



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

Development of a Chatbot for  
Tourist Recommendations  
Technical Documentation



Presentado por Jasmin Wellnitz  
en Universidad de Burgos — June 8, 2017  
Tutor: Bruno Baruque Zanón

---

# Contents

---

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>iii</b>
<b>List of Tables</b>	<b>iv</b>
<b>Appendix A Plan de Proyecto Software</b>	<b>1</b>
A.1 Introducción . . . . .	1
A.2 Planificación temporal . . . . .	1
A.3 Estudio de viabilidad . . . . .	1
<b>Appendix B Especificación de Requisitos</b>	<b>2</b>
B.1 Introducción . . . . .	2
B.2 Objetivos generales . . . . .	2
B.3 Catalogo de requisitos . . . . .	2
B.4 Especificación de requisitos . . . . .	2
<b>Appendix C Especificación de diseño</b>	<b>3</b>
C.1 Introducción . . . . .	3
C.2 Diseño de datos . . . . .	3
C.3 Diseño procedimental . . . . .	3
C.4 Diseño arquitectónico . . . . .	3
<b>Appendix D Technical Programming Documentation</b>	<b>4</b>
D.1 Introduction . . . . .	4
D.2 Directory Structure . . . . .	4
D.3 Developer Manual . . . . .	5
D.4 Program Compilation, Installation and Execution . . . . .	13
D.5 Tests . . . . .	13
<b>Appendix E Documentación de usuario</b>	<b>14</b>

E.1	Introducción . . . . .	14
E.2	Requisitos de usuarios . . . . .	14
E.3	Instalación . . . . .	14
E.4	Manual del usuario . . . . .	14

---

## List of Figures

---

D.1	pgadmin3: A server connection is added . . . . .	6
D.2	The <i>Botfather</i> is used to create the Telegram Bot . . . . .	8
D.3	api.ai agent creation interface . . . . .	9
D.4	Heroku Web App Creation . . . . .	11

---

## List of Tables

---

## *Appendix A*

---

# **Plan de Proyecto Software**

---

### **A.1 Introducción**

### **A.2 Planificación temporal**

### **A.3 Estudio de viabilidad**

Viabilidad económica

Viabilidad legal

## *Appendix B*

---

# **Especificación de Requisitos**

---

- B.1 Introducción
- B.2 Objetivos generales
- B.3 Catalogo de requisitos
- B.4 Especificación de requisitos

## *Appendix C*

---

# **Especificación de diseño**

---

- C.1 Introducción
- C.2 Diseño de datos
- C.3 Diseño procedimental
- C.4 Diseño arquitectónico



## *Appendix D*

---

# Technical Programming Documentation

---

### D.1 Introduction

This section contains the technical programming documentation, describing the directory structure of the presented CD, an installation guide as well as a documentation of the realized tests to measure the code quality.

### D.2 Directory Structure

This report is handed in along with a CD containing the essential data to examine the application in more depth. The presented CD contains the following directories:

**Documentation** Contains the project's documentation, saved as both .pdf and .doc format.

**Software** Contains the executables of the tools needed to run the virtual machine.

**Application** Contains all essential data of the developed software, subdivided in the following directories:

**Virtual Machine** The virtual machine image containing the infrastructure that was set up in the course of the project

**Source Code** The source code of the application

**Javadoc** The source code's documentation in Javadoc format.

**Agent** Contains the exported api.ai agent as .zip

**Data** Contains the raw geographic data and dumps of the initial database

## D.3 Developer Manual

This section serves as an installation guide describing which steps to take to set up the application.

### Database Setup

The geographic database is used by the chatbot application to retrieve and manage geographical data. In this project, the database runs on a Virtual Machine using Ubuntu 16.04 LTS. Therefore, Ubuntu’s terminal is used to install most of the software. In order to make sure Ubuntu has access to the current package index, it is advised to execute an update command before installing the software:

```
sudo apt-get update
```

### Set up PostgreSQL and PostGis

The first step is to install the data management system, PostgreSQL. To install the version used in this project, the following command is used:

```
sudo apt-get install -y postgresql=9.5+173
    postgresql-contrib=9.5+173
```

Then, the database “touristdb” is created as well as the managing user, which is called “touristuser”. The createuser command will prompt for a password which can be chosen by the developers.

```
sudo -u postgres createuser -P touristuser
sudo -u postgres createdb -owner touristuser touristDB
```

Now we have set up the database, the PostGIS extension is installed and added to prepare the database for geospatial data.

```
sudo apt-get install -y postgis
    postgresql-9.5-postgis-2.2
sudo -u postgres psql -c "CREATE_EXTENSION_postgis;_
    CREATE_EXTENSION_postgis_topology;" touristDB
```

The next step is optional, but seems convenient if the developers want to manage their database with the help of a user interface. The managing tool

pgadmin facilitates running and editing SQL queries and viewing the stored data.

```
sudo apt-get install pgadmin3
```

To access the database in pgadmin3, a connection to the server must be added, which can be realized by clicking the plug button in the upper toolbar and then entering the following values.

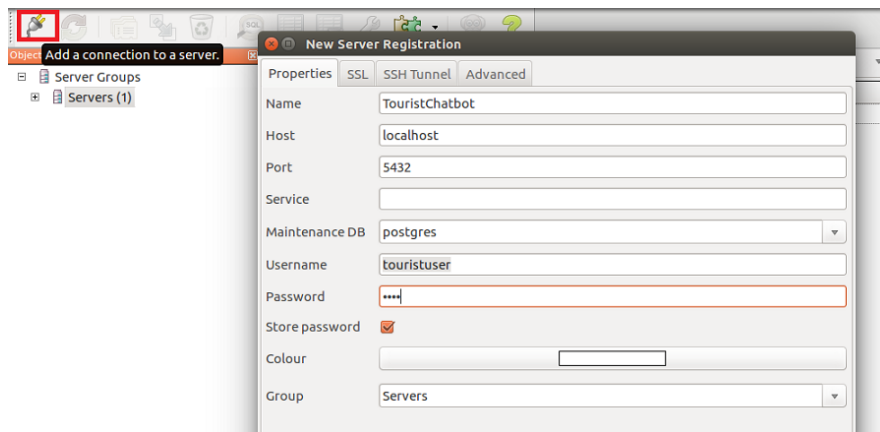


Figure D.1: pgadmin3: A server connection is added

In the object browser, the database schemas can be viewed accessing TouristChatbot -> databases -> touristDB.

### Import Data into Database

Now that we have set up the database, it needs to be filled with geospatial data. In this project, the recommendations are based on test data of Barcelona. The required data is downloaded as a .pbf file from the website <https://download.bbbike.org/osm/bbbike/Barcelona/>. After that, the tool Osmosis is used to import the OSM data which can be installed using the following command:

```
sudo apt-get install osmosis
```

The next commands prepares the database for the osmosis import. It sets the hstore extension and the pgsnapshot database schema which causes that all relevant tag data are stored in a hstore column.

```
sudo -u postgres psql -c "CREATE_EXTENSION_hstore;"
touristDB
psql -U touristuser -d touristDB -f
/usr/share/doc/osmosis/examples/pgsnapshot_schema_0.6.sql
```

After that, the import itself is realized. Remember to execute this command in the folder where the downloaded .pbf file is situated and to add the corresponding password (which is set by the developer in the previous step of this manual).

```
osmosis --read-pbf file="Barcelona.osm.pbf"
--write-pgsql host="localhost" database="touristDB"
user="touristuser" password=password
```

In order to see if the import was successful, pgadmin3 can be used to take a look at the now imported data. Again, this step is optional. In the object browser on the left hand side of the user interface, the database tables can be viewed accessing *TouristChatbot* → *databases* → *touristDB* → *Schemas* → *public* → *Tables*.

### Store User Information

In order to store information of the users or of the ratings they made, the existing users table must be modified. To do so, the following POSTGRESQL queries are executed so that new columns are added to our table. These modifications can either be made using the psql command via bash or pgadmin3's query tool.

```
alter table users add column recommendations bigint [];
alter table users add column unrated bigint [];
alter table users add column radius integer;
alter table users add column name ;
```

The ratings are stored in a newly created table:

```
create table ratings(
  userId bigint ,
  pointId bigint ,
  ratings integer ,
  PRIMARY KEY(userId , pointId))
```

## Access to External Services

In the following, it is described how to set up and access the external services used in this project: To access the conversational interface, the messenger Telegram as well as our natural language parsing platform `api.ai` are used. Additionally, the FourSquare API is used to retrieve images for recommended Points of Interests.

## Telegram Bot

The messenger Telegram is used to provide an interface to our tourist bot. After installing Telegram on a mobile device and setting up an account, the bot can be created using Telegram's *BotFather*. The Botfather can be accessed using the messenger's search function. After that, the creation of the bot is triggered by entering `/newbot` in the input field.

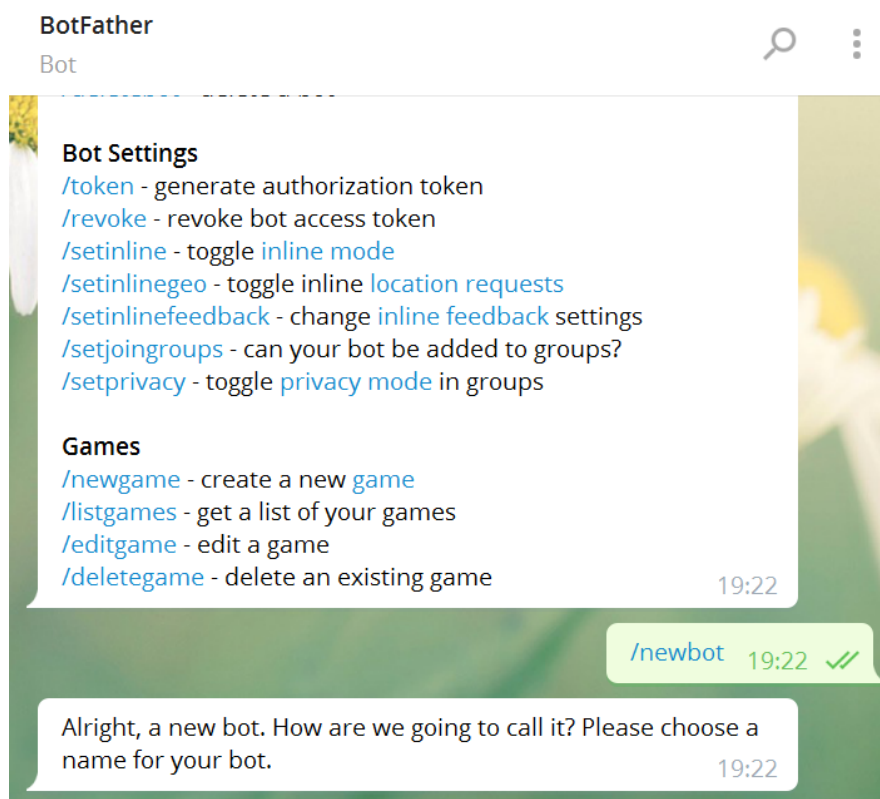


Figure D.2: The *Botfather* is used to create the Telegram Bot

After choosing a name and a username, the BotFather provides you with the authorization token for your bot. This authorization token is needed to access the Telegram bot from our web service. To do so, the token is saved

as an environment variable (see [System Environment Variables](#)). After saving the token, the web service is able to receive updates from and send messages to the Telegram bot via a webhook.

### api.ai agent

In order to use the NLU platform api.ai, we need to set up an [account](#). After doing so, the api.ai agent modeling interface can be accessed. First, we need to create a new agent and enter an agent name.

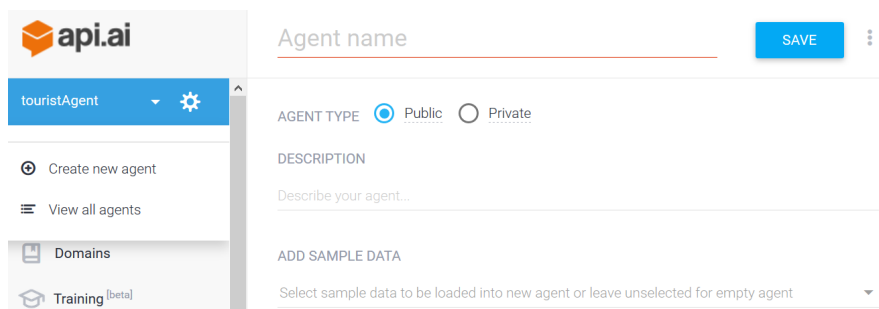


Figure D.3: api.ai agent creation interface

If the creation is successful, the entered agent name will appear on the left sidebar. In order to access the agent from our web service, api.ai's HTTP API is used. Therefore, the agent's API key is needed which can be accessed by clicking on the gear icon right to the agent name. The client access token is be copied and, again, introduced into the system environment variables (see [System Environment Variables](#)).

To restore the agent created in the course of this project, a.zip file containing the modeled agent can be found in the project's documents (Application/Agent). By clicking on the already mentioned gear icon right to the agent name, a subtab called "Export and Import" provides the possibility to import the agent from zip.

### Foursquare

To retrieve images using the Foursquare API, a Foursquare account has to be created first. After that, the application has to be [registered](#). If the registration is successful, the API access token can be found in the application overview. Again, these tokens are saved in the [System Environment Variables](#)).

## Web Service Setup

### Prerequisites

This project runs on Java 8 which is a requirement for the web service framework Java Spark as well as the used cloud service Heroku. The JDK can be downloaded from [Oracle](#). Git is used to manage the project's source code as well as to deploy the code to the Platform as a Service application. In order to manage a Git repository, you have to sign up on [GitHub](#) and install Git using the following command.

```
sudo apt-get install git-all
```

On top of it, all libraries used during this project are included using the build-management tool Apache Maven. This program is installed by executing the following command in Ubuntu's command line. Maven is also needed for deploying a Java application to Heroku.

```
sudo apt-get install maven
```

### Deployment to Heroku

The web service is deployed using Heroku, a cloud Platform as a Service (PaaS). In order to use the application, an account has to be created previously, following <https://signup.heroku.com>. At first, the Heroku command line interface has to be installed. Using Ubuntu, this is achieved executing the following commands:

```
sudo add-apt-repository "deb_
    https://cli-assets.heroku.com/branches/stable/apt_./"
curl -L https://cli-assets.heroku.com/apt/release.key |
    sudo apt-key add -
sudo apt-get update
sudo apt-get install heroku
```

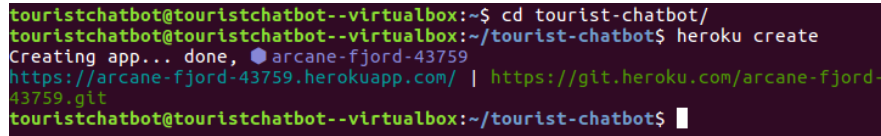
After installing the command line, execute the following command and enter the Heroku credentials when asked.

```
heroku login
```

In Ubuntu's file system, change into the project directory `touristbot` and execute the following command in order to create a Heroku app.

```
heroku create
```

As we can see, a random application name is assigned, in this case e.g. <https://arcane-fjord-43759.herokuapp.com/>. This name has to be copied and inserted as our HEROKU \_ URL to the [System Environment Variables](#)). In doing so, the Telegram bot is hooked to the Heroku app later.



```
touristchatbot@touristchatbot--virtualbox:~$ cd tourist-chatbot/
touristchatbot@touristchatbot--virtualbox:~/tourist-chatbot$ heroku create
Creating app... done, ● arcane-fjord-43759
https://arcane-fjord-43759.herokuapp.com/ | https://git.heroku.com/arcane-fjord-43759.git
touristchatbot@touristchatbot--virtualbox:~/tourist-chatbot$
```

Figure D.4: Heroku Web App Creation

The source code can be now deployed using the command:

```
git push heroku master
```

### Heroku Postgres Setup

In order to access our geospatial database online, Heroku Postgres is used to set up a productive PostgreSQL database. The following commands are executed from the Heroku repository:

```
heroku addons:create heroku-postgresql:hobby-dev
```

After that, the psql command is used in Heroku to enable sending POST-GRESQL queries:

```
PGUSER=postgres heroku pg:psql
```

The following queries are executed to enable the PostGis support in the PostgreSQL database.

```
CREATE EXTENSION postgis;
CREATE EXTENSION hstore;
CREATE EXTENSION postgis_topology;
```

The touristdb that was set up in the [Database Setup](#) is then pushed to the just created database:



```
PGUSER=postgres heroku pg:push touristdb DATABASE_URL
```

## System Environment Variables

In order to manage the access tokens of our external components and not push them publically into a repository, system environment variables are used. These are set differently according to whether tests are run locally on the virtual machine or the application runs productively in Heroku. To set the system environment variables locally, the file `/etc/environment` is modified to contain the following variables:

```
HEROKU_URL=INSERT HEROKU URL
DATABASE_URL="postgres://touristuser:password@localhost:5432/touristdb"
TELEGRAM_TOKEN=INSERT TELEGRAM TOKEN
API_AI_ACCESS_TOKEN=INSERT API.AI CLIENT ACCESS TOKEN
F_CLIENT_ID=INSERT FOURSQUARE CLIENT ID
F_CLIENT_SECRET=INSERT FOURSQUARE CLIENT SECRET
```

The placeholders are replaced with the respective tokens from the external services. Concerning the variable `DATABASE_URL`, the `POSTGRES` password has to be entered that was set in the [Database Setup](#)). For the productive runtime environment, the `bash` is used to set the environment variables on Heroku, entering the following command:

```
heroku config:set TOKEN_PARAMETER=VALUE
```

This command's execution is repeated for all of the above mentioned tokens with the exception of the variable `DATABASE_URL` as it is an already predefined environment variable.

## Integrated Development Environment

The project is developed using Eclipse Neon as an IDE. The 64-bit installer can be downloaded from [Eclipse](#)'s website. After installing Maven and Eclipse, start Eclipse in order to import the source code. This can be easily done by importing a Maven project, executing *File* → *Import* → *Existing Maven Projects* and then choosing the project's source folder.

The project's source code can now be accessed and modified using Eclipse. Additionally, Eclipse is used to run the Junit tests for this project.

## D.4 Program Compilation, Installation and Execution

The presented application was designed for online usage and is deployed to the Platform as a Service Heroku. Therefore, no further compilation, installation or execution steps are needed as this is managed by the PaaS. Changes to the previous source code are published by using the Git workflow, meaning to commit the changes and then push them to Heroku using the command:

```
git push heroku master
```

## D.5 Tests

## *Appendix E*

---

# **Documentación de usuario**

---

- E.1    Introducción**
- E.2    Requisitos de usuarios**
- E.3    Instalación**
- E.4    Manual del usuario**