### 

### TU Dublin - Tallaght campus

# Weather Bot

4th Year project

X00126628 Tudor Voda

Contents

[Project Research Document 3](#_Toc9188396)

[Section 1 Detailed Discussion 3](#_Toc9188397)

[Section 2 Existing Applications in this domain 4](#_Toc9188398)

[Section 3 Platform, Technologies and Libraries 4](#_Toc9188399)

[Section 4 The risks 4](#_Toc9188400)

[Use Cases and Logical Architecture 5](#_Toc9188401)

[Section 1: For Each Use Case: 5](#_Toc9188402)

[Section 2: Logical Architecture 11](#_Toc9188403)

[Logical Architecture Discussion 12](#_Toc9188404)

[Test Report 13](#_Toc9188405)

# Project Research Document

# 

## Section 1 Detailed Discussion

Using Angular for my frontend and NodeJS for my backend. The application will have a bot using Microsoft bot framework, the bot will use LUIS for Language Understanding to understand the user and to answer. The bot will access a weather API called APIXU. The API is coded in Express framework for Node.js. Will then gather and display the weather to the user using Microsoft Adaptive Card. The application uses the browser location with a call to Azure Maps to get the user location. That the users do not need to type in the city. Using a Mongo Database to store the user accounts. The user can register an account on the application and their passwords are encrypted using Bcrypt. The user will have a dashboard that they can edit their profile. The application will be deployed on Azure and the database on Mongo Atlas.

## Section 2 Existing Applications in this domain

|  |  |  |
| --- | --- | --- |
| **Name** | **Similarities** | **Differences** |
| Wear Weather | Detailed information about the weather  Type in location | Bot functionality  Shop that recommends what clothes to wear |
| Brella - Personal Weather | Detailed information about the weather  Notifications | Bot functionality  A picture of thermometer indicating the temperature  Notifications |
| Oshare Weather | Detailed information about the weather  Notifications | Bot functionality  Picture of an avatar what you should wear  Notifications |
|  |  |  |

## Section 3 Platform, Technologies and Libraries

Use Angular for the frontend

Use Microsoft bot framework

LUIS (Language Understanding Intelligent Service)

Express framework for Node.js to call the weather app that the bot will use.

MongoDB and Mongo Atlas

APIXU, weather API

Azure Maps

Microsoft azure for deployment.

## Section 4 The risks

The Microsoft Bot not working or LUIS not working, which means the bot will not answer to the user or it will answer the wrong thing in the worst case the bot will crash. The app is depended on the weather API to work accurate otherwise it will display the wrong information about the. Interface or the bot may not behave the way they should. Everything is a potential risk, as every component support each other to function correctly.

# Use Cases and Logical Architecture

## Section 1: For Each Use Case:

|  |  |
| --- | --- |
| Use Case: | Display weather |
| Actor(s): | User |
| Goal: | The customer gets to see the details about the weather |
| Overview: | The customer accesses the applications, they type in their location and the bot connects to the API and the bot displays the data in a format of an Adaptive Card. |
| Pre- Condition: | Locations exists, Correct name entered. |
| Post- Condition: | Weather display in an Adaptive Card. |
| Successful Scenario: | 1.User opens the application.  2.Enters the locations they want to see weather information.  3.The applications connects to the API and displays the relative information to the user. |
| Alternative Scenario(s): | 1. User opens the application.  2. Enters the locations they want to see weather information.  3. The applications can’t find the location it may be invalid. |

|  |  |
| --- | --- |
| Use Case: | User location |
| Actor(s): | User |
| Goal: | The user clicks a button to input their location without them typing it |
| Overview: | The user clicks the location the button, the application will enter their location in the input box. |
| Pre- Condition: | The user allows location on the browser. |
| Post- Condition: | The application has the location of the user. |
| Successful Scenario: | 1.User allows location on the browser.  2.User clicks the Location button.  3.The application get the user location.  4. Application displays user location. |
| Alternative Scenario(s): | 1. The user has location block on the browser.  2.Application cannot get location. |

|  |  |
| --- | --- |
| Use Case: | Register |
| Actor(s): | User |
| Goal: | The user created an account for the application |
| Overview: | The user will click on the registration button on the navigation bar. The user will enter their email, password and confirm password. |
| Pre- Condition: |  |
| Post- Condition: | User has created an account. |
| Successful Scenario: | 1. The user goes onto the registration page.  2. Enters a valid email, password and confirm password.  3. The application registers the user to the database. |
| Alternative Scenario(s): | 1. The user goes onto the registration page.  2. User enters an email already in the database.  3. User unsuccessful registration to the database. |

|  |  |
| --- | --- |
| Use Case: | Login |
| Actor(s): | User |
| Goal: | The user login on the application. |
| Overview: | The user will click on the login button on the navigation bar. The user will enter their email and password. |
| Pre- Condition: | User has an account in the database. |
| Post- Condition: | User has access to their account. |
| Successful Scenario: | 1. The user goes onto the login page.  2. Enters their email and password.  3. The application checks the credentials against the database  4. User is given access to their account. |
| Alternative Scenario(s): | 1. The user goes onto the login page.  2. Enters their email and password.  3. The application checks the credentials against the database.  4. User credentials are incorrect.  5. User is denied access. |

|  |  |
| --- | --- |
| Use Case: | Access the Database |
| Actor(s): | Admin |
| Goal: | Admin access the database to add, remove or change the entries in the database |
| Overview: | The admin login the database which will allow the admin to add, remove or change any data that is stored in the database |
| Pre- Condition: | Have an admin account |
| Post- Condition: |  |
| Successful Scenario: | 1.Admin login to the applications  2.The admin connects to the database  3.The admin can add, remove or change the database |
| Alternative Scenario(s): | 1.Admin login to the applications  2.The admin connects to the database  3.The login details enter are wrong, refuse connection |

|  |  |
| --- | --- |
| Use Case: | User updates their profile |
| Actor(s): | User |
| Goal: | User updates their favourite quote |
| Overview: | The user accesses their account and updates their quote. |
| Pre- Condition: | User has an account. |
| Post- Condition: | User has a new quote. |
| Successful Scenario: | 1. The user logins in the application. 2. Goes to their dashboard. 3. Types the new quote and updates the quote |
| Alternative Scenario(s): | 1.User enters invalid credentials.  2.Application denies access to the user. |

## 

## Section 2: Logical Architecture

Software Components: UI connects to the Microsoft bot, with connection to the database

Databases: MongoDB

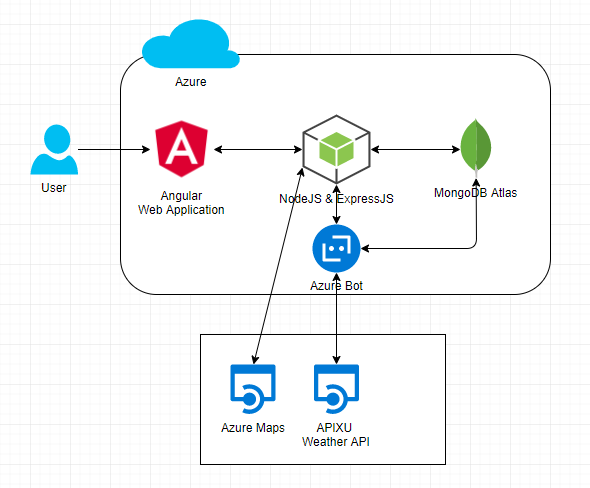
App engines: Angular 7, Microsoft bot framework

Mobile platforms: All

API’s and languages: APIXU Weather API, Azure Maps, Typescript, NodeJS and ExpressJS

Deployment EG RESTful JDBC session less: RESTful

Security e.g. Https certifications authentication: Azure security and Bcrypt for passwords



## Logical Architecture Discussion

The user accesses the application that is deployed in azure. The use can use the webchat to ask the bot what the weather is. The bot will connect to an API from the backend to retrieve the data. Data is in json format and the application will display the request information to the user in the frontend. The weather is displayed on a Microsoft Adaptive Card to the user. The user can click get location button for the application to type in their location. The application is connected to Azure maps to get the user location using longitude and latitude. The user can also register an account to the database, allowing them to edit their profile.

# Test Report

|  |  |
| --- | --- |
| **Test Plan Summary** |  |
|  |  |
| %pass | 95.45 |
| %fail | 4.55 |
| # of tests performed | 22 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Plan Detail** |  |  |  |  |  |  |
| **Test Type** | **Test Objective** | **Screen shot** | **Steps** | **Test data** | **Expected result** | **Pass/Fail?** |
| Unit Test | App componet is created |  | 1. Write test in app.component.spec.ts 2. Run the ng test command | Componet itself | Create the app | Pass |
| Unit Test | AuthGuard service created |  | 1. Write test in auth.guard.spec.ts 2. Run the ng test command | Componet itself | Create service | Pass |
| Unit Test | BotService service created |  | 1. Write test in bot.service.spec.ts 2. Run the ng test command | Componet itself | Create service | Pass |
| Unit Test | AuthService service created |  | 1. Write test in auth.service.spec.ts 2. Run the ng test command | Componet itself | Create service | Pass |
| Unit Test | Dashboard component created |  | 1. Write test in dashboard.component.spec.ts 2. Run the ng test command | Componet itself | Create dashboard page | Pass |
| Unit Test | Home component created |  | 1. Write test in home.component.spec.ts 2. Run the ng test command | Componet itself | Create home page | Pass |
| Unit Test | Login component created |  | 1. Write test in login.component.spec.ts 2. Run the ng test command | Componet itself | Create login page | Pass |
| Unit Test | Logout component created |  | 1. Write test in logout.component.spec.ts 2. Run the ng test command | Componet itself | Create logout function | Pass |
| Unit Test | Register component created |  | 1. Write test in register.component.spec.ts 2. Run the ng test command | Componet itself | Create register page | Pass |
| Unit Test | UserService servuce created |  | 1. Write test in user.service.spec.ts 2. Run the ng test command | Componet itself | Create service | Pass |
| Security Test | Registration outside the constraints |  | 1. Enter email  2. Enter password | Input email and password | Fail to register | Pass |
| Security Test | Register successful to the application |  | 1. Open postman, enter the URL and the email and password in the body 2. Write the test. 3. Send the request to the backend | email= test@test.com password= test123 | User register to the database | Pass |
| Security Test | Login successful to the application |  | 1. Open postman, enter the URL and the email and password in the body 2. Write the test. 3. Send the request to the backend | email= test@test.com password= test123 | User login into their account | Pass |
| Security Test | Get user data |  | 1. Open postman, enter the URL 2. Write the test. 3. Send the request to the backend | GET request | Displays the user information  to the user | Pass |
| Security Test | Update user quote |  | 1. Open postman, enter the URL and the value=quote in the body 2. Write the test. 3. Send the request to the backend | value=Hello | User successful updates  their quote | Pass |
| Security Test | User is logged in |  | 1. Open postman, enter the URL 2. Write the test. 3. Send the request to the backend | GET request | User is logged in | Pass |
| Security Test | Users logout |  | 1. Open postman, enter the URL2. Write the test.3. Send the request to the backend | GET request | User successful logout and session is destroyed | Pass |
| Security Test | Uses user location to get their city name |  | 1. Open postman, enter the URL and the latitude and longitude in the body 2. Write the test. 3. Send the request to the backend | latitude= 53.3389312 00000005 longitude= -6.2734336 | Displays the user city | Pass |
| Security Test | Get a conversation ID from the bot |  | 1. Open postman, enter the URL 2. Write the test. 3. Send the request to the backend | type = message from = id:user1 text = hello | Get conversation ID from the  microsoft bot | Pass |
| Security Test | Send a message to bot and get an id back |  | 1. Open postman, enter the URL 2. Write the test. 3. Send the request to the backend | convId = HQu7yYf9DrIGDdJI0LPQvW-9 message = hello | Send a message to the bot, bot replies with an ID | Pass |
| Security Test | Receive a message from the bot and watermark | |  | | --- | |  | | 1. Open postman, enter the URL 2. Write the test. 3. Send the request to the backend | GET request | Receive message from bot and watermark. Display the message to the user | Pass |
| Scalability Test | Test azure scalability |  | Enable autoscale on azure | Azure web application | Azure scalability response well at high and low demand | Fail |

**Code Coverage**

