**Министерство образования и науки Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

# ОТЧЕТ ПО УЧЕБНОМУ ПРАКТИКУМУ

| | | |
|---|---|---|
| Студент | , 28.05.19 | **Ковалёв Д. С.** |
| | *подпись, дата* | *фамилия, и.о.* |
| Ментор команды | , 28.05.19 | **Бибин В. А.** |
| | *подпись, дата* | *фамилия, и.о.* |
| Руководитель практики | , 28.05.19 | **Оленев А. О.** |
| | *подпись, дата* | *фамилия, и.о.* |

*2019 г.*

# Оглавление

# Введение

Процесс регистрации при проведении летней практики нашей команде показался достаточно неоптимизированным и немасштабируемым. Было принято решение создать продукт, который исправит недостатки существующей системы, а также позволит дать студентам больше простора для организационных действий внутри команды и при коммуникации с ментором.

**Ментор команды:** Владимир Бибин

**Состав команды и распределение ролей:**

- Алексей Романов — руководитель проекта, Python-разработчик
- Дмитрий Ковалёв — Python-разработчик
- Павел Пересторонин — Python-разработчик
- Эмиль Симоненко — Python-разработчик, QA-инженер
- Сергей Кононенко — Python-разработчик, DevOps-инженер

# Аналитический раздел

## Обзор существующих решений

В данный момент для регистрации при проведении летней практики используются **Google Forms** (https://www.google.ru/intl/ru/forms/about/). Помимо этого ресурса так же есть **Survivo** (https://www.survio.com/ru/onlain-ankety), который обладает схожим функционалом с **Google Forms**. Остановимся на текущем решении, которое используется для регистрации на практику.

## Анализ достоинств и недостатков Google Forms

**Google Forms** имеет несколько ключевых недостатков:

- Невозможно изменить раннее созданную анкету; в случае, если пользователю нужно изменить данные, необходимо создавать новую анкету
- Невозможно просмотреть заполненную студентом анкету
- Студенту без команды невозможно подобрать команду без помощи организатора практики
- Отсутствие как таковой регистрации для ментора, что усложняет выбор команды
- Невозможно нормально посмотреть информацию о команде ментору
- Для того, чтобы получить общее представление о собранных данных, данные необходимо выгрузить в специальный сервис, который приведет их к читаемому виду (например, **Tableau Public** (https://public.tableau.com/s/))

С другой стороны, упомянутый **Google Forms** имеет и достоинства:

- Ресурс доступен из любой части Российской Федерации без установки дополнительного ПО
- Ресурс не требует регистрации в системе для заполнения анкеты

## Постановка задачи. Выводы

Количество недостатков перевешивает количество достоинств, притом достоинства не такие значительные по сравнению с недостатками, потому нашей командой было принято решение разработать продукт, который позволит упростить регистрацию и расширить её функционал.

В качестве платформы для реализации был выбран мессенджер **Telegram**, используемый кафедрой как средство общения студента с преподавателями. Так как у каждого студента к моменту проведения практики должен быть аккаунт в **Telegram**, развертка платформы на его базе позволит охватить всех студентов и не вызовет никаких трудностей в использовании.

# Конструкторский раздел

## Декомпозиция задачи

Перед реализацией задача была разбита на части, которые требовали последовательной реализации:

1. Создать опрашивающий механизм (процесс заполнения анкеты)
2. Создать базу данных (БД), куда опрашивающий механизм будет записывать информацию для каждого пользователя
3. Организовать возможность создания команд и отображения их списка для менторов
4. Организовать возможность набора практикантов без команд в наиболее подходящие по информации в БД неполные команды
5. Провести тестирование продукта
6. Организовать наиболее удобный способ работы с информацией в приложении

## Задачи, поставленные перед каждым участником

Следуя плану реализации, для начала необходимо было создать опрашивающий механизм. Данной задачей занялся Павел Пересторонин и Сергей Кононенко. После того, как каркас опрашивающего механизма был создан, была создана и настроена БД на сервере, в которой хранятся все данные, полученные опрашивающим механизмом. Созданием и настройкой БД занимались Дмитрий Ковалёв и Алексей Романов. После того, как были связаны БД и опрашивающий механизм, необходимо было добавить функцию создания команды и отображения списка всех команд для менторов. Над процессом написания функционала командного менеджмента работала вся команда. Набор практикантов, оставшихся без команды был реализован таким образом, что любой руководитель команды сможет вызвать список студентов, который в данный момент остались без команды и написать им о предложении вступить в команду. Этой функцией занимался Алексей Романов. Тестированием продукта занималась вся команда и фокус-группа, а руководил тестированием Эмиль Симоненко. Работа с информацией в приложении построена на основе общения студента с чат-ботом, который связан с БД и записывает в неё все данные, отправленные студентом.

## Разработка структуры создаваемого программного продукта

Как уже было сказано ранее, за платформу для реализации поставленной задачи был выбран мессенджер **Telegram**. **Telegram** позволяет создавать чат-ботов, которые можно наделять различным функционалом. В нашем случае чат-бот является связующей частью между пользователем и БД. Пользователь посылает сообщение боту, а бот в свою очередь обрабатывает и записывает его в БД.

## Описание назначения, требований к выделенным компонентам и их интерфейса

Интерфейс каждого компонента тривиален — это диалог с ботом в мессенджере.

**Опрашивающий механизм**

Используется для заполнения БД информации о пользователе. Вызывается при начале работы с ботом а также при необходимости изменить анкету.

**База данных пользователей**

Используется для хранения подробной информации о каждом пользователе. Обновляется каждый раз, когда вызывается опрашивающий механизм.

**Менторские возможности**

Используются ментором для работы в системе.

- Регистрация в системе и изменение анкеты
- Просмотр всех команд
- Подбор команды по интересам команды и ментора
- Подбор команды по направлению
- Просмотр анкеты любого студента
- Возможность закрепления команды за собой

**Пользовательские возможности**

Используются пользователем для работы в системе.

- Регистрация в системе и изменение анкеты
- Поиск команды (не тимлид)
- Просмотр списка своей команды
- Кик участника из команды (тимлид)
- Подбор участников в команду

## Разработка алгоритмов и структур данных для выделенных компонент

Для опрашивающего механизма было разработано два алгоритма: регистрация и изменение регистрации. Все данные содержатся в БД в структуре **User**, содержащей поля:

- Expected input — ожидаемый ввод пользователя
- Expected command — ожидаемая команда пользователя
- Is correct — логическое поле, отвечающее за корректность введенной анкеты

- Is mentor — логическое поле, отвечающее за роль пользователя (1 — ментор, 0 — студент)
- User id — идентификационный номер пользователя в БД
- Login — имя пользователя в Telegram
- Group — учебная группа студента
- Name — имя пользователя
- Surname — фамилия пользователя
- Father name — отчество пользователя
- Role command — роль пользователя в команде
- Team — название команды
- Team type — профиль команды
- User tech — технические средства, которыми владеет пользователь
- User exp — опыт пользователя в программировании
- Team idea — идея проекта
- Who mentor — логин ментора, прикрепленного к команде
- Python level — уровень знания Python
- C level — уровень знания C
- C sharp level — уровень знания C#
- Java level — уровень знания Java
- JS level — уровень знания JavaScript

Все компоненты продукта работают со структурой **User**, изменяя или получая из неё определенную информацию.

## Проектирование пользовательского интерфейса

В данном случае интерфейс един — диалоговое окно с ботом в **Telegram**.

## Описание способов тестирования как выделенных компонент, так и программного продукта в целом. Подготовка тестовых данных

Так как самой важной частью является пользовательская регистрация, огромное внимание было уделено корректности введенных данных: учитывались многие варианты написания фамилий (двойные, состоящие из одного символа), имён (по аналогии с

фамилией), отчеств (отсутствие отчества). Планируется провести ручное тестирование под управлением QA-инженера Эмиля Симоненко при помощи фокус-группы из 20 человек.

Помимо цельного тестирования планируются юнит-тесты для проверки корректности работы каждого из модулей.

После ручного и юнит-тестирования планируется провести функциональное тестирование готового продукта.

В качестве тестовых данных были выбраны некорректные личные данные пользователей.

## Выводы

Проработав конструкторскую часть и получив одобрение на реализацию данной бизнес-логики, команда перешла к выбору технологического стека.

# Технологический раздел

## Выбор и обоснование технических средств

В качестве основного языка разработки был выбран **Python**, так как для него существует библиотека **pyTelegramBotAPI**, существенно упрощающая и оптимизирующая разработку бота для Telegram.

В качестве основы для развертки был выбран фреймворк **Django**, а также идущая вместе с ним база данных **SQLite**. Была идея в качестве базы данных использовать **PostgreSQL**, но решением было остаться на **SQLite**, так как она даёт большую производительность на малом количестве одновременно использующих её пользователей, что как раз удовлетворяет нашему случаю.

Сервер был развёрнут на платформе **DigitalOcean** с использованием **ngrok** для обеспечения доступа по протоколу **HTTPS**. **DigitalOcean** позволяет развернуть готовый сервер без углубленной настройки. В свою очередь **ngrok** обеспечивает доступ по HTTPS без ручного изменения правил файервола системы.

Для создания документационной схемы был использован **Ramus Educational**. Программа бесплатна и проста в использовании для составления IDEF0 диаграмм.

Командная работа была организована на кафедральном **GitLab**.

Модульное тестирование проводилось при помощи встроенного Python-модуля **unittest**.

## Выбор и обоснование модели разработки

Была выбрана многофайловая модель разработки. У каждого участника команды был собственный Python-файл, в котором он разрабатывал функции. Затем всё это импортировалось в главный файл, который запускался при начале работы с ботом. Такая модель разработки позволила каждому участнику команды быть вовлеченным в процесс разработки. Таким образом мы добились максимально возможной производительности.

## Организация выпуска сборок, использование CI

На сервере был развернут рабочий репозиторий, то есть при каждом изменении в репозитории, изменения автоматически появлялись на сервере и для того, чтобы их применить, достаточно было запустить интеграционный скрипт. CI при разработке не использовался.

## Реализация программного продукта

Согласно с моделью разработки и декомпозиции задачи изначально была сделана настройка сервера для возможности развертывании бота на его основе. Далее была

разработана система регистрации и изменения уже заполненной анкеты. После того, как регистрация была закончена и протестирована, была начата параллельная разработка менторских и практиканстких возможностей. Как и в случае с регистрацией, после тестирования продукт был отправлен на цельное функциональное и ручное тестирование.

## Реализация тестирования

Для тестирования регистрации, менторских возможностей и пользовательских возможностей были разработаны юнит-тесты при помощи встроенного Python-модуля **unittest**. После модульного тестирование было проведено цельное ручное тестирование, а после него, с учётом всех погрешностей и исправления ошибок функциональное тестирование.

## Развертывание разработанного программного продукта, инструкция для системного администратора (установка) и пользователя (использование)

Данный продукт уже работает в автономном режиме и не требует развертки. Для того, чтобы начать с ним работу, достаточно начать общение с чат-ботом.

## Выводы

Придерживаясь первоначальному плану, мы не столкнулись с какими-либо проблемами при разработке. Грамотный выбор технологических средств позволил сделать процесс тестирования и разработки максимально эффективным и стабильным.

# Личный вклад

## Детальное описание поставленных задач

Изначально, передо были поставлены задачи:

1. Изучить фреймворк **Django**. Необходимо было разобраться в азах фреймворка, её структуре, настройке и связи с **Telegram**-ботом.
2. Изучить СУБД **SQLite 3**. Так как для хранения всех данных нам нужна была СУБД.
3. Изучить библиотеку **pyTelegramBotAPI** для **Python**.
4. Настройка **Webhook'a** для **Telegram**-бота.
5. Создать **Django** проект и связать его с телеграмм ботом, добиться чтобы бот просто отвечал на сообщения.
6. Настройка **Ngrok.**
7. Привязать **SQLite3** к **Django.**
8. Распределение функций в основном файле.
9. Реализация входа для ментора.
10. Осуществить взаимодействие между Тимлидом и другими участниками команды (сделать функцию добавления в команду, и выбор принять или нет для Тимлида).
11. Сделать меню после регистрации.

## Выбранные способы решения

1. При изучении фреймворка **Django**, я пользовался официальной документацией (https://docs.djangoproject.com/en/2.2/).
2. С СУБД **SQLite 3** тоже не возникло каких-либо вопрос, для ее изучения я так же пользовался документацией **Django**.
3. Ознакомление с библиотекой **pyTelegramBotAPI** прошло без трудностей. При ее изучении я использовал данный репозиторий на **GitHub**: https://github.com/eternnoir/pyTelegramBotAPI.
4. Создание **Ngrok** сервера, который обеспечивает доступ по **HTTPS** без ручного изменения правил фаервола системы (который мне предоставил Сергей Кононенко).
5. **JSON** формат.

## Возникшие трудности и способы их преодоления

1. Основные трудности возникли при попытке настроить **Webhook**. Мы решили получать сообщения с помощью веб хука, для этого нужен был какой-то веб адрес, чтобы телеграмм туда отправлял запросы, а **Django** запускался на **localhost**. В решении этой проблемы мне помог **ngrok**, который обеспечивает доступ по HTTPS

без ручного изменения правил фаервола системы. Так у меня появился URL на который можно было установить веб хук, запросы начали приходить на **Django**.

2. Запросы приходили, но не обрабатывались в коде. После двух дней проведенных в гугле я понял, что к доменному имени **ngrok** надо добавить путь до ресурса, и именно на такой url добавить веб хук. Тогда в **urls.py** оставалось только подключить код, в котором будут обрабатываться post запросы.
3. Запросы начали приходить в заданный файл, но все еще не работали как на **Telegram** декораторы. Тогда помогли **@csrf_exempt** и **json** парсинг, декораторы начали ловить post запросы и передавать в функцию объект **message**.

# Заключение

Благодаря успешному проектированию и реализации продукта, команда не столкнулась с какими либо глобальными трудностями, способными коренным образом сместить сроки разработки и тестирования. Грамотный подбор инструментов, выбор платформы для развертывания позволили дать на выходе продукт, в точности соответствующий плану разработки.

# Литература


- Writing your first Django app, part 1 — Django Documentation [Электронный ресурс]. URL: https://docs.djangoproject.com/en/2.2/intro/tutorial01/ (дата обращения: 17.04.19)
- PyTelegramBotAPI — GitHub [Электронный ресурс]. URL: https://github.com/eternnoir/pyTelegramBotAPI (дата обращения: 20.04.19)
- Функциональные тесты: Django + Selenium WebDriver и 3 варианта на Ваш выбор / Хабр [Электронный ресурс]. URL: https://habr.com/ru/post/268385/ (дата обращения: 15.05.19)