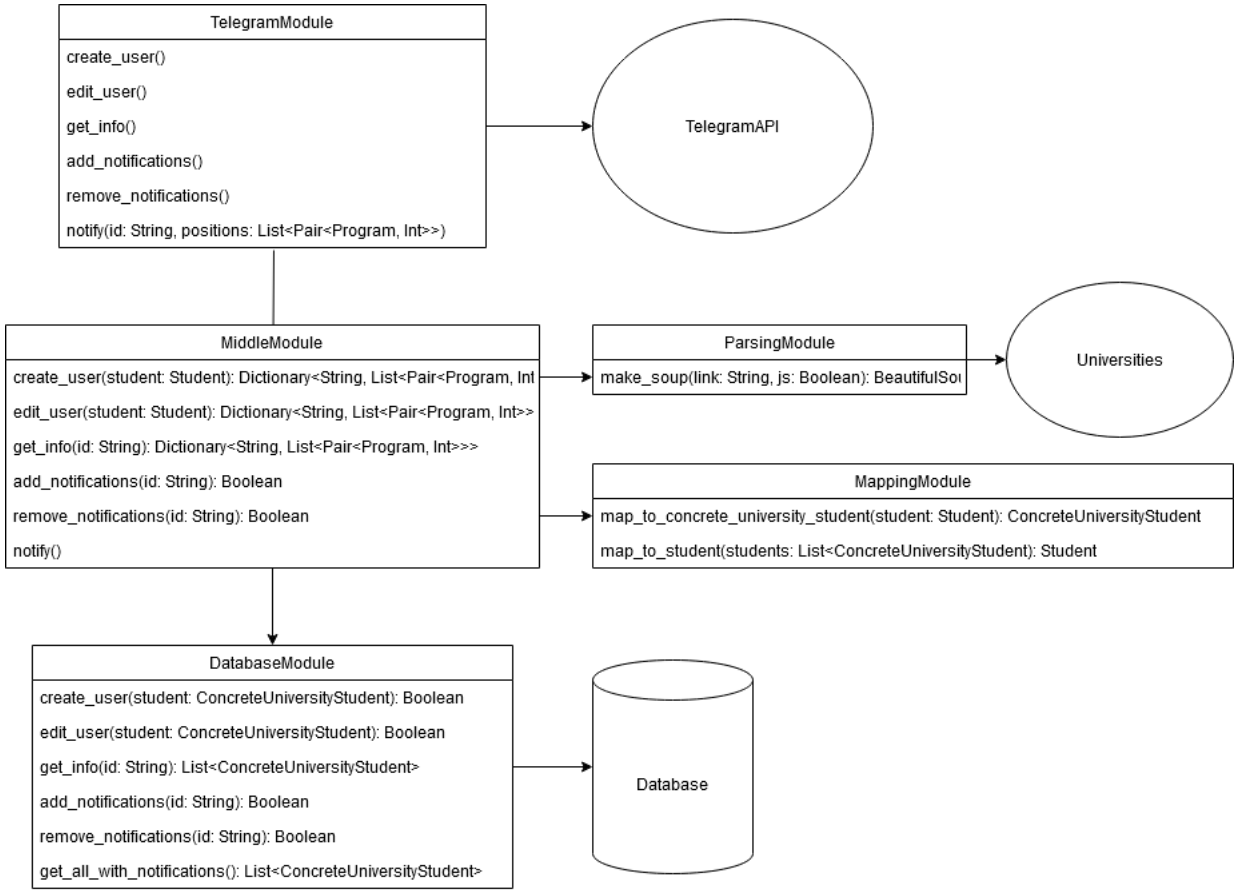

Рис. 1: Модульная диаграмма проекта.

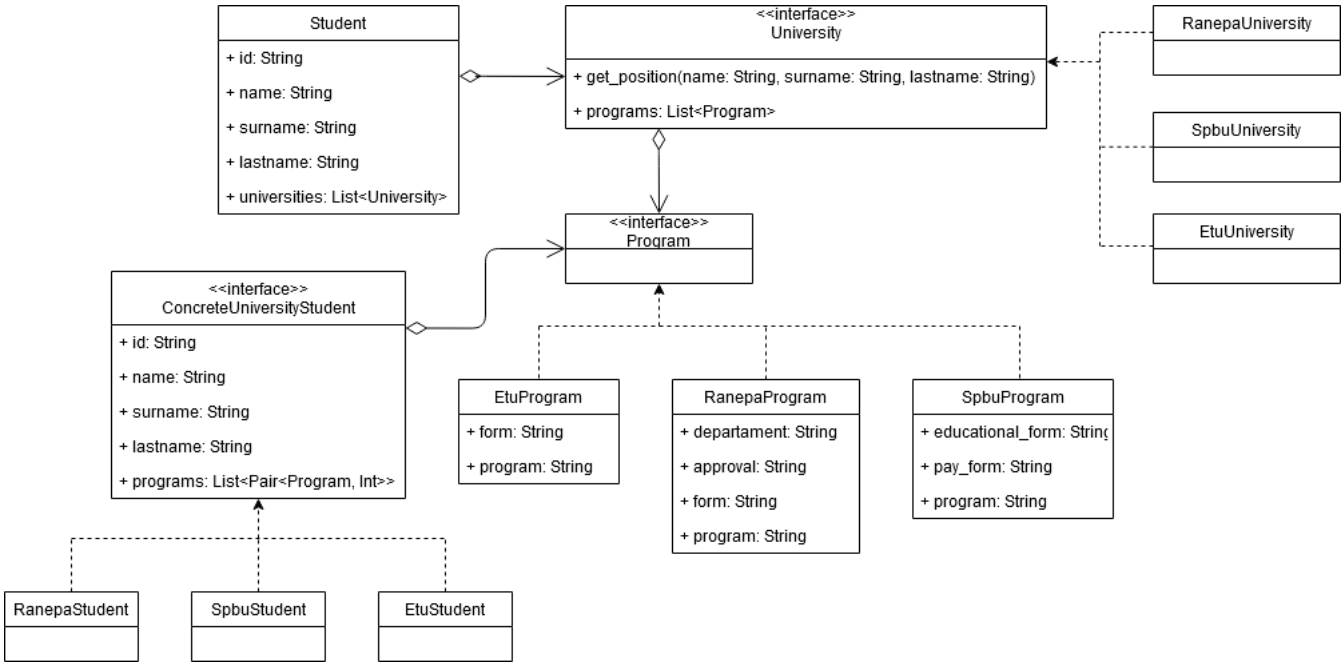

Рис. 2: Диаграмма классов проекта.