



Министерство науки и высшего образования 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

«Информационная безопасность» (ИУ8)

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОМУ ПРОЕКТУ*

### *НА ТЕМУ:*

*«Разработка веб-сервиса  
по планировке пользовательских задач»*

Студент

ИУ8-34  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

**А. Н. Александров**  
(И.О.Фамилия)

Руководитель курсового проекта

\_\_\_\_\_  
(Подпись, дата)

**А. А. Бородин**  
(И.О.Фамилия)

Консультант

\_\_\_\_\_  
(Подпись, дата)

**Д. С. Кокин**  
(И.О.Фамилия)

2020 г.

## Оглавление

Цель работы .....	3
Введение.....	4
Стек технологий .....	5
Описание работы.....	6
Описание API сервиса .....	8
Описание модели объекта User.....	13
Описание модели объекта FastTask.....	15
Описание модели объекта Schedule. ....	16
Описание взаимодействия с сервисом OpenWeatherMap. ....	18
Заключение .....	19
Список литературы .....	20
Приложение А .....	21

## **Цель работы**

Разработать сервис, реализующий создание пользовательских задач и списков дел. Создать подпрограмму для мессенджера Telegram, которая посредством HTTP запросов будет взаимодействовать с сервисом. С помощью этой подпрограммы, пользователь должен иметь возможность составлять регулярное расписание, создавать/редактировать списки текущих дел, получать необходимые напоминания об установленных планах и задачах, получать полезную информацию от сторонних сервисов для правильного планирования распорядка дня.

## Введение

В современном мире через нас проходит огромное количество информации, до 5 раз больше, чем 20 лет назад. Новостные сайты и блоги, электронные книги и их аудио аналоги, медиа и социальные сети — с появлением всех этих разнообразных источников информации мы, с одной стороны, получили доступ к огромной базе знаний, а с другой — порой сами не справляемся с большим потоком информации. Становится сложно держать в уме повседневные задачи и дела, о которых мы можем запросто забыть и попасть в неприятные ситуации.

Я поставил перед собой задачу создать сервис, позволяющий создавать дела с заданным временным интервалом, который используется для отправки пользователю напоминаний; выводить список добавленных дел; удалять их в случае их выполнения. Также была добавлена возможность создания регулярного расписания, которое можно заполнять и выводить на нужный нам день с данными о времени, месте, и другим. Сервис предоставляет API для этих целей. Его использует созданный телеграм-бот, с которым взаимодействует пользователь. Дабы пользователь был готов к любым погодным условиям, существует возможность получения погодных данных на текущий момент времени, а также прогноз на ближайшие дни.

## Стек технологий

1. Язык программирования **Go (Golang)**;
2. СУБД **PostgreSQL**;
3. **Travis-CI**;
4. ПО **Docker**;

Выбор Go не случаен. Это компилируемый язык программирования, имеющий высокую скорость при умеренном потреблении ресурсов. Go разрабатывался с учётом актуальных тенденций в области веб-разработки, он многопоточен и эффективно использует ресурсы вычислительного устройства. Язык прост в изучении, имеет исчерпывающую документацию, он лаконичен и читабелен; имеет большую популярность, позволяющую быстро найти ответы на возникающие вопросы.

Для хранения пользовательских данных была использована реляционная СУБД PostgreSQL, способная обрабатывать большие объёмы данных, при этом сохраняя их целостность и структурированность. Она очень гибкая и адаптивная для дальнейшего расширения. Хорошая поддержка в виде документации и статей, а также широкое сообщество помогает в реализации собственных задач.

Платформа непрерывной интеграции Travis-CI помогла с автоматизированной сборкой проекта. Она легко интегрируется с GitHub, где хранится проект, бесплатна для использования и хорошо подходит для большинства необходимых кейсов. Для автоматизации развёртывания было использовано программное обеспечение Docker, которое является устоявшимся стандартом контейнеризации, в котором уже доступна работа со многими популярными приложениями, включая необходимую технологию Postgres.

## Описание работы

На базе популярного кроссплатформенного мессенджера **Telegram** был создан бот, с которым и будет взаимодействовать пользователь посредством команд и кнопок. Чтобы сформировать запрос к сервису со стороны бота, порой приходится выстраивать цепочки сообщений, чтобы пользователь добавлял всё новую и новую информацию. При этом отключение бота или какой-то сбой на сервере не должен помешать составлению запросов: ввод не должен начинаться с самого начала, а другие пользователи никак не должны затрагивать наш ввод.

Пакет *telegram-bot-api* для работы с ботом «из коробки» не предоставлял возможность выстраивать цепочки сообщений с пользователями, из-за чего пришлось хранить состояния пользователей в виде уникальных кодов на каждое состояние, а переходы между ними осуществлять в виде абстрактной модели конечного автомата (рисунок 1). Все состояния сохраняются в базе данных для каждого пользователя, таким образом падение сервера или перезапуск бота не приводит к потере состояния для пользователя.



## Описание API сервиса

Таблица 1 Описание API сервиса

Название	URL	Функция-хендлер	Метод	Пример тела запроса	Пример тела ответа
Создание нового пользователя.	/user/	AddUserHandler	POST	{ "user_id": 478749325, "username": "lesha", "state_code": 0, "state_request": "{}" }	—
Получение информации обо всех пользователях.	/user/	GetUsersHandler	GET	—	[ { "user_id": 5728279325, "username": "petr82", "state_code": 0, "state_request": "{}" }, { "user_id": 238749315, "username": "henry73", "state_code": 0, "state_request": "{}" } ]
Получение информации о пользователе по его id.	/user/{id}	GetUserHandler	GET	—	{ "user_id": 778749347, "username": "kirill", "state_code": 0, "state_request": "{}" }
Обновление состояния пользователя по его id.	/user/{id}	UpdateUserStateHandler	PUT	{ "user_id": 478749325, "username": "lesha", "state_code": 1, "state_request": "{}" }	—



				}	
Добавление задачи.	/id/fast_task/ /	AddFastTaskHandler	POST	{ "id": 3, "assignee_id": 478749325, "chat_id": 478749325, "task_name": "Купить хлеба", "interval": 60000, "deadline": "2017-11-20 10:20:10 +0000 UTC" }	—
Получение задач всех пользователей (для проверки дедлайнов напоминий).	/fast_task/	GetAllFastTasksHandler	GET	—	[ { "id": 3, "assignee_id": 478749325, "chat_id": 478749325, "task_name": "Купить хлеба", "interval": 60000, "deadline": "2017-11-20 10:20:10 +0003 UTC" }, { "id": 6, "assignee_id": 778749347, "chat_id": 778749347, "task_name": "Сделать лабы", "interval": 6000000, "deadline": "2020-12-21 13:42:10 +0003 UTC" } ]
Получение задач для пользователя по его id.	/id/fast_task/ /	GetFastTasksHandler	GET	—	[ { "id": 3, "assignee_id": 478749325, }

					<pre> "chat_id": 478749325,  "task_name": "Купить хлеба",  "interval": 60000,  "deadline": "2017- 11-20 10:20:10 +0003 UTC"  },  {  "id": 4,  "assignee_id": 478749325,  "chat_id": 478749325,  "task_name": "Купить молока",  "interval": 50000,  "deadline": "2020- 12-01 07:42:10 +0003 UTC"  }  ] </pre>
Обновление дедлайнов для напоминаний для всех пользователей.	/fast_task/	UpdateFastTasksHandler	PUT	<pre> [  {  "id": 3,  "assignee_id": 478749325,  "chat_id": 478749325,  "task_name": "Купить хлеба",  "interval": 60000,  "deadline": "2017-11- 20 10:20:10 +0003 UTC"  },  {  "id": 6,  "assignee_id": 778749347,  "chat_id": 778749347,  "task_name": "Сделать лабы",  "interval": 6000000, </pre>	—

				<pre>"deadline": "2020-12-21 13:42:10 +0003 UTC" } ]</pre>	
Удаление задачи.	<code>/ {id} / fast_task / {ft_id}</code>	DeleteFastTaskHandler	DELETE	—	—
Добавление дела в расписание.	<code>/ {id} / schedule /</code>	AddScheduleTaskHandler	POST	<pre>{   "id": 1,   "assignee_id": 478749325,   "week_day": "Monday",   "title": "Лек. Комбинаторика",   "place": "501ю",   "speaker": "А. Е. Жуков",   "begin": "2020-12-05 13:50:00",   "end": "2020-12-05 15:25:00" }</pre>	—
Получение расписания для пользователя на заданный день недели.	<code>/ {id} / schedule / {week_day} /</code>	GetScheduleTaskHandler	GET	—	<pre>[   {     "id": 1,     "assignee_id": 478749325,     "week_day": "Monday",     "title": "Лек. Комбинаторика",     "place": "501ю",     "speaker": "А. Е. Жуков",     "begin": "2020-12-05 13:50:00",     "end": "2020-12-05 15:25:00"   },   {     "id": 2,     "assignee_id": 478749325,     "week_day": "Monday",</pre>

					"title": "Вечерняя пробежка", "place": "Стадион Гидросталь", "speaker": "—", "begin": "2020-12-05 19:00:00", "end": "2020-12-05 19:15:00" } ]
Обновление дела в расписании.	/ {id} /schedule /	UpdateScheduleTaskHandler	PUT	{ "id": 1, "assignee_id": 478749325, "week_day": "Monday", "title": "Лаб. Физика", "place": "Дом Физики", "speaker": "И. Н. Алиев", "begin": "2020-12-05 13:50:00", "end": "2020-12-05 15:25:00" }	—
Удаление дела в расписании.	/ {id} /schedule / {sch_id} /	DeleteScheduleTaskHandler	DELETE	—	—
Удаление расписания на день недели.	/ {id} /schedule /delete/ {week_day} /	DeleteScheduleWeekHandler	DELETE	—	—
Полная очистка расписания для пользователя.	/ {id} /schedule /	ClearAllHandler	DELETE	—	—

## Описание модели объекта User.

Используется для создания пользователей и хранения информации о них, включая состояния.

Структура **User** имеет следующие поля:

- Id (тип *int*) — уникальный идентификатор пользователя;
- UserName (тип *string*) — имя пользователя;
- StateCode (тип *int*) — код состояния пользователя для построения цепочек сообщений в Telegram боте;
- StateRequest (тип *string*) — в себе строку с JSON представлением структуры, которая заполняется в процессе составления пользователем запроса.

Соответствующая таблица **tg\_user** в базе данных представлена следующим образом:

Таблица 2 Поля SQL таблицы *tg\_user*

Название	Тип/ограничение/связь
user_id	integer unique
username	varchar(255)
state_code	integer
state_request	text

Как упоминалось ранее, состояния пользователей нужны для составления цепочек сообщений между ботом и пользователем. Также стоит отметить поле StateRequest, в котором находится строка с JSON представлением структуры, которая «собирается» в процессе заполнения пользователем новых данных. Таким образом мы не теряем никаких данных даже в случае перезапуска бота/удаления чата с ним. Благодаря конечному

автомату и хранению состояний, удаётся сохранять целостность данных: пользователи ведут диалог с ботом, не мешая вводу чужих данных.

## Описание модели объекта **FastTask**.

Используется для создания «быстрых» задач. Пользователь имеет возможность задать задачу с интервалом времени напоминания.

Структура **FastTask** имеет следующие поля:

- Id (тип *int*) — идентификатор задачи;
- AssigneeId (тип *int*) — идентификатор владельца этой задачи;
- ChatId (тип *int64*) — идентификатор Telegram чата, куда нужно прислать напоминание пользователю;
- TaskName (тип *string*) — название задачи;
- NotifyInterval (тип *time.Duration*) — интервал времени, с которым пользователь будет получать напоминания о задаче;
- Deadline (тип *time.Time*) — время ближайшего дедлайна.

Соответствующая таблица **fast\_task** в базе данных представлена следующим образом:

Таблица 3 Поля SQL таблицы *fast\_task*

Название	Тип/ограничение/связь
id	serial primary key
assignee_id	integer references tg_user(user_id)
chat_id	bigint
task_name	varchar(255)
notify_interval	bigint
deadline	timestampz

В отдельном потоке происходит проверка, прошёл ли срок напоминания о задаче, была реализована отправка сообщений пользователю. Также предусмотрен вывод списка задач, их удаление за ненадобностью; обновление дедлайна производится на величину интервала в момент отправки очередного.

## Описание модели объекта **Schedule**.

Используется для создания регулярных расписаний. Когда пользователь имеет какое-то постоянный список дел, который не изменяется в продолжительном промежутке времени (занятия в ВУЗе, спортивные мероприятия, курсы, ежедневные встречи с командой, написание статусов о проделанной работе и др.), он может просто заполнить его один раз, в дальнейшем имея возможность вызывать его на нужный день.

Структура **ScheduleTask** имеет следующие поля:

- Id (тип *int*) — идентификатор дела;
- AssigneeId (тип *int*) — идентификатор владельца этого дела;
- WeekDay (тип *time.Weekday*) — день недели;
- Title (тип *string*) — название дела;
- Place (тип *string*) — место проведения этого дела;
- Speaker (тип *string*) — имя выступающего (лектора, преподавателя);
- Start (тип *time.Time*) — время начала дела;
- End (тип *time.Time*) — время окончания дела.

Соответствующая таблица **schedule** в базе данных представлена следующим образом:



Таблица 4 Поля SQL таблицы *schedule*

Название	Тип/ограничение/связь
id	serial primary key
assignee_id	integer references tg_user(user_id)
week_day	varchar(10)
title	varchar(255)
place	varchar(50)
speaker	varchar(50)
start_time	time
end_time	time

Итак, пользователь имеет возможность заполнять расписание, добавляя новые задачи по дням недели. Добавляя задачи, пользователь может заполнить время начала события только раньше его окончания, что логично. Если произошла ошибка в заполнении какого-либо дела, его можно обновить. Пользователю предоставляется вывод расписания на сегодня, на завтра (так как это одни из самых распространённых кейсов), на произвольный день недели по выбору, а также полный вывод на все заполненные дни. При выводе, все дела сортируются по полю времени, так что нам не обязательно добавлять их последовательно. Неактуальные события и дела можно удалять из расписания, а также очищать расписание полностью или все дела на какой-либо день недели.

## **Описание взаимодействия с сервисом OpenWeatherMap.**

*OpenWeatherMap* — онлайн сервис, предоставляющий API для доступа к данным о текущей погоде и прогнозам для веб-сервисов и мобильных приложений. В качестве источника данных используются официальные метеорологические службы, данные из метеостанций аэропортов, и данные с частных метеостанций.

Я воспользовался бесплатным тарифом, который предоставляет мне данные о погоде на текущий момент времени, а также прогноз на ближайшие 5 дней. Таким образом пользователь получает возможность ввести название места где он находится, либо отправить данные геопозиции и получить соответствующий вывод. Данные приходят в формате JSON, парсятся и отправляются пользователю. При этом вывод приводится к виду минимально необходимому и достаточному. К примеру, код описания погоды использовался для вывода миниатюрного изображения с соответствующими погодными явлениями, а показатель угла направления ветра — для представления в виде указателя на 8-ветровой розе компаса.

## **Заключение**

В результате проделанной работы был разработан рабочий сервис, который предоставляет API для решения поставленных выше задач, а также подпрограмма для мессенджера Telegram для удобного взаимодействия пользователя с сервисом.

Были получены навыки разработки клиент-серверных приложений на языке Go; опыт работы с базой данных PostgreSQL, а также использование API сторонних сервисов для извлечения полезного функционала.

В дальнейшем планируется добавление возможности прокладывания маршрутов с выводом затраченного времени на дорогу; просмотр расписаний общественного транспорта, ближайших рейсов; задачи с пользовательскими статусами их выполнения.

## Список литературы

1. Разработка веб-сервисов на языке Go - основы языка / Романов В. В. [Электронный ресурс]. -URL: <https://www.coursera.org/learn/golang-webservices-1> (дата обращения 14.09.20). - Режим доступа: Электронный портал онлайн-образования *Coursera.org*
2. Спецификация языка программирования Go: от 14 января 2020 г [Электронный ресурс]. - URL: <https://golang.org/ref/spec>
3. Документация к PostgreSQL 13.1 / The PostgreSQL Global Development Group. - Текст: электронный. -URL: <https://www.postgresql.org/files/documentation/pdf/13/postgresql-13-A4.pdf>

## **Приложение А**

Ссылка на репозиторий с проектом

[https://github.com/bmstu-iu8-g4-2020-project/todo\\_web\\_service](https://github.com/bmstu-iu8-g4-2020-project/todo_web_service)