**Programmers Guide**

|  |  |
| --- | --- |
| UMGC Fall 2020 | Benjamin Fetterman, Benjamin Murray, Hanim Danur, James Cornelius, Robert Lee  SWEN 670 |

Project Plan Approvals

|  |  |  |
| --- | --- | --- |
| Name | Signature | Date |
| Approved by:  Dr. Mir Assadullah |  |  |
| Approved by:  "Stakeholder" |  |  |
| Approved by:  "Project Manager" | Robert Lee | 10/20/2020 |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Name | Description of Change(s) |
| 1.0 | 10/20/2020 | Robert Lee | Initial Release |
| 1.1 | 11/03/2020 | James Cornelius | Update with Final Code Changes |
| 1.2 | 11/03/2020 | Robert Lee | Final Review |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1.0 Introduction 6](#_Toc55322405)

[1.1 Purpose 6](#_Toc55322406)

[1.2 Intended Audience 6](#_Toc55322407)

[1.3 Technical Project Stakeholders 6](#_Toc55322408)

[2.0 System Architecture 6](#_Toc55322409)

[2.1 Architectural Design 7](#_Toc55322410)

[2.2 Architectural Overview 7](#_Toc55322411)

[3.0 Application Setup 7](#_Toc55322412)

[3.1 Git and Code Deployment 7](#_Toc55322413)

[3.1.1 Downloading GitHub Desktop 8](#_Toc55322414)

[3.1.2 Downloading the Latest Code 8](#_Toc55322415)

[3.1.3 Downloading Eclipse 8](#_Toc55322416)

[3.1.4 Downloading Spring Tools 4 for Eclipse 8](#_Toc55322417)

[3.1.5 Creating MapQuest Developer Account 8](#_Toc55322418)

[3.2 IBM Watson® setup 8](#_Toc55322419)

[3.2.1 Dashboard 9](#_Toc55322420)

[3.2.2 Resource List 10](#_Toc55322421)

[3.2.3 Watson Assistant 1.0 Overview 10](#_Toc55322422)

[3.2.4 Assistants 11](#_Toc55322423)

[3.2.5 Skills 11](#_Toc55322424)

[3.2.6 City Chatbot Configuration 12](#_Toc55322425)

[3.3 Code Structure 13](#_Toc55322426)

[3.4 Source code 13](#_Toc55322427)

[3.5 Class and Interface Descriptions 14](#_Toc55322428)

[3.5.1 Maps 14](#_Toc55322429)

[3.5.2 Polygons 14](#_Toc55322430)

[3.5.3 MapsRepository 14](#_Toc55322431)

[3.5.4 PolygonsRepository 14](#_Toc55322432)

[3.5.5 ParsingService 14](#_Toc55322433)

[3.5.6 JsonParsingService 14](#_Toc55322434)

[3.5.7 MainController 14](#_Toc55322435)

[3.5.8 ZonePolygon 15](#_Toc55322436)

[3.5.9 MapHandler 15](#_Toc55322437)

[3.5.10 DBConnection 15](#_Toc55322438)

[3.5.11 ProcessRequest 15](#_Toc55322439)

[3.5.12 WatsonArguments 15](#_Toc55322440)

[3.5.13 MunicipalPermitChatbotApplication 16](#_Toc55322441)

[4.0 Code Scaffolding 16](#_Toc55322442)

[5.0 Running the Application 16](#_Toc55322443)

[5.1 Server (Database) 16](#_Toc55322444)

[5.1.1 Prerequisites 18](#_Toc55322445)

[5.1.2 Installation Steps 18](#_Toc55322446)

[5.1.3 Configuration 19](#_Toc55322447)

[5.1.4 Configured Values 21](#_Toc55322448)

[5.1.5 Administration 21](#_Toc55322449)

[5.2 Server (REST API) 21](#_Toc55322450)

[5.2.1 Prerequisites 21](#_Toc55322451)

[5.2.2 Installation Steps 23](#_Toc55322452)

[6.0 Location of Repository 29](#_Toc55322453)

[7.0 Challenges/Issues/Concern 29](#_Toc55322454)

[8.0 Additional Help 30](#_Toc55322455)

[9.0 License Information 30](#_Toc55322456)

[Appendix A: Acronyms and Abbreviations 31](#_Toc55322457)

[Appendix B: References 32](#_Toc55322458)

List of Tables

[Table 1 Stakeholders 6](#_Toc55322459)

[Table 2 MySQL Supported Operating Systems (MySQL, N.d) 18](#_Toc55322460)

List of Figures

[Figure 1 IBM Cloud Login Page 9](#_Toc55322461)

[Figure 2 IBC Cloud Dashboard 9](#_Toc55322462)

[Figure 3 User's Resources in IBM Cloud 10](#_Toc55322463)

[Figure 4 Watson Assistant Overview Page 10](#_Toc55322464)

[Figure 5 List of User's Watson Assistants 11](#_Toc55322465)

[Figure 6 Example Skill of Watson Assistant 11](#_Toc55322466)

[Figure 7 Customize an Assistant's Skill 12](#_Toc55322467)

[Figure 8 Example Watson Assistant Intent 12](#_Toc55322468)

[Figure 9 Example of Watson Assistant Entity 12](#_Toc55322469)

[Figure 10 Example Watson Assistant Dialog 13](#_Toc55322470)

[Figure 11 Database GitHub Repository 19](#_Toc55322471)

[Figure 12 SQL File to Initialize Database 19](#_Toc55322472)

[Figure 13 The Tables in the Database 20](#_Toc55322473)

[Figure 14 Example Database Configuration 21](#_Toc55322474)

[Figure 15 Import from Git 23](#_Toc55322475)

[Figure 16 Repository Source 24](#_Toc55322476)

[Figure 17 GitHub Repository Information 24](#_Toc55322477)

[Figure 18 Branch Selection 24](#_Toc55322478)

[Figure 19 Host Machine Location 25](#_Toc55322479)

[Figure 20 Import Wizard Selection 25](#_Toc55322480)

[Figure 21 Finish Importing Project 26](#_Toc55322481)

[Figure 22 Eclipse Run Configuration Menu 27](#_Toc55322482)

[Figure 23 Specify Run Configuration for Eclipse 28](#_Toc55322483)

[Figure 24 Application Environment Variables 29](#_Toc55322484)

1. Introduction

Purpose

The purpose of this programmer’s guide is to provide current and future programmers with an understanding of the City of Pasadena’s City Chatbot application and its development environment; here onto referred to as the system, application, or agent. This guide will discuss the software architecture of the application and how it is incorporated in the architecture of its deployment.

Intended Audience

The intended audience of this programmer’s guide is specifically the software developers, programmers, or testers of the chatbot application. This guide will cover the deployment of back end architecture, the configuration of the Watson chatbot, and the testing abilities that have been established.

**Technic**al Project Stakeholders

This section provides a list of all current stakeholders with an interest in this project at the time of release.

Table 1 Stakeholders

|  |  |  |
| --- | --- | --- |
| Name | E-mail address | Role |
| Professor Assadullah | mir.assadullah@faculty.umgc.edu | Stakeholder |
| Robert Lee | rlee97@student.umgc.edu | Project Manager |
| Benjamin Fetterman | bfetterman2@student.umgc.edu | Developer |
| Benjamin Murray | bmurray19@student.umgc.edu | Developer |
| Hanim Danur | hdanur@student.umgc.edu | Developer |
| James Cornelius | jcornelius10@student.umgc.edu | Developer |
| Glenn Goodlett | ggoodlett1@student.umgc.edu | DevOps |
| Dustin Emerson | demerson2@student.umgc.edu | DevOps |

1. System Architecture

This section addresses the design that was selected for this project. This project being done on behalf of the UMGC SWEN 670 Capstone course dictated some of the technology sections made. In order for this project to be passed from semester to semester with relative ease, the Spring.IO and MySQL selections were made. In an effort to select the most cost-effective way to keep this project going, IBM Watson and MapQuest API were selected because their free editions allow an adequate number of API calls per month to facilitate the development of the project. If the end user would like to have more robust features, they may elect to replace those individual components. The final part of this project that was driven by the UMGC requirements was the election to not integrate directly with a single map platform. In exchange classes were developed to import KML files for the zone data from a city and import it into the database. Java is then used to manipulate that data and provide the results instead of making API calls to specific geospatial platforms.

Architectural Design

The system utilizes the IBM Watson Chat Application for the client front end section of the design, the Spring IO framework for the back end, and MySQL for the database. This guide will describe how to do a local deployment when developing on any of the components. The production deployment process is described in the deployment guide. The MapQuest Geocoding API is used for verifying a user’s address. Webhook services handle all calls from Watson to the Spring IO back end. Exception handling is implemented to avoid crashes, errors, and other system failures.

Architectural Overview

The user facing component of the application consist of the IBM Watson Assistant application. Watson was selected based on its free implementation of multiple dialog flow nodes, API request limits, and overall general availability vs its competitors in Azure and Google. The Watson application was tailored to observe key input variables, or entities, from the user that allow the chat bot to determine an intent from the user. The bot allows for multiple conversation flow patterns with the primary one being address verification happening first. The Chatbot takes in the user address and uses a webhook to communicate with a Java Spring.IO application hosted in Azure. Both Azure and Spring.IO were selected at the request of the sponsor to facilitate the easy passage of the application to other teams. The application takes the street address, appends the city and state of Pasadena, CA before sending it to MapQuest to return the longitude and latitude of the address to the application. MapQuest, like IBM Watson, was selected for its free processing of up to 100,000 request per month. The Java application then processes the longitude and latitude of the address to identify the zone that the address resides in. If a good zone is found, the zone is returned to Watson to continue the conversation with the user. If no zone is found, Watson is triggered to reengage the user for another address. In the standard dialog flow, once the chatbot has identified a correct zone, the user is prompted to ask their question. Watson takes key components of those questions, leads them to more identify a more specific request and sends that request back to the java application. As with the zone identification, the application queries a MySQL database in order to select the necessary information to respond to the users request.

1. Application Setup

Git and Code Deployment

GitHub is a code hosting platform, or repository, that provides a collaborative environment for project documentation, version control, code repository, etc. GitHub is a common and widely used repository for software development projects. A few benefits from utilizing GitHub are increase in efficiency, team collaboration, and organization.

* + 1. Downloading GitHub Desktop

The following link, **<https://desktop.github.com/>**, is for downloading the GitHub Desktop application. This application will allow one to easily access code repositories on GitHub. Click on the Purple link/button to download the GitHub desktop application for the appropriate OS. Once one has downloaded the setup file follow the instructions presented. One will need an active GitHub account in order to sign into the application.

* + 1. Downloading the Latest Code

The current code for the project can be found in the GitHub repositories at the following links, <https://github.com/umgc/umgc.chatbot> and https://github.com/umgc/umgc.chatbot.db. Before one can access the code ensure that a GitHub account is created. Reach out to the DevSecOps Team to ensure that the GitHub account has access to the repository. Once signed in, one can retrieve the current code by clicking the green download button with the text Code. Select the open with GitHub Desktop option, then in the GitHub Desktop click the Fetch Origin button. One will now have the latest code on their system.

* + 1. Downloading Eclipse

The Eclipse IDE, in which the Code was developed can be found at the following link, [**https://www.eclipse.org/downloads/**](https://www.eclipse.org/downloads/). Download the latest version of Eclipse, once the installer has downloaded follow the instructions presented.

* + 1. Downloading Spring Tools 4 for Eclipse

Open Eclipse, go to Help à Eclipse Marketplace, in the search bar type in Spring Tools 4. Click on the install link. One’s Eclipse now has Spring Tools 4 installed and can support Spring projects.

* + 1. Creating MapQuest Developer Account

The MapQuest Geocode API is used in the address verification process for the application. An account needs to be created in order to use this API. Sign up for MapQuest Developer can be found at [**https://developer.mapquest.com/plan\_purchase/steps/business\_edition/business\_edition\_free/register**](https://developer.mapquest.com/plan_purchase/steps/business_edition/business_edition_free/register). Once an account has been created, go to the Manage Keys section of your account profile. Click the button to create a new key. Fill in a name for the project and click the Create App button to generate the API key. The API key is listed as the Consumer Key under the project you just created.

IBM Watson® setup

The agent is accessible through IBM Cloud®. Setup and maintenance is minimized because IBM Cloud® maintains the agent’s infrastructure and the agent’s model. The agent is accessed through IBM Cloud, and configured within IBM Cloud.

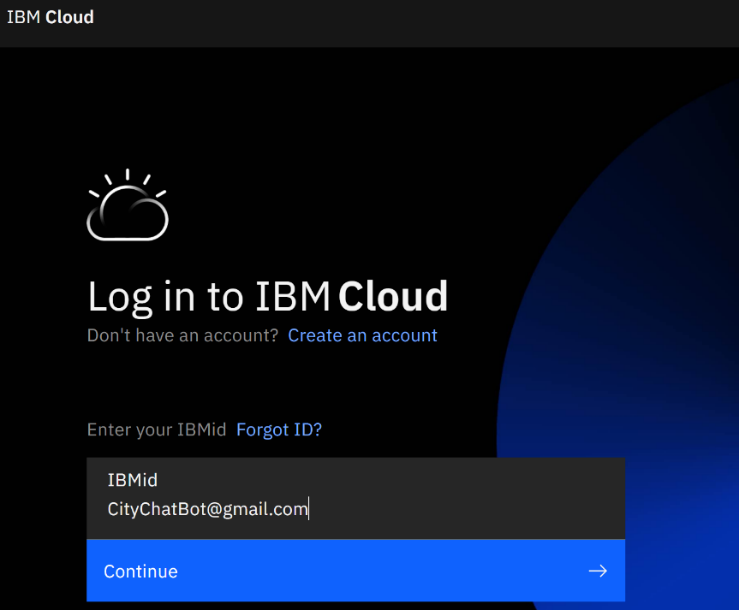


Figure 1 IBM Cloud Login Page

Below is the URL and credentials required to access the agent.

IBM Watson® URL: <https://cloud.ibm.com>

UserName: [CityChatbot@gmail.com](mailto:CityChatbot@gmail.com)

Password: Passwordz1

* + 1. Dashboard

IBM Cloud contains a Dashboard that summarizes the resources within the cloud instance. The agent is a cloud service/resource and is accessible by accessing “Services” within the Dashboard.

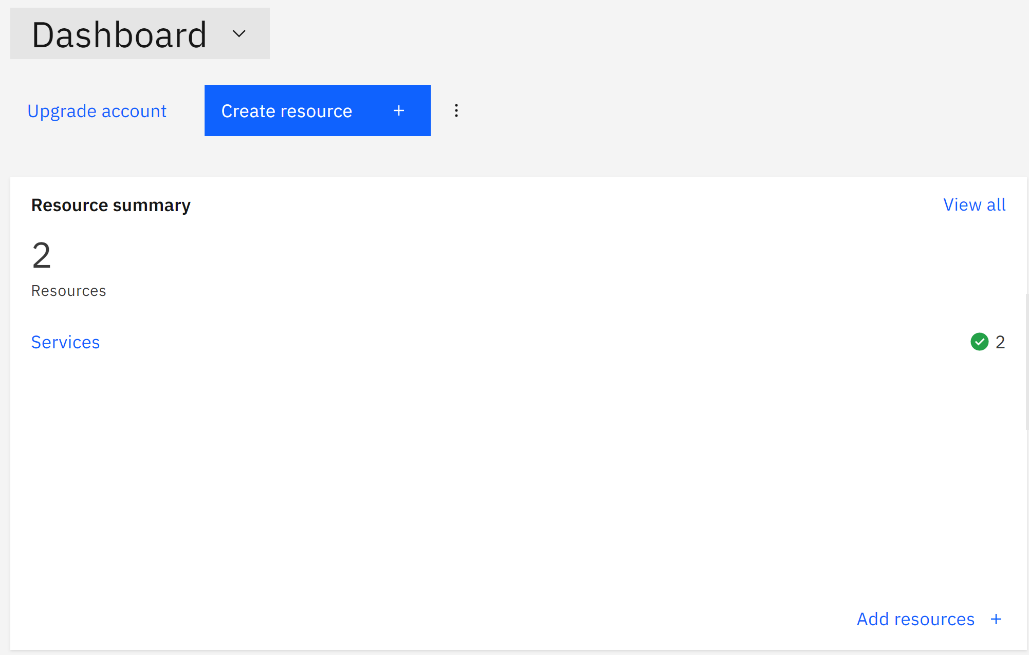


Figure 2 IBC Cloud Dashboard

* + 1. Resource List

The cloud resource list, shows all resources associated and available with the cloud instance. It details the server location and status of the resources. From the resource list new resources can be created, or existing resources can be accessed. Below is the resource list for the CityChatbot:

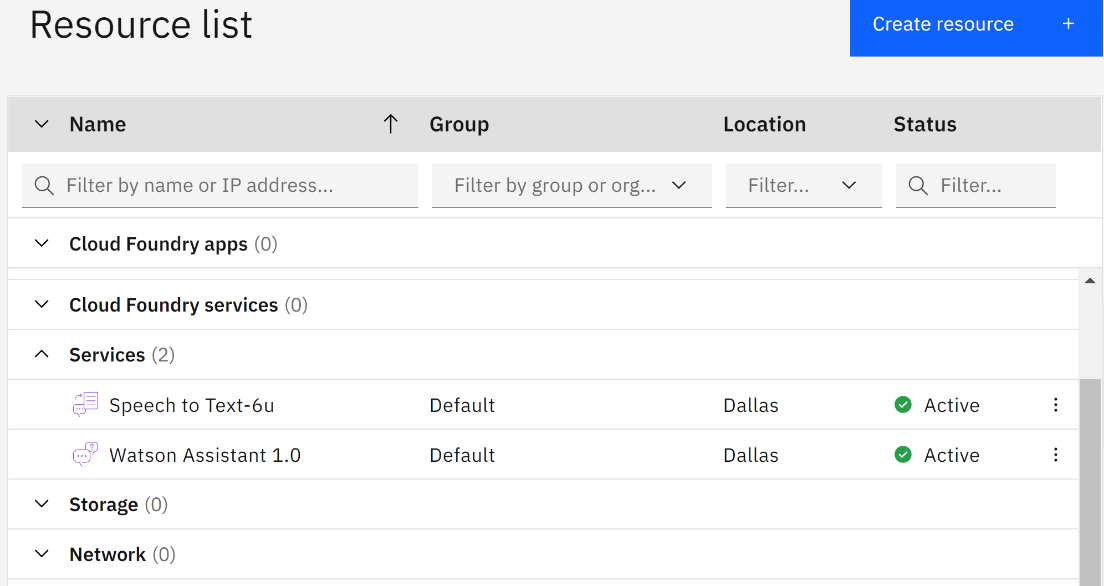


Figure 3 User's Resources in IBM Cloud

Speech to Text-6u – The speech to text resource for IBM Watson®.

Watson Assistant 1.0 – The chatbot agent resource.

* + 1. Watson Assistant 1.0 Overview

An overview of the agent provides high level configurability, by defining service credentials, plan, and connections. The Overview launches the agent, allowing configuration and customization of the agent.

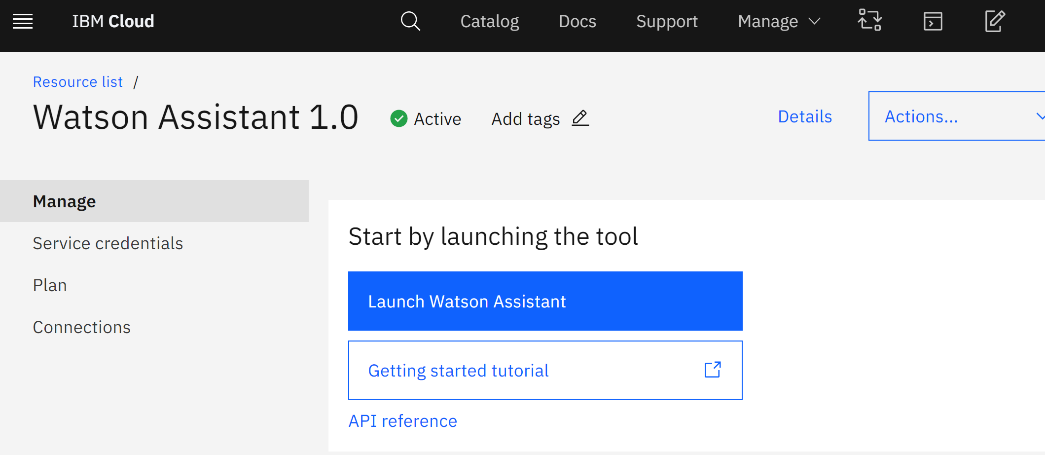


Figure 4 Watson Assistant Overview Page

* + 1. Assistants

Assistants are the various dialogue flows that the agent is able to access. The assistant for the CityChatbot is “City Chatbot v.1”.

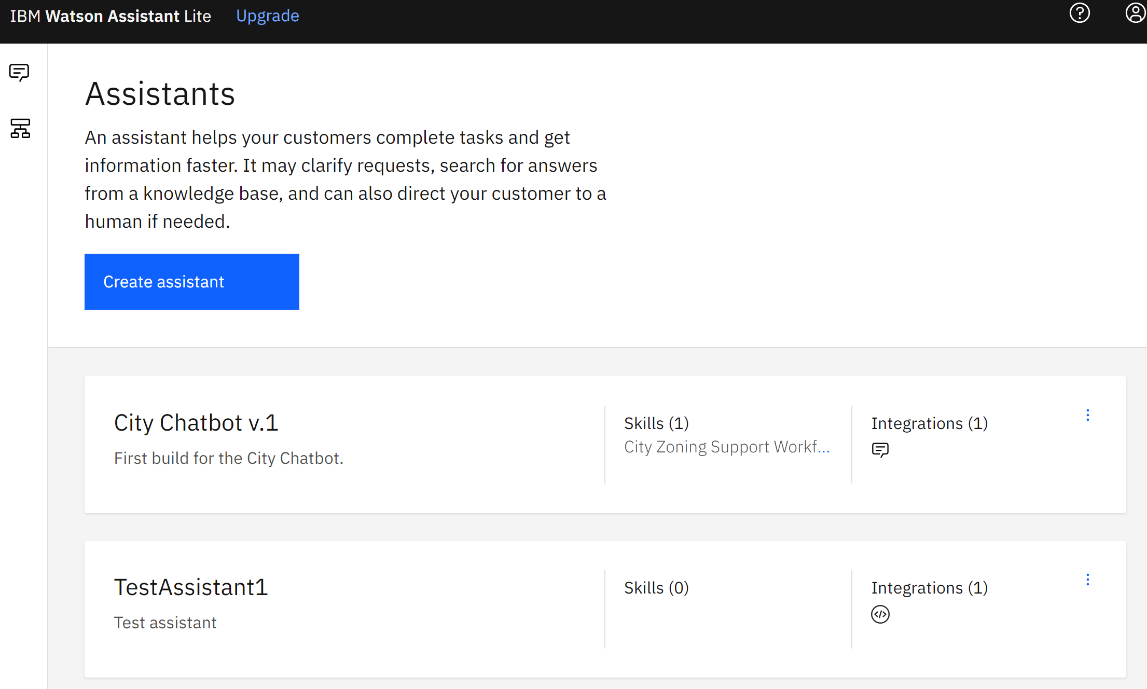


Figure 5 List of User's Watson Assistants

* + 1. Skills

A skill is the dialogue flow for the City Chatbot for zoning information. A separate skill is required if the agent is to access other information for services. The “City Zoning Support Workflow” is the specific skill that provides the zoning functionality of the chatbot.

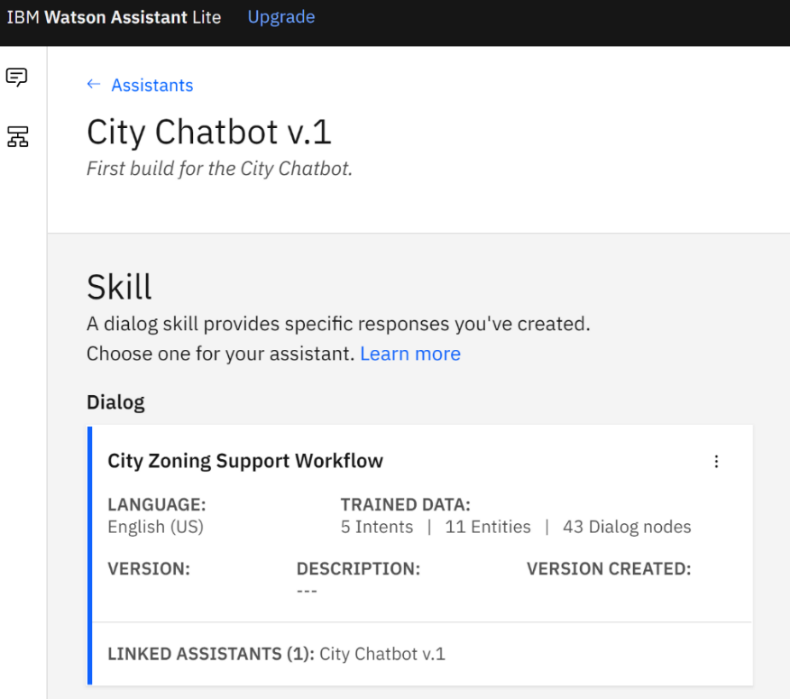


Figure 6 Example Skill of Watson Assistant

* + 1. City Chatbot Configuration

Accessing the Skill allows the agent to be customized and modified.

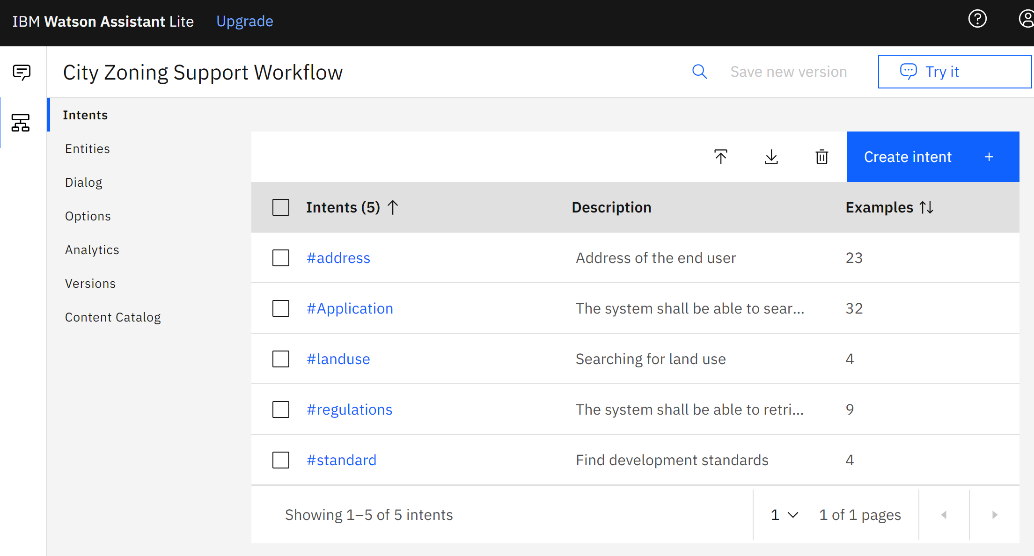


Figure 7 Customize an Assistant's Skill

Intent – An intent is a categorical representation of a body of entities used to guide and organize workflow dialogue.

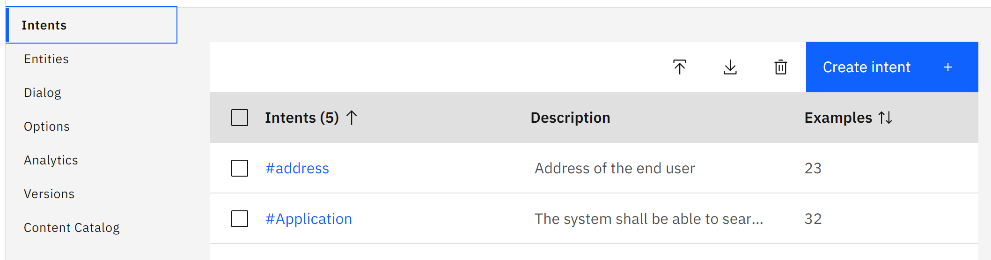


Figure 8 Example Watson Assistant Intent

Entity – An entity is a specific and defined concept appropriate to the defined intent. Entities are the detailed pieces of information such as address, zone, permit type, etc.

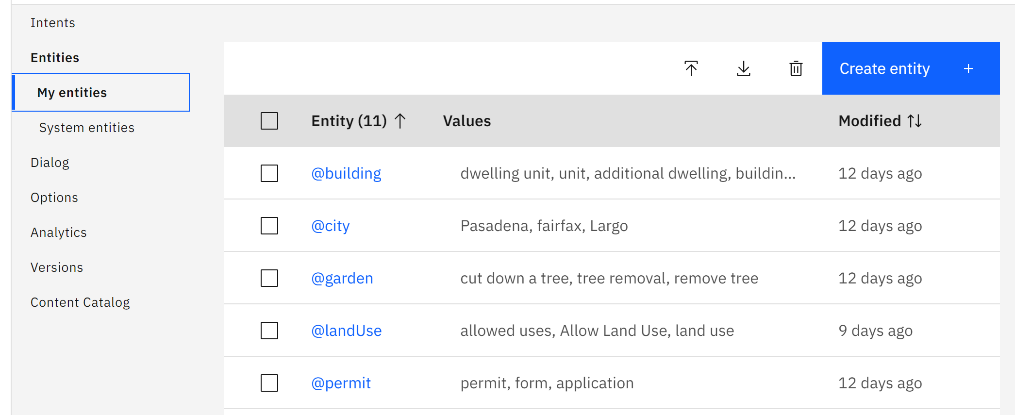


Figure 9 Example of Watson Assistant Entity

Dialogue – Dialogue is the dialogue flow that maps and controls the back and forth between end user and the agent. The Dialogue flow is comprised of dialogue nodes. Each dialogue node is configured for each step to fulfill that steps requirement.

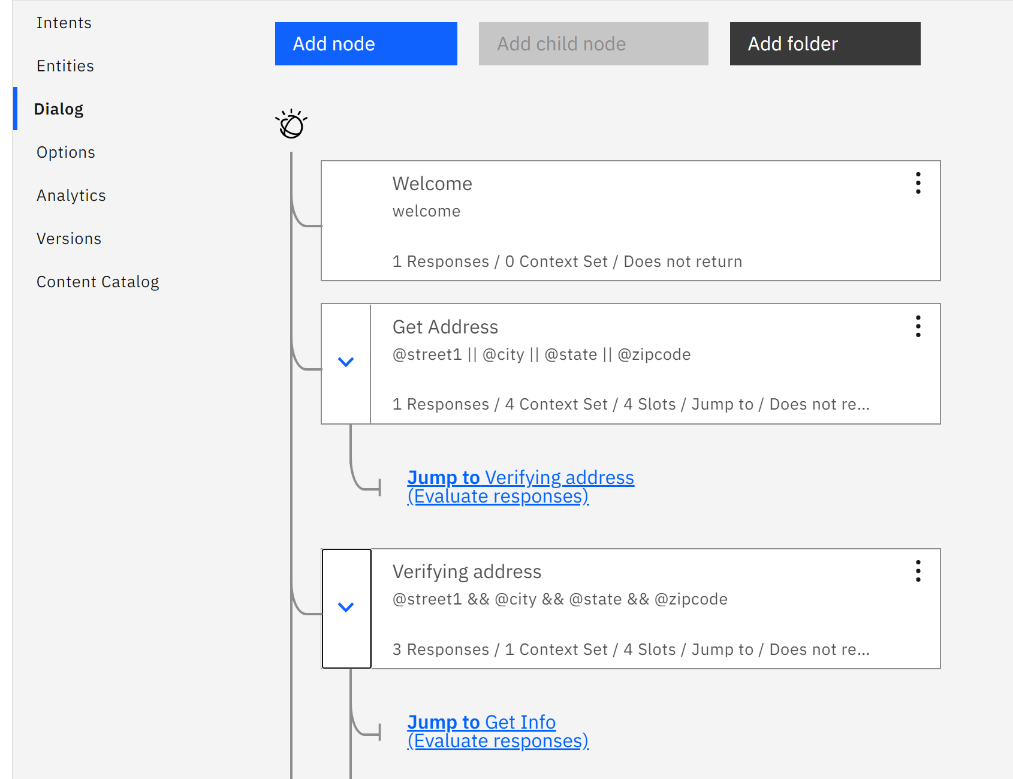


Figure 10 Example Watson Assistant Dialog

The other remaining configuration options are primarily ancillary, and in some cases accessible within dialogue node configuration, entity configuration, etc.

Code Structure

The code structure for this project is utilizing the Spring.IO framework to facilitate faster development and standard processing. In addition to using spring.io the code is broken down into four major categories. Those categories address the individual functionality needed to complete a chatbot action including, processing an address, handling map data to determine a zone, querying the city permit database, and facilitating webhooks between the Java application and the Watson chatbot instance.

Source code

All source code for this project is located in the spring.io GitHub for city chatbot inside the src/main/java/com/chatbot/permit/municipal folder. The source code is divided into the following Java packages:

* controller
* db
* domain
* model
* repository
* service
* watsonactions
* zones

Class and Interface Descriptions

* + 1. Maps

The Maps class is a model of the Maps table in the MySQL database. It contains getter and setter methods for each column in the Maps table. Spring uses this class when auto wiring the MapsRepository interface.

* + 1. Polygons

The Polygons class is a model of the Polygons table in the MySQL database. It contains getter and setter methods for each column in the Polygons table. Spring uses this class when auto wiring the PolygonsRepository interface.

* + 1. MapsRepository

The MapsRepository interface extends the JpaRepository interface from Spring. When used with Spring’s AutoWired feature, it provides methods for updating and querying information from the Maps table in the MySQL database. This interface declares two functions that define two custom database queries. The first one is findByFKPOLYGONID, and it returns a list of all the rows in the Maps table that have the FK\_POLYGON\_ID specified by the id parameter. The second one is findMapsDistinctBy, and it returns a list of all the distinct FK\_POLYGON\_IDs that are in the Maps table.

* + 1. PolygonsRepository

The PolygonsRepository interface extends the JpaRepository interface from Spring. When used with Spring’s AutoWired feature, it provides methods for updating and querying information from the Polygons table in the MySQL database.

* + 1. ParsingService

The ParsingService interface specifies the methods needed to parse the JSON response that the MapQuest Geocode API returns.

* + 1. JsonParsingService

The JsonParsingService class implements the ParsingService interface. It uses Spring’s RestTemplate to make the request to the MapQuest Geocode API and to return the response from the API call as a Java Object.

* + 1. MainController

The MainController class sets up the controller for the application. It defines the API endpoint that is used in the chatbot’s webhook with the PostMapping annotation syntax. Instances of the PolygonsRepository and MapsRepository are created and passed to a MapHandler instance so that it can access those tables in the MySQL database. An instance of ParsingService is created in order to make the request to the MapQuest geocode API. JSON request parameters are used to determine what type of information that the chatbot is requesting. The types of information requests are the following:

* verifyAddress - Returns the zone ID of the address passed to it or -1 if no zone was found for the address.
* retrieveInformation – Returns the permit or regulation information for a given zone ID.
* retrieveZoneSymbol – Return the zone symbol for a given zone ID.
* retrieveStandard – Return the development standards information for a given zone ID.
  + 1. ZonePolygon

The ZonePolygon class inherits the Java Polygon class and adds additional data points for capturing the and associated zone and zone ID for the given polygon.

* + 1. MapHandler

The MapHandler class is used to process a longitude and latitude and return the appropriate zone information for the coordinates. On initiation or zone changes the import feature is used to import industry standard KML files into the database by generating a zone ID for each individual zone and load their associated coordinate points and zone name. When loading the Zone information, the MapHandler adds an additional 2 fields to convert six point decimal coordinates into integer values for future processing. Once the database has been populated, functions are available to load the zones into Java as ZonePolygons in a Java List. Once ZonePolygons are created the MapHandler has a function to process longitude and latitude coordinates and identify which ZonePolygon the coordinates reside in. If a good Zone is found, the Zone ID is returned. If no Zone is found the program returns a -1 to indicate a new address is needed.

* + 1. DBConnection

A class for managing a connection to the MySQL database.

* + 1. ProcessRequest

A class for retrieving permit, regulation, zone symbol, and development standards information from the MySQL database. Uses DBConnection to connect to the MySQL database. Executes different SQL queries to retrieve the correct information to return to the user.

* + 1. WatsonArguments

This class represents the different JSON request parameters that the chatbot can send to the REST API. These request parameters are used to determine what type of information that the REST API is to return to the user. The different parameters include

* street1 – The street address of the user
* webhookType – The type of information that the chatbot requests for. Valid options are
  + verifyAddress
  + retrieveInformation
  + retrieveZoneSymbol
  + retrieveStandard
* object – The entity that the user is inquiring about.
* action – The intent that the user has for the object.
* type – Permit or regulation information
* zoneID – The zone ID of the user’s address
  + 1. MunicipalPermitChatbotApplication

The main class for the application that runs the Spring application.

1. Code Scaffolding

This project makes use of Spring Framework. Spring framework is a configuration and programming model for Java enterprise applications. Spring Framework supports the infrastructure of the enterprise application so that the development team can work on the business logic. Spring is most identified with Dependency Injection. Spring standardizes the configuration and management of references to created objects. This allows for easy testing of the application. Spring manages the Java classes and handles configuration. The team uses Spring to control Database and webhook access.

1. Running the Application

Server (Database)

The Team has chosen MySQL Database Service to create the database on. The MySQL application is a relational database management system. It is open source and uses structured query language. As it is a relational database management system it is used to store, maintain, and retrieve data. MySQL is supported on a wide variety of platforms:

| Operating System | Architecture | 8.0 | 5.7 | 5.6 |
| --- | --- | --- | --- | --- |
| Oracle Linux / Red Hat / CentOS | | | | |
| Oracle Linux 8 / Red Hat Enterprise Linux 8 / CentOS 8 | x86\_64, ARM 64 | X |  |  |
| Oracle Linux 7 / Red Hat Enterprise Linux 7 / CentOS 7 | ARM 64 | X |  |  |
| Oracle Linux 7 / Red Hat Enterprise Linux 7 / CentOS 7 | x86\_64 | X | X | X |
| Oracle Linux 6 / Red Hat Enterprise Linux 6 / CentOS 6 | x86\_32, x86\_64 | X | X | X |
| Oracle Solaris | | | | |
| Solaris 11 (Update 4+) | SPARC\_64, x86\_64 | X | X | X |
| Solaris 10 (Update 11+) | SPARC\_64, x86\_32, x86\_64 |  |  | X |
| Canonical | | | | |
| Ubuntu 20.04 LTS | x86\_64 | X |  |  |
| Ubuntu 18.04 LTS | x86\_32, x86\_64 | X | X |  |
| Ubuntu 16.04 LTS | x86\_32, x86\_64 | X | X |  |
| SUSE | | | | |
| SUSE Enterprise Linux 15 / OpenSUSE 15 | x86\_64 | X |  |  |
| SUSE Enterprise Linux 12 (12.4+) | x86\_64 | X | X | X |
| Debian | | | | |
| Debian GNU/Linux 10 | x86\_64 | X | X |  |
| Debian GNU/Linux 9 | x86\_32, x86\_64 | X | X | X |
| Microsoft Windows Server | | | | |
| Microsoft Windows 2019 Server | x86\_64 | X |  |  |
| Microsoft Windows 2016 Server | x86\_64 | X | X | X |
| Microsoft Windows 2012 Server R2 | x86\_64 | X | X | X |
| Microsoft Windows | | | | |
| Microsoft Windows 10 | x86\_64 | X | X |  |
| Apple | | | | |
| macOS 10.15 | x86\_64 | X |  |  |
| macOS 10.14 | x86\_64 | X | X |  |
| FreeBSD | | | | |
| FreeBSD 12 | x86\_64 | X |  |  |
| Various Linux | | | | |
| Generic Linux (tar format) | x86\_32, x86\_64, glibc 2.12, libstdc++ 4.4 | X | X |  |
| [Yum Repo](https://dev.mysql.com/downloads/repo/yum/) | X | X | X |
| [APT Repo](https://dev.mysql.com/downloads/repo/apt/) | X | X | X |
| [SUSE Repo](https://dev.mysql.com/downloads/repo/suse/) | X | X | X |

Table 2 MySQL Supported Operating Systems (MySQL, N.d)

* + 1. Prerequisites

Microsoft .NET Framework 4.5.2 or later is required when installing MySQL on any Windows OS. It can be found by visiting the Microsoft website. Knowledge of the SQL and database concepts is needed in order to create, to maintain, and to curate databases developed on MySQL.

* + 1. Installation Steps

The newest version of MySQL can be installed from <https://www.mysql.com/downloads/>. If there is difficultly in the installation process the MySQL Installer guide, <https://dev.mysql.com/doc/mysql-installer/en/>, should be referred to.

Once installed the data and structure can be found in the GitHub repository <https://github.com/umgc/umgc.chatbot.db>.

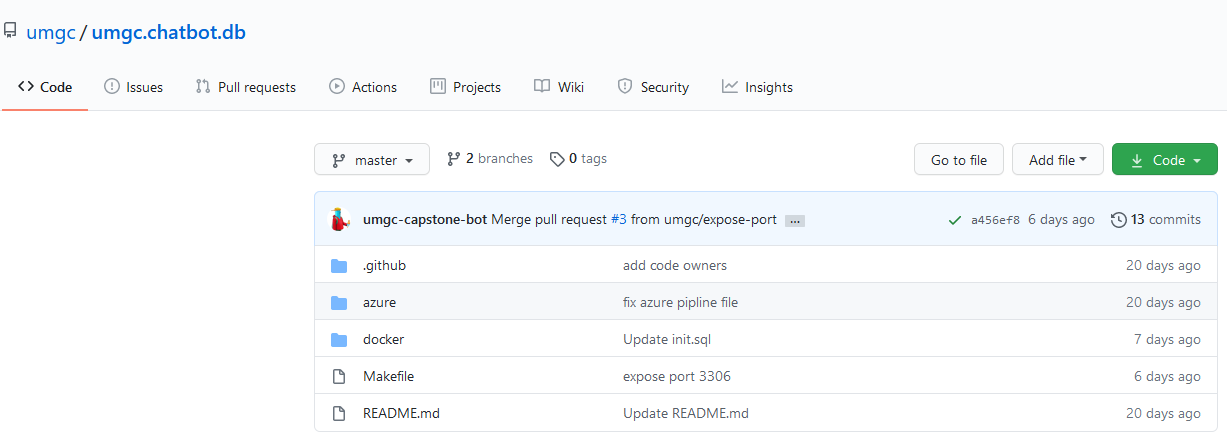


Figure 11 Database GitHub Repository

* + 1. Configuration

The init.sql file, umgc.chatbot.db/docker/include/docker-entrypoint-initdb.d/init.sql, contains not only the data but the structure of the database. Open this file in MySQL and run it.

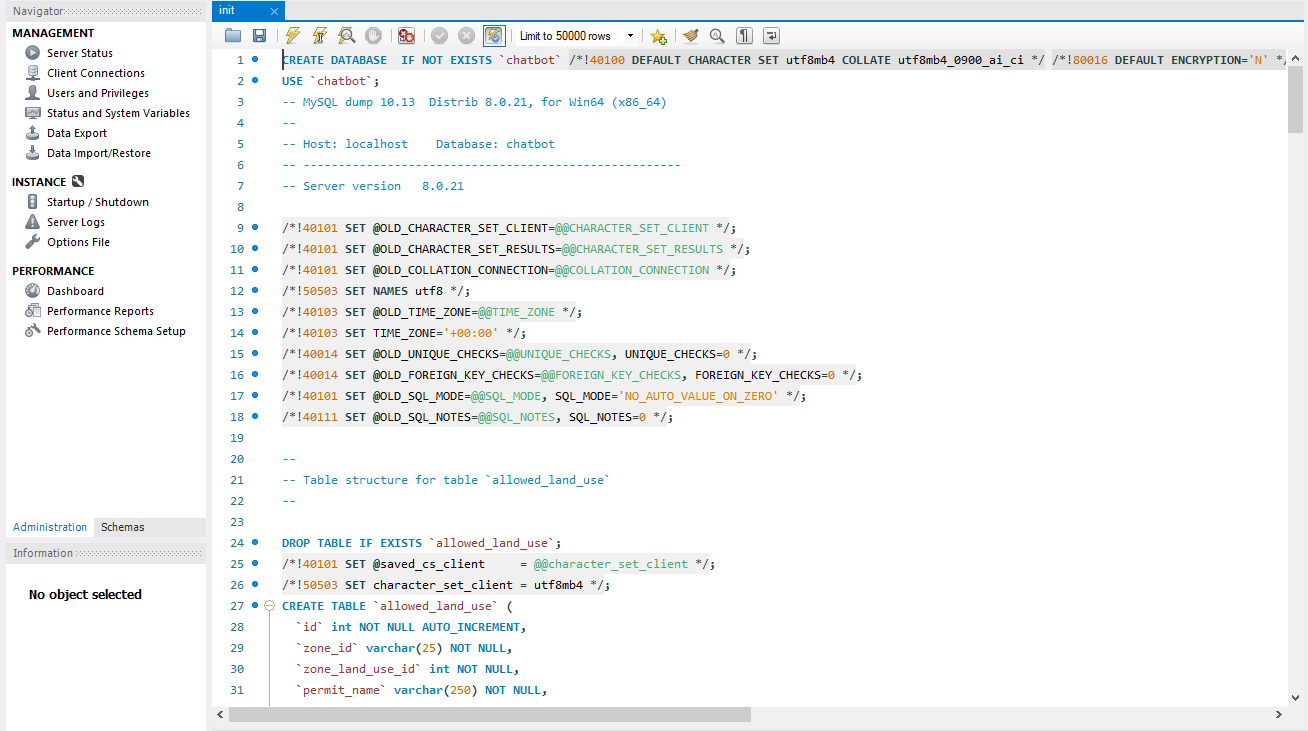


Figure 12 SQL File to Initialize Database

After running the chatbot database will have been created.

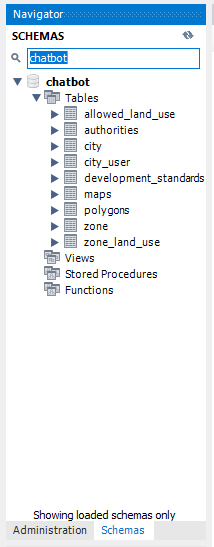


Figure 13 The Tables in the Database

* + 1. Configured Values

As stated above the database configurations value are within the init.sql file. The connection is up to where the user chooses to store the database. Assuming a local host, the user should set up the connection as follows:

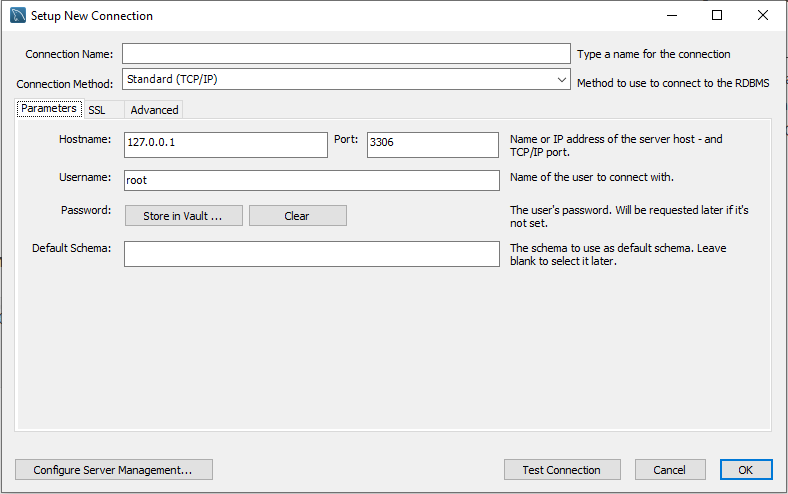


Figure 14 Example Database Configuration

* + 1. Administration

The owner of the data base can and should update the database when new information about the city is made available. There is, as of yet, no backend for this project. Therefore, the administration of said data must be done through the MySQL application.

Server (REST API)

* + 1. Prerequisites

Before running the REST API, the MySQL database must be set up following the instructions listed in section 5.1. The REST API requires a MapQuest Developer API key in order to perform the address verification. Section 3.1.5 has instructions for generating the MapQuest Developer API key. The REST API has successfully been developed on the following operating systems.

* Windows 10
* Ubuntu 20.04

Knowledge of the following programming languages and concepts is needed in order to be able to develop on the REST API application successfully.

* Java 8
* HTTP
* REST API
* MySQL

A system must have the following software and tools installed before starting development on the REST API.

* OpenJDK 8
* Maven 3.6.3

The following packages are dependencies for the REST API application.

* Spring Boot Starter Web 2.3.4
* Azure Spring Boot Starter 2.3.5
* MySQL Connector/J 8.0.21
* Spring Boot Starter Test 2.3.4
* [Spring REST Docs MockMvc](https://mvnrepository.com/artifact/org.springframework.restdocs/spring-restdocs-mockmvc) 2.0.5
* Testcontainers :: JUnit Jupiter Extension 1.14.3
* Testcontainers :: JDBC :: MySQL 1.14.3
* Javax Persistence 2.1.0
* Spring Boot Starter Data JPA 2.3.4
* Mockito Inline 3.6.0
* Mockito Core 3.6.0
  + 1. Installation Steps

The Eclipse IDE can be used to run the application. See sections 3.1.3 and 3.1.4 for instructions on how to download and set up Eclipse. Once Eclipse has been installed, the source code for the REST API can be imported from GitHub into Eclipse using the instructions listed below.

* Select File > Import
* Select Project from Git on the import on the import wizard and select the Next button

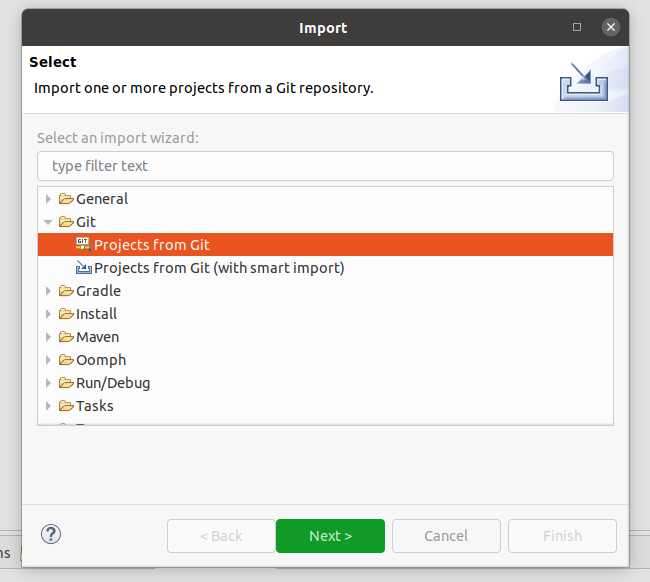


Figure 15 Import from Git

* Select the *Clone URI* option and select the Next button

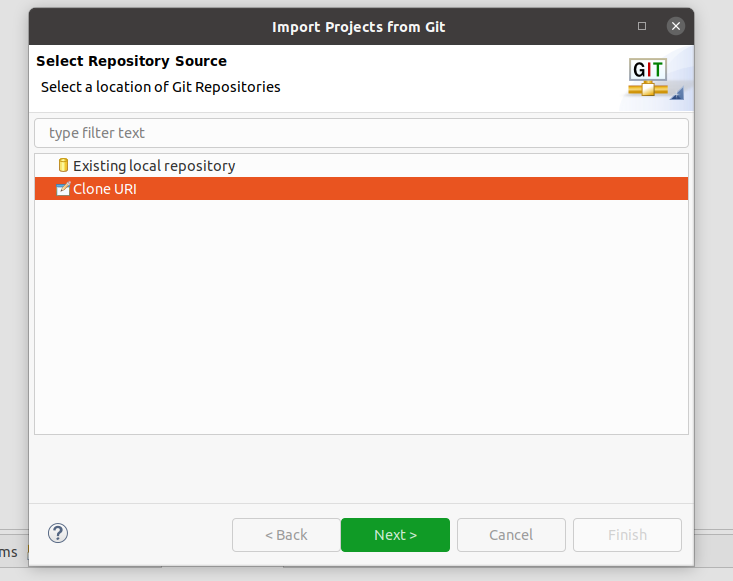


Figure 16 Repository Source

* Enter the URI of the GitHub repository in the URI field. The rest of the fields should auto populate based off of the URI.

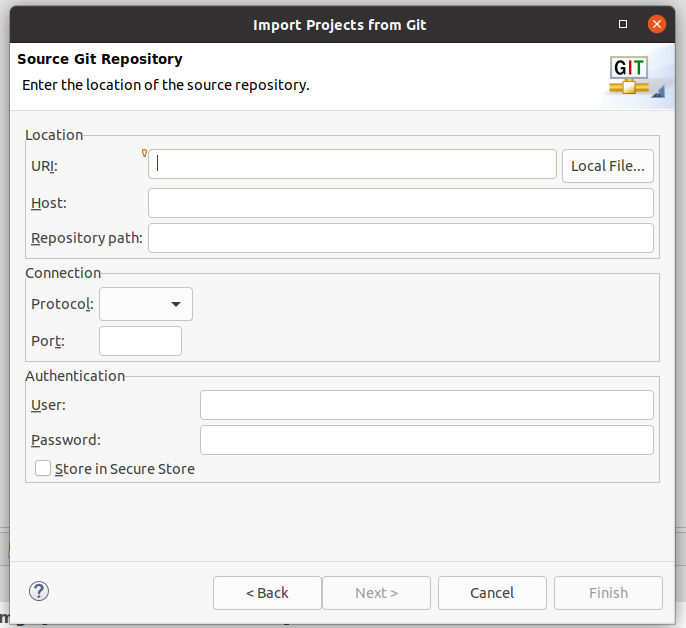


Figure 17 GitHub Repository Information

* Select the branches to pull from GitHub. At a minimum, the master branch should be selected. Click the Next button after choosing the branches to pull.

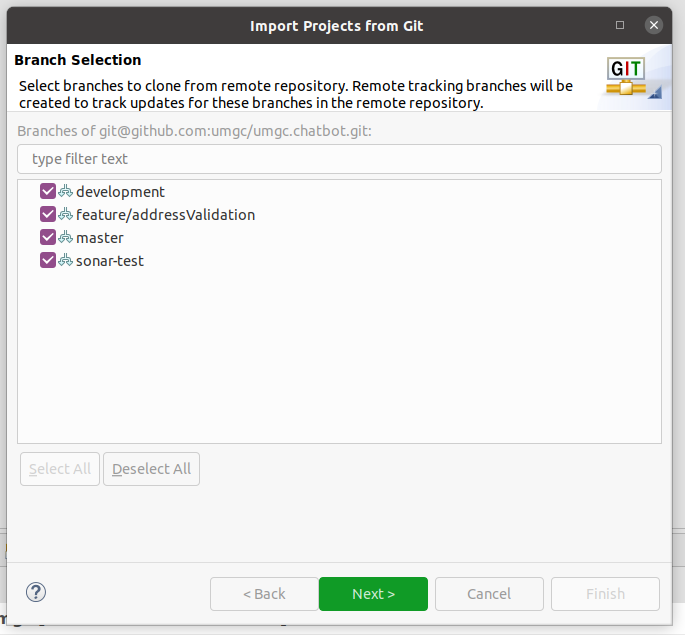


Figure 18 Branch Selection

* Specify the location on the host machine to download the code to. Click the Next button when finished.

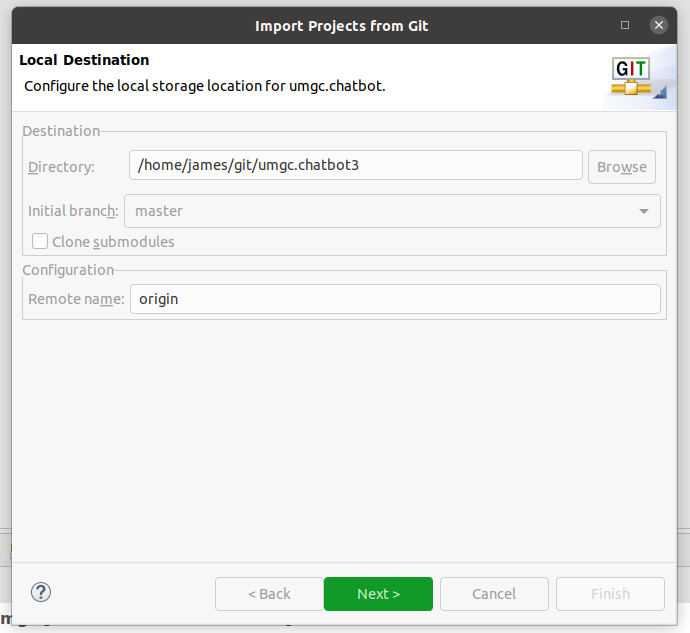


Figure 19 Host Machine Location

* Select the option to *Import as general project*. Click the Next button when finished.

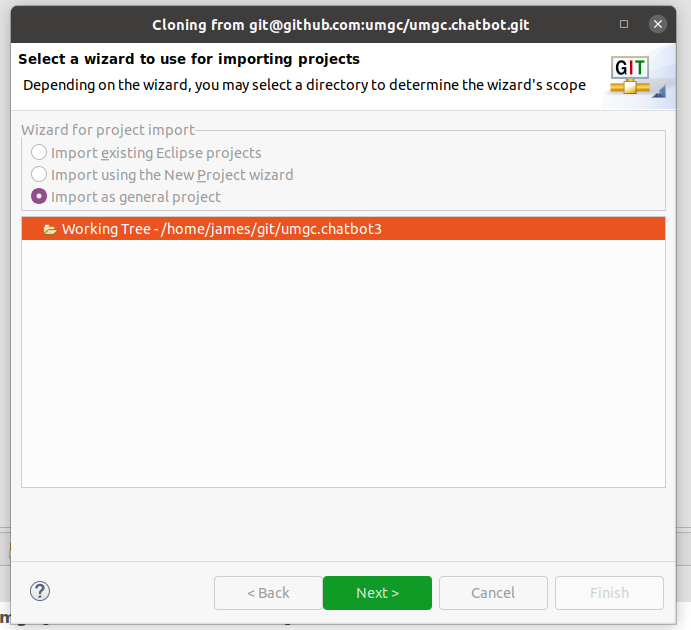


Figure 20 Import Wizard Selection

* Click the Finish button to import the code from GitHub

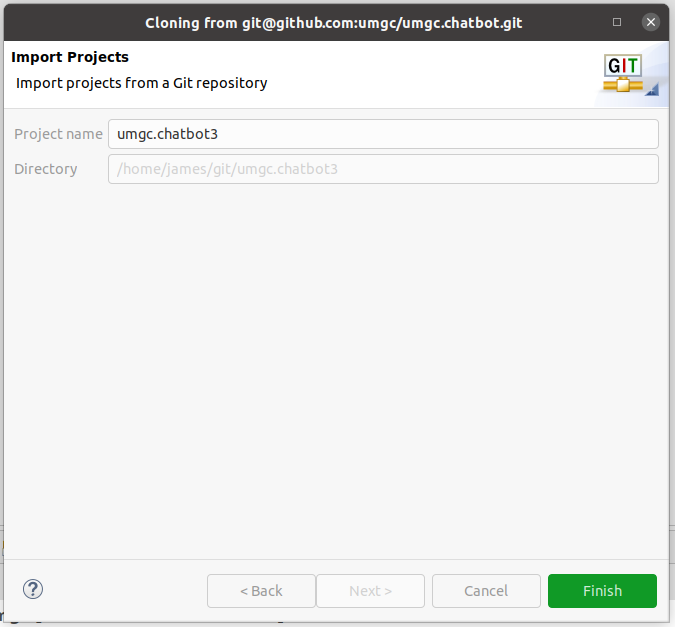


Figure 21 Finish Importing Project

The REST API application can be run directly from the Eclipse IDE using the instructions listed below.

* Right click on the project in the Project Explorer and select *Run As… > Run Configurations…*
* Select the *Java Application* option and click the New Configuration button

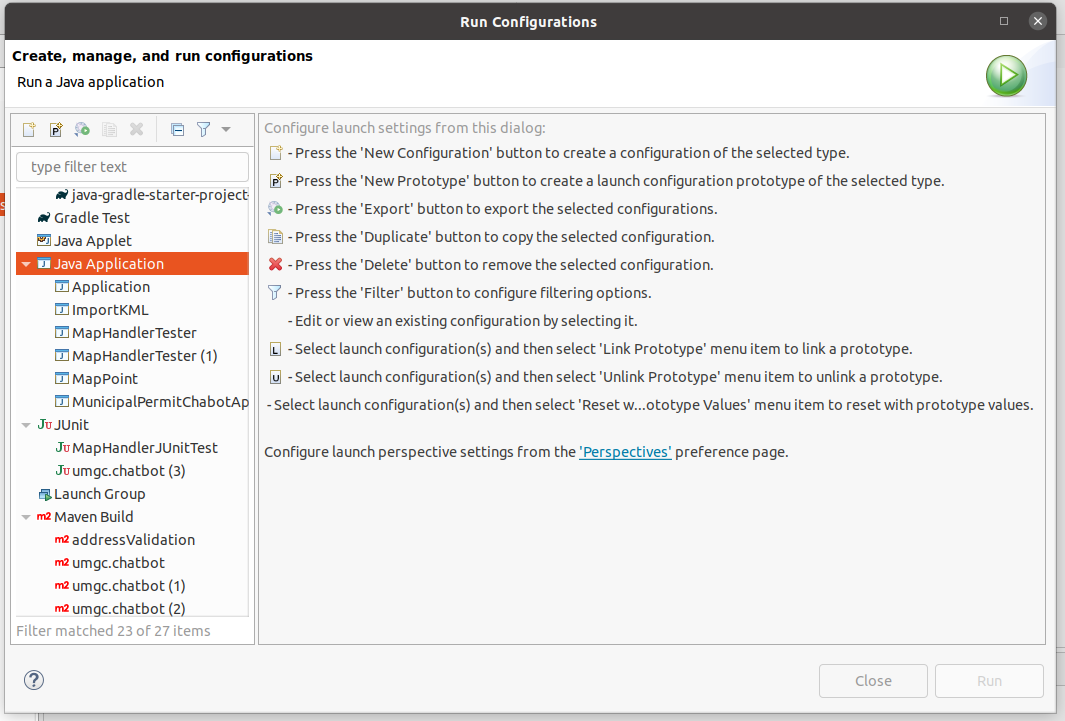


Figure 22 Eclipse Run Configuration Menu

* Enter the REST API project in the Project field and select com.chatbot.permit.municipal.MunicipalPermitChabotApplication as the Main class.

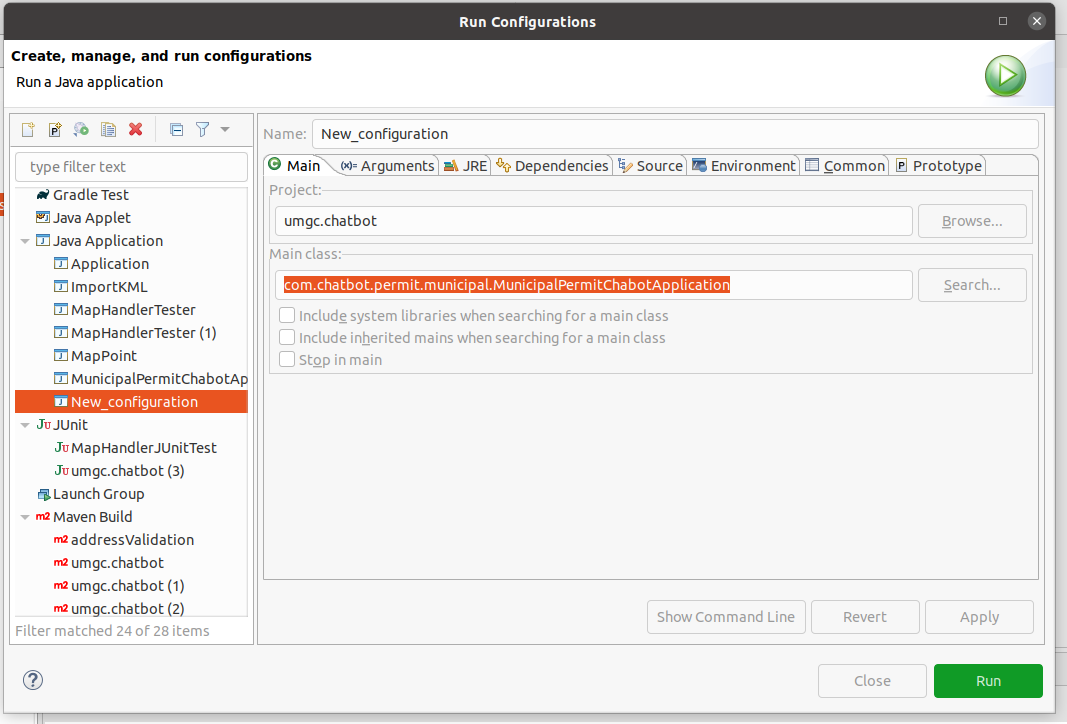


Figure 23 Specify Run Configuration for Eclipse

* Click on the Environment tab. Enter the following environment variables. These environment variables are used to populate the values in the src/main/resources/application.properties file. This file configures the database, MapQuest Developer API key, and the city and state for the application.
  + ADDRESS\_CITY – The city the chatbot is configured for.
  + ADDRESS\_STATE – The state that ADDRESS\_CITY is located in
  + DB\_URL – The URL of the MySQL database that you are using
  + DB\_USER – A username of a user for the MySQL database
  + DB\_PASS -The password of DB\_USER
  + MAPQUEST\_APIKEY – The API key for MapQuest Developer

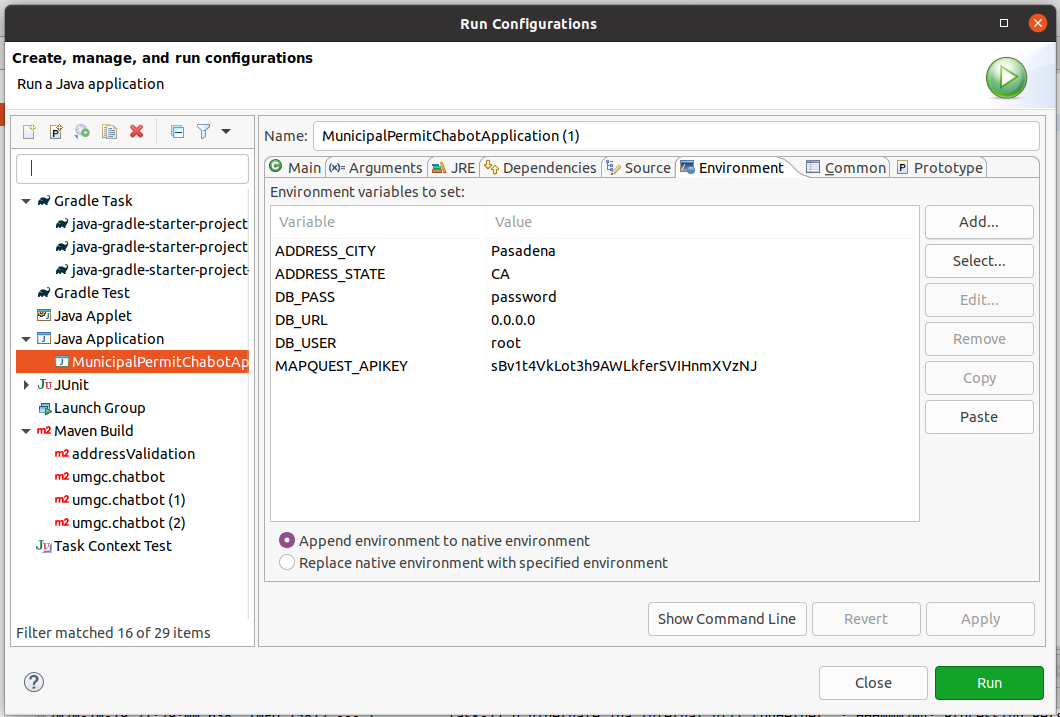


Figure 24 Application Environment Variables

* Click the Run button to launch the application. The application will run on <http://localhost:8080>.

1. Location of Repository

The code for this project and the database for this project is both stored on the UMGC SWEN 670 GitHub repository. That repositories can be found here: https://github.com/umgc/umgc.chatbot and here https://github.com/umgc/umgc.chatbot.db.

1. Challenges/Issues/Concern

The application contains inherent deficiencies that impede development. These deficiencies impact functionality and operability. These challenges are considered systemic, and will require a high volume of work hours to remediate.

* Data Ambiguity and Differentiation – The existing data contains ambiguous identifiers and modifiers without a clear definition of how two pieces of data are distinct. This occurs at key logical branches drilling down through zone and intent to the required base information.
* Bad Data – The existing data contain discrepancies not accurate to real world conditions. The data in the database is not formal Pasadena, CA city data. The data was collected from official Pasadena sources such as websites and publicly available documents. Production use of the application, within an official manner, may require an update to database architecture and related application components.

1. Additional Help

For additional help on this project please reach out to the UMGC Masters in Software Engineering department prior to attempting to contact the city of Pasadena or the team members listed in this document.

1. License Information

The Eclipse IDE and the Spring.IO initialization of the application are managed by the free and open source licenses Eclipse Public License (EPL) which is approved by the Open Source Initiative (OSI) and is not compatible with the inclusion of a software GPL licensed product. More information on the EPL can be found here, <https://www.eclipse.org/legal/epl-2.0/>.

MySQL is managed by the GNU General Public License (GPL). You can find more information on the GPL here, <https://www.gnu.org/licenses/gpl-3.0.en.html>.

The Docker deployment package and application are open sourced projects managed by an Appache 2.0 licenses.

All software developed under this project is deemed to be open source educational work and available for use and modification.

1. Acronyms and Abbreviations

|  |  |
| --- | --- |
| Acronym/Abbreviation | Definition/References |
| AI | Artificial Intelligence – An application that aims to mimic human intelligence. |
| API | Application Programming Interface |
| CH | Chatbot |
| CU | Conditional Use Permit |
| DevSecOps | Development, Security and Operations – Group of developers responsible for the deployment and security of an application. |
| ECUP | Expressive Use Permit |
| EPSP | East Pasadena Specific Plan |
| ETL | Extract, transfer, and load |
| FGSP | Fair Oaks/Orange Grove Specific Plan |
| GIS | Geographical Information System – System for working with geographical data. |
| HTTP | Hyper Text Transfer Protocol – A network protocol for specifying how servers and clients communicate with each other. |
| IDE | Integrated Development Environment |
| KML | Keyhold Markup Language – A markup language for visualizing geographical data. |
| LASP | Lincoln Avenue Specific Plan |
| MCUP | Minor Conditional Use Permit |
| MVC | Model-View-Controller |
| REST API | Representation State Transfer – An API for interacting with data. |
| SRS | Software Requirements Specification |
| UI | User Interface – The part of the application that users use to interact with the application. |
| URL | Uniform Resource Locator |
| WSL | Windows Subsystem for Linux |

1. References

MySQL. (n.d.). Supported Platforms: MySQL Database. Retrieved October 14, 2020, from https://www.mysql.com/support/supportedplatforms/database.html

MySQL. (n.d.). Chapter 1 MySQL Installer for Windows. Retrieved October 14, 2020, from https://dev.mysql.com/doc/mysql-installer/en/mysql-installer.html

IBM, I. B. M. (2020, April 24). IBM Watson, Development Process. IBM Cloud Documents. <https://cloud.ibm.com/docs/assistant?topic=assistant-dev-process>

Google, G. (2020, January 1). DialogFlow Documentation. DialogFlow Documentation. https://cloud.google.com/dialogflow/docs