# Landmark Remarks

Alister Holland

30/04/2019

# Summary

I designed this solution to adhere to the user stories provided. The base project was generated with the React project template for .NET Core. This uses create-react-app and includes a basic UI framework (reactstrap). I was limited by reactstrap so also included Material-UI for other basic UI components. Both UI frameworks adhere to Material Design, so they work seamlessly together. For the mapping, I used react-google-maps. This proved simple and functional, but I ran into webpack issues attempting to add extras such as labels to the map pins. I used xUnit for unit testing. Note: I have never used react before.

Some implicit requirements I uncovered were:

- We need to use a third party API for mapping functionality
- We need a user authentication system
    - It needs to have some standard of security
- We need a database to store user data and remarks
- Current user's remarks must display differently to other user's remarks (user story 3)

# Build and Run

Opening the solution in Visual Studio should install all required dependencies. If not, you may be required to run 'npm install' from within the ClientApp/ directory. The database should be available and populated; if not, you will need to run 'dotnet ef database update'.

When you run the solution, you should be presented with a login screen. Use one of the following credentials to login:

- User: test1@example.com Pass: testing123
- User: test2@example.com Pass: 321testing

You will be directed to the Map screen, which will autofocus on your current location. Note you will need to give your browser geolocation access at the prompt. Here you can add a new remark, view all remarks on the map, or go to the remarks screen. At the remarks screen you can search for remarks by username or by note text. Remarks with a blue border are your remarks; remarks with an orange border are other peoples' remarks.

## Limitations

- The architecture is not very scalable. As it is such a small project, I have not separated the business logic and data layers. Some ways to make it more scalable could be to use multi-tier architecture and/or use CQRS (Command Query Responsibility Segregation).
- The user is unable to delete or update remarks.
- I have never used react before, so wasn't comfortable using a state management solution such as Redux, given the time-frame. The front-end could be made much cleaner if I used a state management solution.
- As I was using Mac OS, I was unable to use LocalDB for storage; as such, I had to use SQLite.
- The authentication system is very simple due to time constraints and passwords are stored without salts, which comes with a higher risk dictionary attacks. A better solution would be to use ASP .NET Identity.
- Keys are stored in config file; in production they should be stored in environment variables

## Time Spent

Planning & Architecture: 2 hours

Development: 10.5 hours

Documentation: 1 hour

**Total: ~13.5 hours**