# **Annex - 4**

# 4. Developer Manual

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

## 4.1 Geographic Database

### 4.1.1 Set up PostgreSQL and PostGis

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 16.0.4 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

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 make the database able to handle geospatial data.

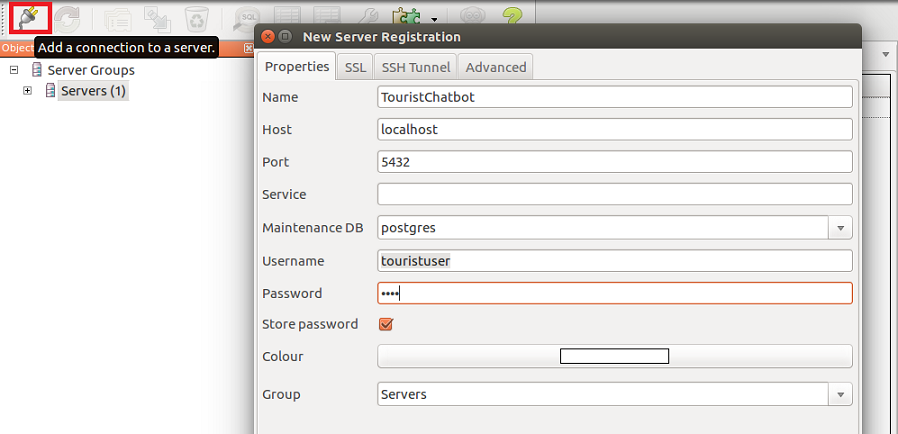
sudo apt-get install -y postgis postgresql-9.5-postgis-2.2

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 facilates 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 toobar and then entering the following values.



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

### 4.1.2 Import Data into Database

Now that we have set up the database, it needs to be filled with geospatial test data. In this project, test data of the Barcelona is used which can be downloaded as a .pbf file from the website https://download.bbbike.org/osm/bbbike/Barcelona/. Then, the tool Osmosis is used to import the OSM data which can be installed by 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.

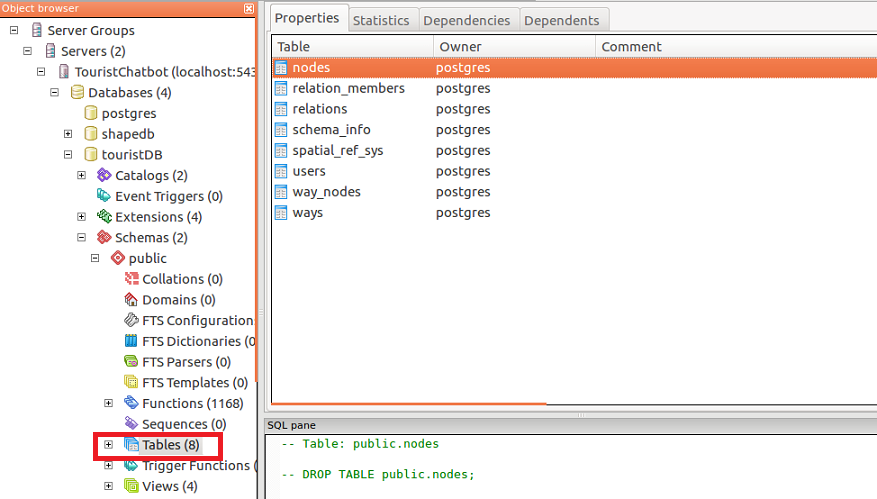
sudo -u postgres psql -c "CREATE EXTENSION hstore;" touristDB

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, the database tables can be viewed accessing TouristChatbot -> databases -> touristDB-> Schemas -> public -> Tables.



If you open the context menu on one of the tables (e.g. nodes) by clicking right, the option *view data* is available which shows the previously imported data.

## 4.2 Web Service Setup

### 4.2.1 Requirements

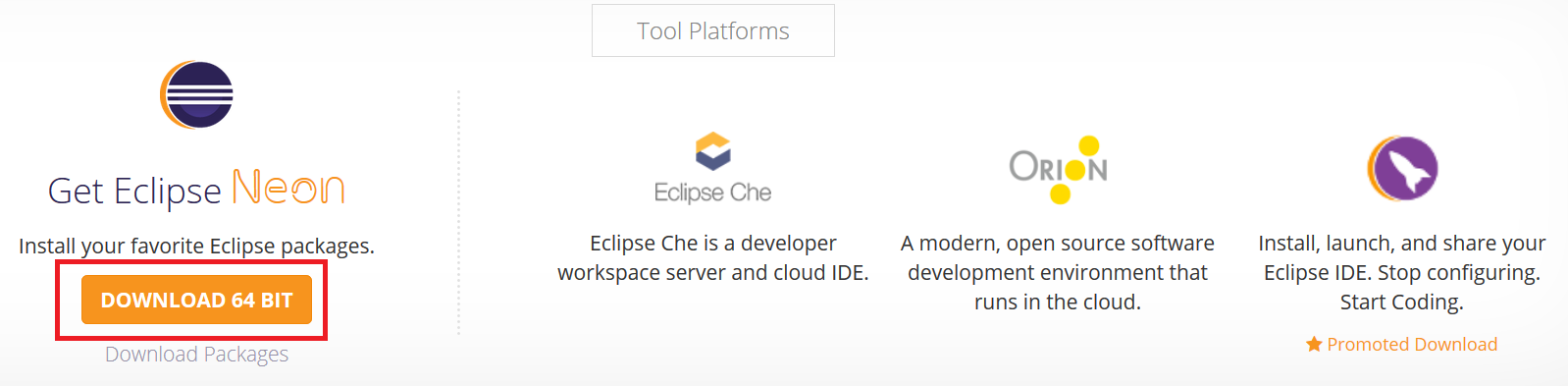
This project runs on *Java 8* which is a requirement for the web service framework *Java Spark* as well as the used cloud service*.* The *JDK* can be downloaded from [**Oracle**](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html).

*Git* is used to manage the project’s source code as well as deploying the code to a *Platform as a Service* application. In order to use *Git*, you have to sign up on [GitHub](https://github.com/join?source=header-home) and install *Git* using the following command.

sudo apt-get install git-all

### 4.2.2 Basic IDE

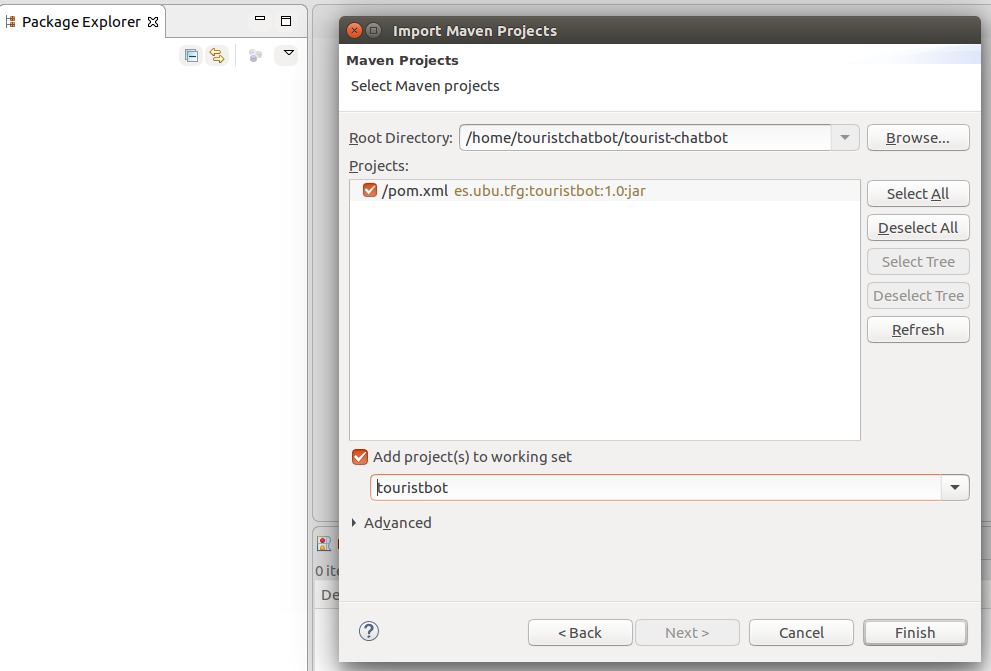
The project is developed using *Eclipse Neon* as an IDE. The 64-bit installer can be downloaded from the [**Eclipse**](https://www.eclipse.org/downloads) website.



All libraries used during this project are included using the build-management tool *Apache Maven*. This program can be installed by executing the following command in Ubuntu’s command line:

sudo apt-get install maven

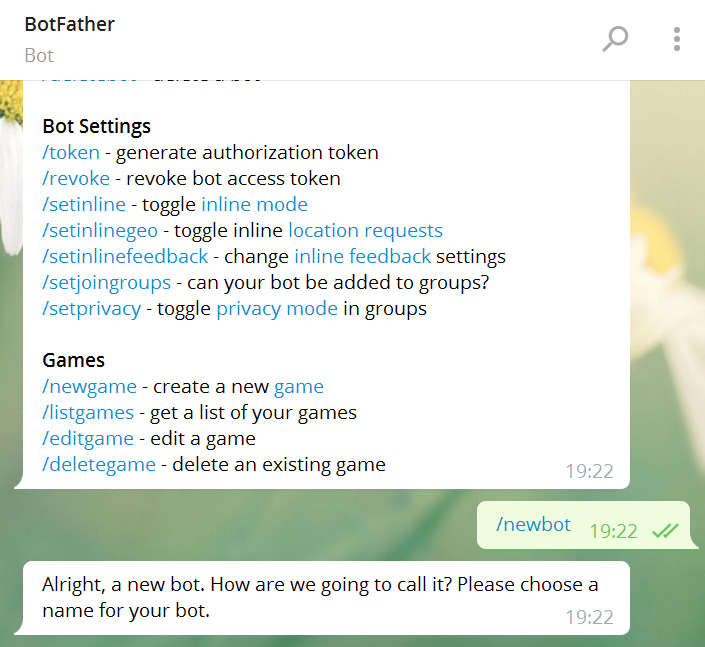
After installing *Maven* and *Eclipse*, start *Eclipse* in order to import the source code. This can be easily done by importing a Maven project, executing **File->Import->Existing Maven Projects** and then choosing the project folder source.



The project’s source code can now be accessed and modified using Eclipse.

### 4.2.3 Telegram Bot

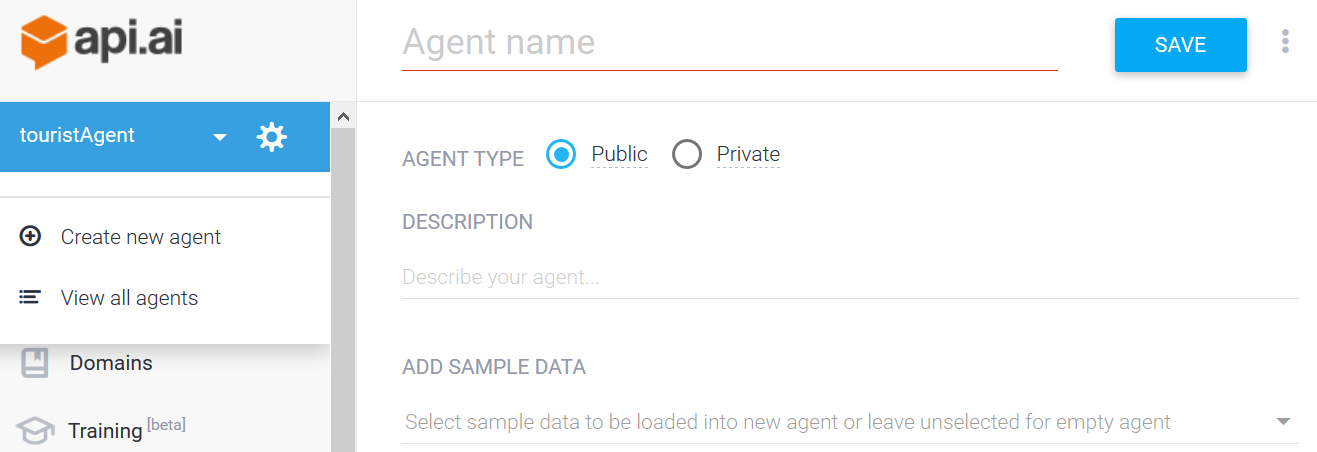
The messenger *Telegram* is used to provide an interface to our tourist bot. After installing *Telegram* on a mobile device and setting up an account, the bot can be created using *Telegram’s* *BotFather*. Using the messenger’s search function, the *BotFather* can be accessed and the creation of the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



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. Therefore, we need to introduce the token into the *config.properties* file as a value for the parameter *telegramToken*. After introducing this token, our web service is able to receive updates from and send messages to the Telegram bot via a webhook.

### 4.2.4 api.ai agent

In order to use the NLU platform *api.ai*, we need to set up an [account](https://console.api.ai/api-client/)*.* Creating a free account is sufficient for this application. 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.

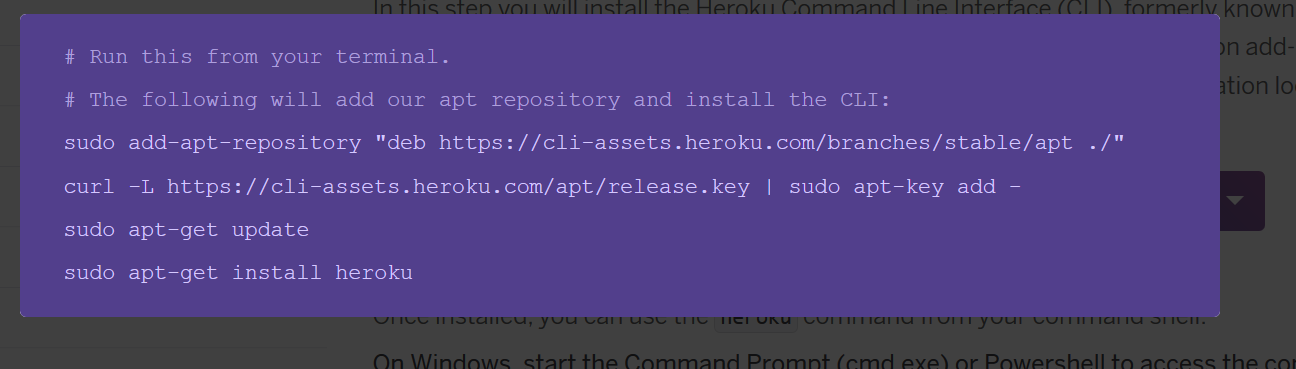


If the creation is successful, the entered agent name will appear on the left sidebar. In order to access our agent from our web service, *api.ai’s* HTTP API is used. Therefore, the agent’s API keys are needed which can be accessed by clicking on the gear icon right to the agent name. Both the client and the developer access token have to be copied and introduced into the *config.properties* file to the corresponding parameter (*clientAccessToken* and *developerAccessToken*).

### 4.2.5 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>. During this project a free account was used.

At first, the *Heroku* command line interface has to be installed. Using Ubuntu, this is achieved executing the following commands:



How to install Heroku’s command line interface, extracted from Heroku’s Getting Started guide [\*]

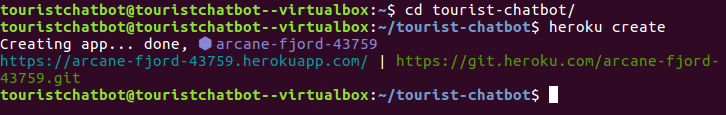
After installing the command line, execute the following command and enter the *Heroku* credentials when asked.

heroku login

Change into our 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 <https://arcane-fjord-43759.herokuapp.com/>.



To connect our web service with this application, we need to change the file *config.properties* and add the name to the parameter *serviceUrl*.

The source code can be now deployed using the command:

git push heroku master

To make sure that the application is running, the following command can be used:

heroku ps:scale web=1

# Bibliography

<http://www.paulshapley.com/2016/04/how-to-install-postgresql-95-and.html>

<https://devcenter.heroku.com/articles/getting-started-with-java#set-up>