ALLERGY ADVICE

Technical Report

BSc in Computing

Niall Whelehan, x12102954

Allergen Advice

TABLE OF CONTENTS

EXECUTIVE SUMMARY

A common problem for those affected by serious allergy afflictions is that the rules and laws governing food packaging or general awareness of contaminants vary from country to country. Allergen Advice aims to act as a travelling companion for food related allergy sufferers.

Using a simple to use user interface, app users can select their own countries food laws and regulations and compare them with selected destinations. Additional features include a phrasebook, listing common allergens in multiple languages, a community area where people can recommend and view suggested restaurants and a smartwatch extension that alerts a user when they are nearby such a restaurant. While the phrasebook exists in other translation style applications, the allergen information on a country by country basis does not exist currently on the Play Store.

As a parent of a child with a severe nut allergy I am acting as one of my own target audience. I love this idea, and the colleagues and friends I have shared it with have endorsed the project as something they would use and recommend to families affected.

A closed beta of the application has been viewed, updated and reported on by close friends and colleagues as well as a group of mothers of children suffering from serious food allergies.

The goal of this report is to highlight the risk of such allergies as well as to show how easy, and cheap it can be to create an application not for monetary gain.

INTRODUCTION

This document is presented to include all the research and findings made so far in the Allergen Advice project. The current content has changed, in some places drastically, since the beginning of the project and is likely to change again if and when better implementations or user experiences are defined.

BACKGROUND

THE CONCEPT

As mentioned above, I am the father of a serious allergy sufferer. It is a terrifying fact and recurring nightmare for a parent that there is a food that could cause

such catastrophic damage to their child. Fueled by this fear my wife and I have done our research on the specific allergen, have become hyper vigilant while shopping to check all ingredients and warnings, and prepared his bag to always have his adrenaline pen close should something happen. This is what I suspect most parents would do in our situation. Feeling somewhat secure in our knowledge of Ireland's laws surrounding this subject is one thing, planning a family holiday abroad is another.

Food laws differ greatly from country to country. Some countries do not even have food laws and regulations. The idea for this application was conceived when I read an article about the risks prevalent in the United States for allergy sufferers due to the fact that only eight of the commonly known allergens must be displayed on the packaging of food items. Compare that with the fourteen mandatory displayed items in the EU and you already have six hidden killers in American food labels.

GOALS

My plan for this project was to create a philanthropic application that raises awareness of the dangers of food allergies and offer a simple reference for those affected. As of January, 2015, I have been working as a backend developer for a mobile application company. I will take what I have learned and use it to develop my idea. On a personal development level, I also wanted to use this project to try out new technologies that I might try bring to my job. The eventual goal for me with this project is to get it uploaded to the Play Store. From there I can review feedback and try to expand on the original concept.

** Update on this: I successfully setup the Allergen Advice application on the Play Store and have gathered a new insight into the process of builds and deployment, as well as gathering feedback in a constructive manner.

TECHNOLOGIES

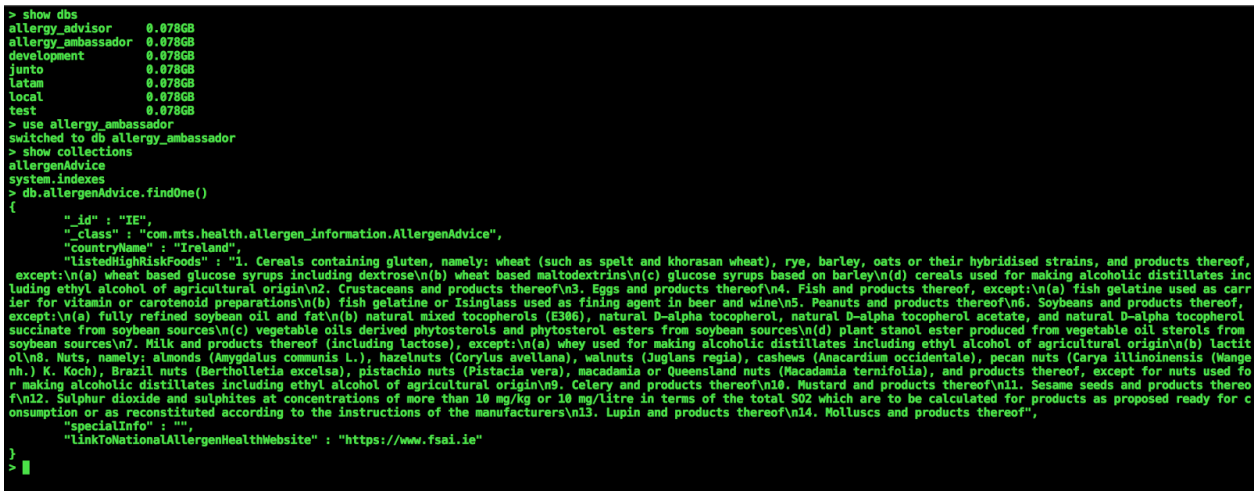
CLOUD SOLUTION

The project is hosted using Amazon Web Services. The suite provided by Amazon make everything so much easier. Using Amazon Elastic Compute Cloud (Amazon EC2) allows me to scale my project as the scope requires. The cost of the service is free for the first year, so long as an agreed bandwidth of requests is not being processed.

It is my intention to switch to an Elastic Load Balancing (ELB) server closer to the final deadline. This offers some advantages for not much more in price. The main advantage of an ELB is that additional servers can be called in to use if the application saw a significant increase in traffic. As well as this, ELB servers have a built in failover that automatically direct traffic to another server should the main one fail.

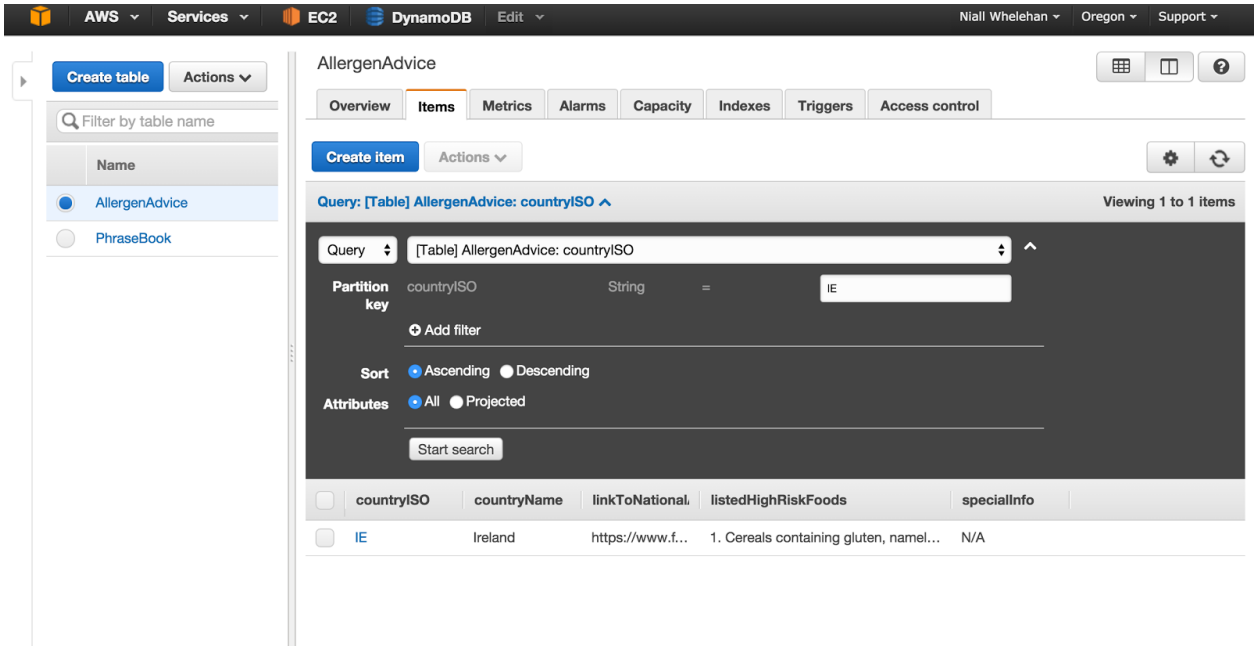
DATABASE

Originally I used MongoDb as a database because of how easily the structure of the database could be updated when new features were added. As well as this, mongo is a light service that is quick and easy to use.



Mongo is based on documents, instead of tables. Like Relational databases, each document must have a unique _id. The benefit of MongoDB is that the design is schema less, and the project code defines the structure of the db. For a project that is updated as often as a mobile application, this is a great feature. Having the documents stored in JSON format makes life a lot easier when communicating using Rest Services.

With the risk of Amazon tearing down a running service for maintenance purposes I migrated my implementation to use DynamoDb, Amazon's own Item based (similar to a mongo document) data solution. The database is now hosted on a dedicated server that will be automatically backed up before any maintenance might take place.



For the most part, DynamoDb is very similar to Mongo. Mongo has more features and a better query language, but it is complicated to host. Dynamo does the job that I need with the extra security guaranteed by Amazon.

Having now worked with DynamoDB for a number of months my opinion is that it does not stand up to MongoDB. It was educational and I learned a lot about alternative ways of marshalling data but it is a lot less user friendly than MongoDB and has poorer sources of community support.

MOBILE BACKEND

The backend runs a Java encoded tomcat instance with Spring Frameworks.

Spring is a framework based on two core concepts:

Dependency Injection

Aspect Oriented Programming

Dependency Injection allows for the management of object lifecycles, from instantiation to destruction. Using Spring Beans to define id's and reference points, interfaces can be used to design the flow of the working server.

Aspect Oriented Programming deals with the idea of 'aspects'. An aspect is the concept of a component that operates across different levels of the code base but is not directly part of any of the areas touched. The best example of this is a security controller that is used to return advice once an error arises in the program. This could be implemented to be triggered by a rest call being handled

wrong, or a memory leak. Neither are necessarily related, and you could not say that a security controller is an extension of Rest controller.

Tomcat is an open source web server that takes a war file that is generated from the Java project, and runs it in a controlled environment.

CLIENT:

The front end is an Android application. My reasoning behind this is that I am an Android user and there is a strong link with Java and Android. I presumed I would be pressed for time so learning iOS is not something I thought I would be able to fit in.

The app makes use of a third party library called Retrofit. Retrofit takes the projects API and turns it into a Java interface.

REQUIREMENTS

The following is a revised edit of the previously submitted Requirements Spec. Allergen Advice has evolved as a project in many ways. The database, instead of being a locally deployed MongoDB instance is now a DynamoDB stand-alone server hosted by Amazon. The configuration of the backend has been refactored from XML base to Java and annotations. New functionality has been added and some taken back to better reflect the goal of the application.

I think the most significant requirement of a mobile application in the current market is changeability. The market moves very quickly and ratings are the deciding factor on whether or not an application will be downloaded. If the application cannot be hot fixed and redeployed quickly after receiving negative feedback about a bug or bad user experience, then the ratings will reflect the issue and the app will disappear into the lower rankings.

For this reason, I have refactored the code to allow for faster deployments, more reliable hosting environments and an overall more scalable project.

FUNCTIONAL REQUIREMENTS (MOBILE APPLICATION)

FUNCTION: “SCREEN NAVIGATION”

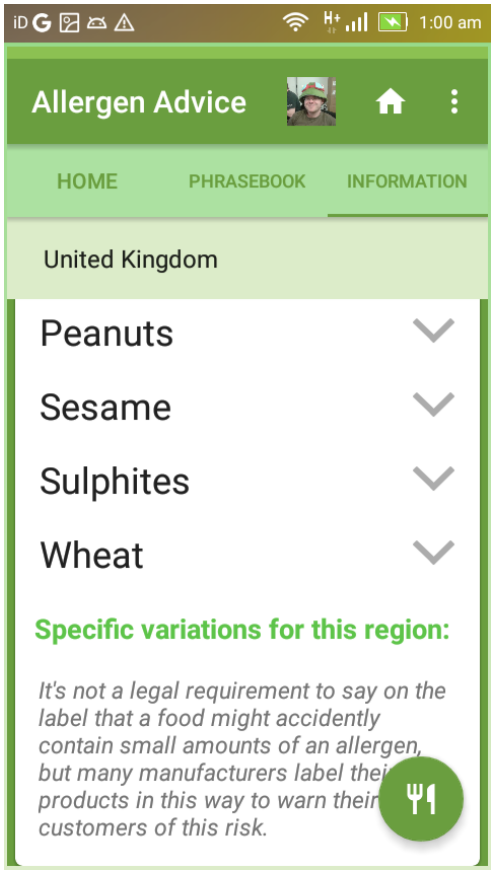
Input: User swipes or presses tab buttons to navigate main screens.

Action: The client should process the request and direct the user to the desired activity or fragment. The fragment should be pre-loaded so as not to slow the navigation down.

Output: The screen should now show the page the user wanted to access.

Priority: Very important. The app should be intuitive, and poor navigation can ruin any user experience

FUNCTION: "RETRIEVE ALLERGEN FOOD LAWS INFORMATION BY COUNTRY"



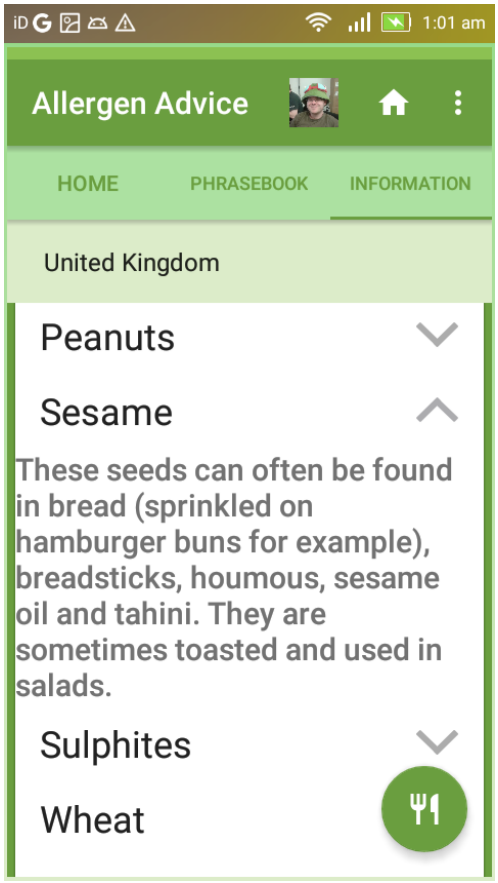
Input: User has country of origin configured from initial setup.

Action: The client should make a request to the backend requesting information based on country.

Output: A table is returned displaying the relevant information.

Priority: The core functionality of the app. Critical priority.

FUNCTION: “PROVIDE NEAT ANIMATIONS TO ALLOW USER TO PICK AND CHOOSE WHAT INFO THEY WANT TO READ”



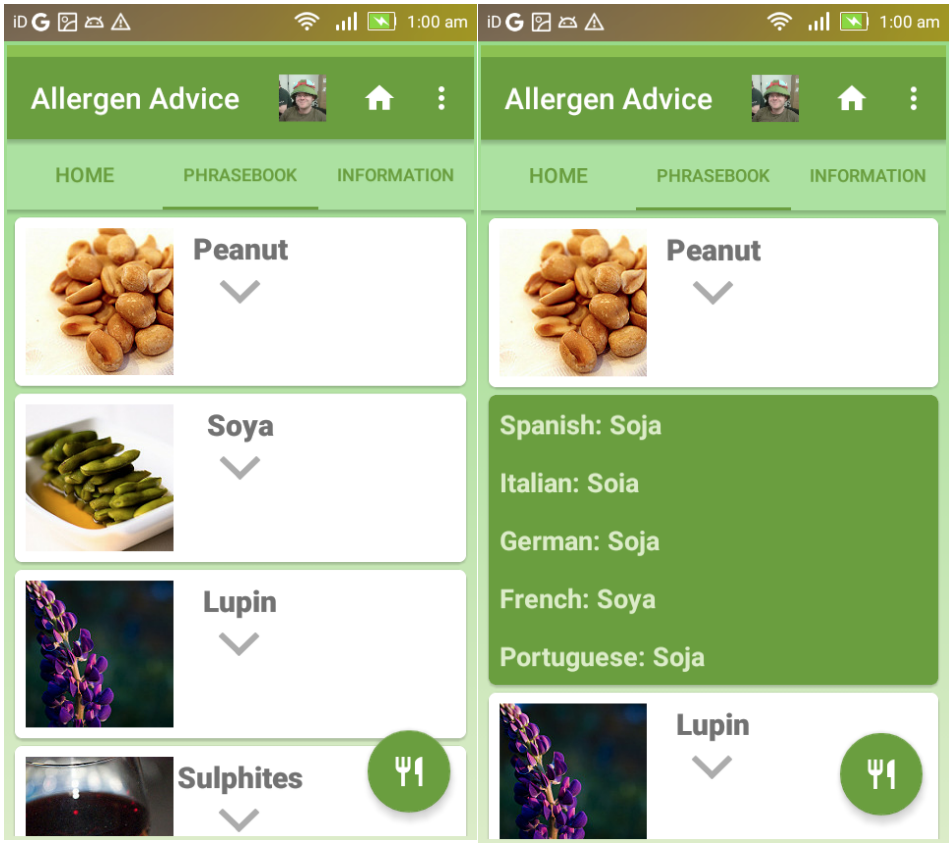
Input: User selects country from drop down spinner

Action: The selected information is returned. The specific allergens can be expanded to read more info

Output: A text is returned displaying the relevant information.

Priority: Very important. This is a natural progression in the thought processes of what this app should do and it is what the user would expect to be available.

FUNCTION: "PHRASEBOOK TO TRANSLATE COMMON ALLERGEN FOODS"



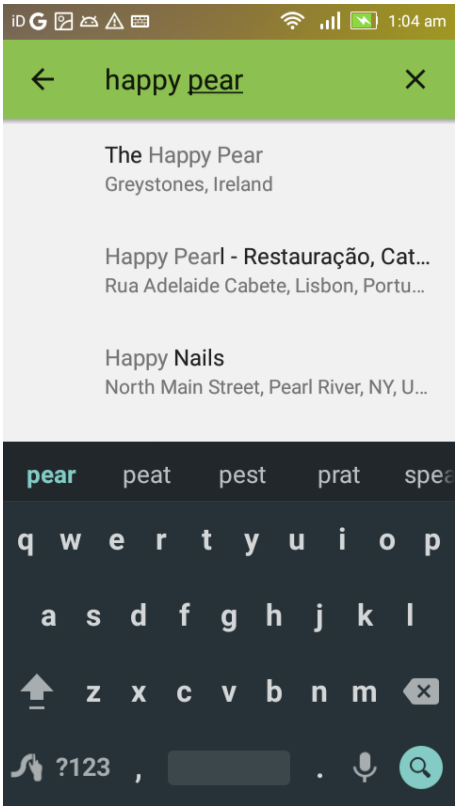
Input: User selects phrasebook from navigation bar and selects the language they want.

Action: The application should send the request to the backend based on the selected language and the user's spoken language (configured at initial setup).

Output: A comparative list of translations will be displayed

Priority: A very nice feature to have. Not top priority.

FUNCTION: "RESTAURANT RECOMMENDATION SERVICE"



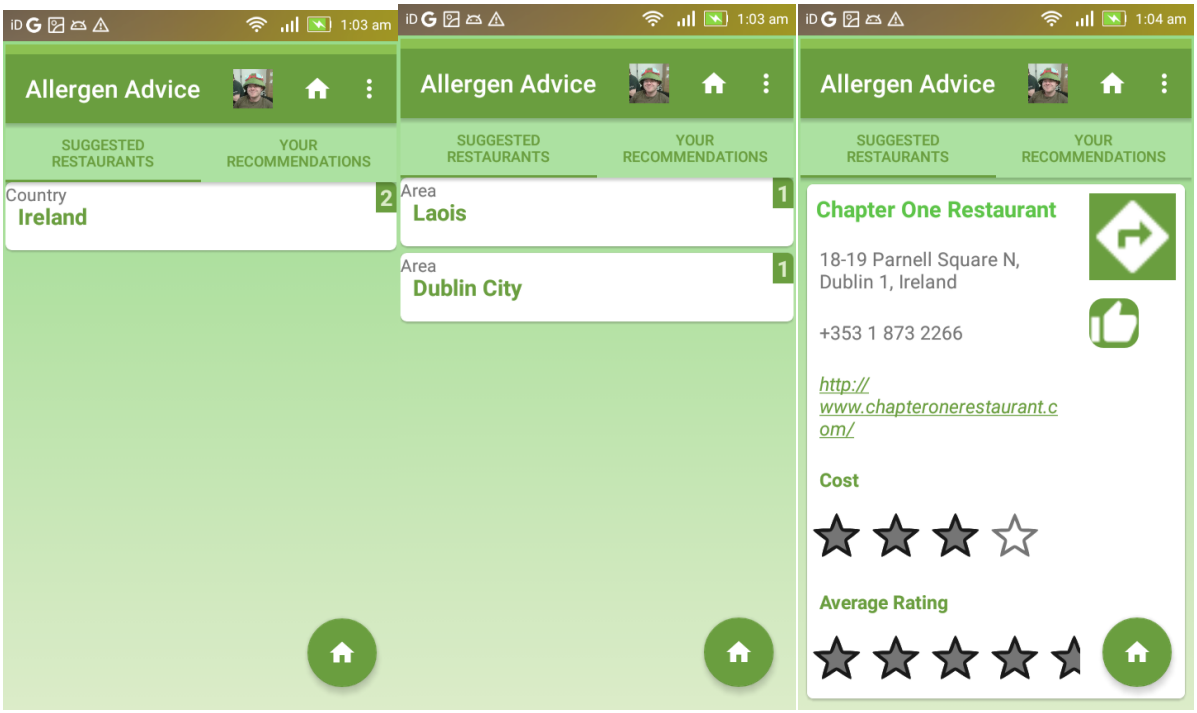
Input: User enters an eatery they suggest.

Action: The client opens a Google Place Picker widget. The Place object is then verified to be a bar, restaurant, cafe or bakery.

Output: The restaurant is stored along with the recommending user's allergen affliction information (form profile).

Priority: Critical. Based on midpoint recommendations this is a necessary feature.

FUNCTION: "RECOMMENDED RESTAURANTS"



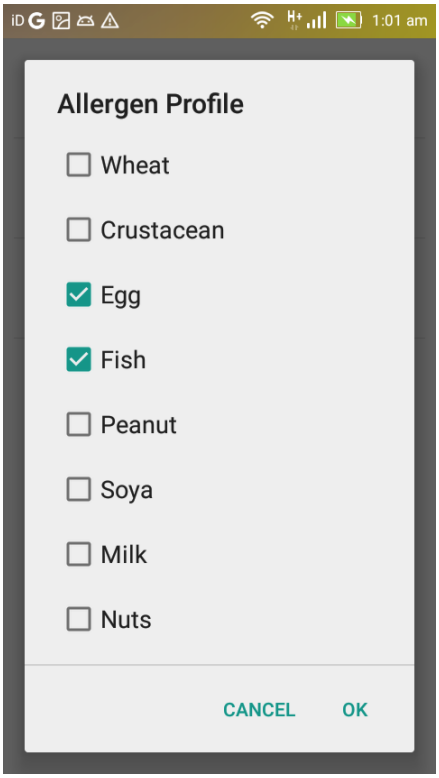
Input: User navigates through location map to find list of recommended restaurants in a locality.

Action: Selecting the Locality loads a list of cards.

Output: Restaurant cards display information about recommended restaurants including breakdown of relevant allergens of recommending users.

Priority: Very important. Ties in with recommendation service.

FUNCTION: "PROFILE INFORMATION"



Input: The user can navigate to the settings menu and specify the allergens relevant to them.

Action: The allergens are added to a profile stored on the phone.

Output: When a user opens phrasebook the allergens are ordered to prioritize the most relevant ones. When a user recommends a restaurant the data is stored in the cloud.

Priority: Nice feature. Not vital.

FUNCTIONAL REQUIREMENTS (MOBILE BACKEND)

FUNCTION: “STATUS/HEALTH CHECK ENDPOINT”

Input: Admin calls a server endpoint with “/status”

Action: The server should process the request.

Output: The server version and rest endpoints should be displayed in a HTML page in JSON format.

Priority: Very important. The most common issue with a server is a configuration file, not with a specific component of the service. A status endpoint will exist outside of all the other micro services in a system and should not be affected by an

individual component being down. But if the status endpoint does not work then there is a configuration error and the developer does not waste too much time checking individual features.

FUNCTION: “COUNTRY HEADER REQUEST LISTENER”

Input: Request containing “country” as a header parameter is received.

Action: An interceptor should pick up on this request and store the current country in context.

Priority: Not crucial but makes for better code that has less repetition.

FUNCTION: “PROCESS ALLERGEN ADVICE BASED ON COUNTRY HEADER”

Input: Server retrieves a request from the application with country parameter

Action: The server should request the entry for the specific country from the Dynamo repository.

Output: A JSON object is returned to the application with the requested information.

Priority: Critical importance. This is the base functionality of the application.

FUNCTION: “FOR PHRASEBOOK RETURN TRANSLATION SHEET FOR FOOD GROUPS IN SELECTED LANGUAGE”

Input: Server retrieves a request from the application with language parameter and requested “translateTo” parameter.

Action: The server should request the entry for the specific languages from the Dynamo repository.

Output: A JSON object is returned to the application with the requested information.

Priority: A very nice feature to have. Not top priority.

FUNCTION: “RESTAURANT STORE”

Input: Server stores and retrieves stored restaurant models.

Action: The client makes a request to the backend. The server reads from the database and creates a concurrent map.

Output: If the request is to get a restaurant info then the server returns the cached info. For post requests the cache is cleared and the map is remade.

Priority: Vital to the client’s restaurant service.

FUNCTION: “LOCATIONS SERVICE”

Input: Server navigates through list of restaurants in store.

Action: Server compiles a nested map of { Country, Greater Area, Locality } from every restaurant.

Output: Map is returned of all locations.

Priority: Vital to the client’s restaurant service.

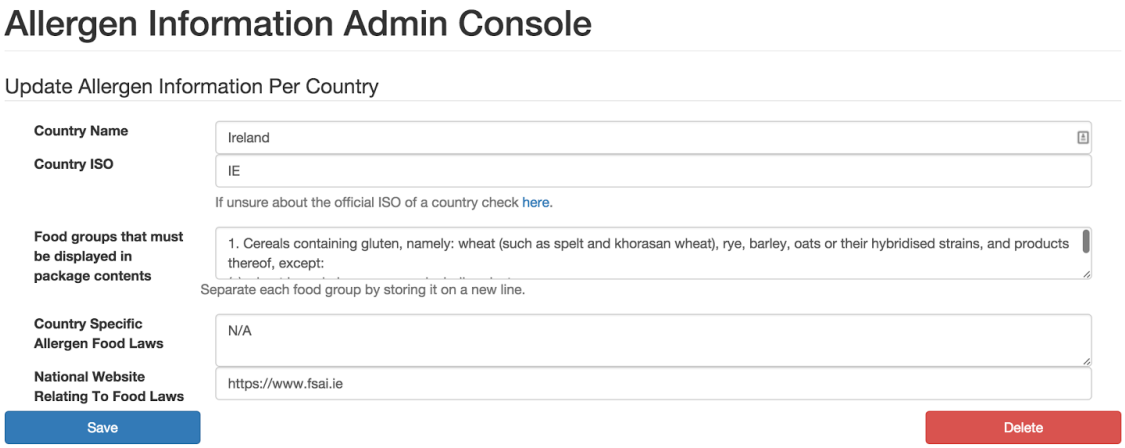
DATA REQUIREMENTS

In order to interact optimally with the mobile backend using Restful services, the Database should return objects in LinkedHashMap compatible form, because it is the class that closest matches a JSON object.

Due to the critical dependence of the stored data to the functionality of the application, backup and recovery options must be available.

To update tables within the database two simple web forms have been created:

One to update Allergen Information



and the other to update the Phrasebook.

USER REQUIREMENTS

In order to use the application, the user must be connected to the internet. I have considered using caching to allow for snapshots of the database to be saved on a phone but as of yet there is no plan to add this feature.

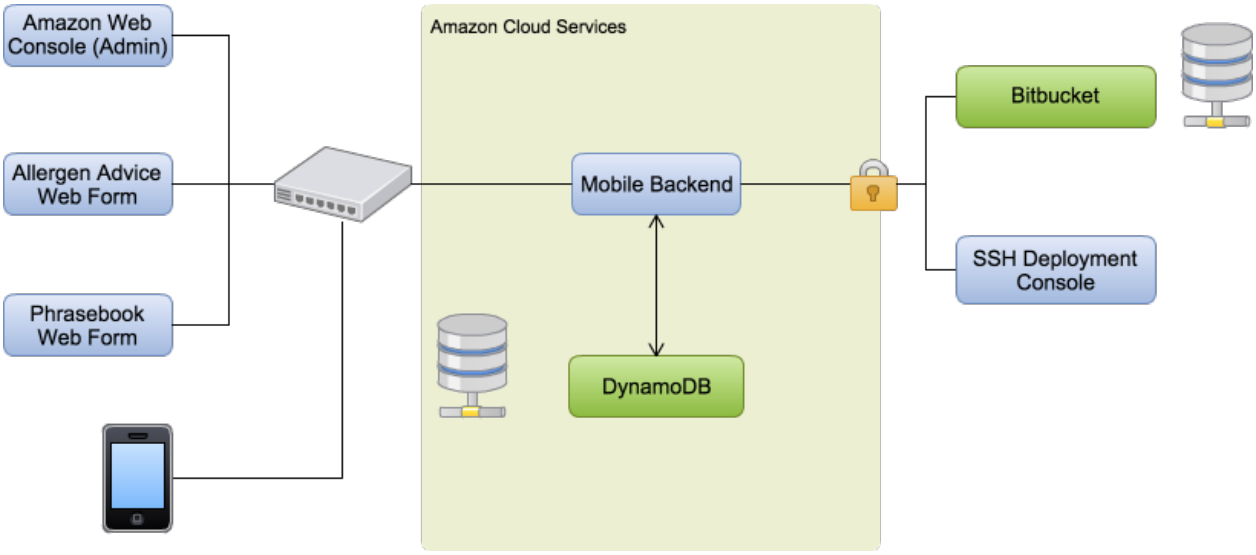
Initially the application had a minimum version of Android 4.0. In order to make full use of some nice new features as well as expanding functionality cleanly to a smartwatch I have opted to change the absolute minimum supported version to Android 4.4.

UX REQUIREMENTS

In order to make the user's experience the best one it can be the application will make use of a "light" and easy on processing power navigation bar. Where possible Fragments will be used in place of Activities in order to improve on more fluid view transitions.

This application is a simple idea and as such needs to be simple in its implementation. Every backend component is tested to assure it does what it was programmed to do but in order to better gauge usability requirements I will be depending on family and friends using a beta version of the application and giving their honest feedback. A developer is almost always too close to the implementation to have a clear idea of what issues users might run into.

Design and Architecture



The service will operate from an AWS instance. The mobile backend will communicate with some of the services on offer from Amazon, including an EC2 and DynamoDB instance. Spring Web Frameworks and Spring MVC will be used to handle dependency injection as well as the Restful dialogue between the backend and other components.

In order to access the Amazon, instance an ssh private and public key must be present on the machine, unless accessing port 80, which will be open to allow the application to access the backend over public ip.

The reason to use Amazon is because they are free for one year, if the decision is made to submit the application to the Play store there is an option for instant scalability, and they are doing all the heavy lifting in terms of devops tasks.

The main reason for a mobile backend is that scalability is greater. Phone performance and power is less relevant because the server takes care of

processing the requests. If the application ends up on the play store it is a lot less expensive to release hotfixes or minor releases on the backend than on the client, as the client requires users to update their device.

IMPLEMENTATION

In this section I have singled out, in my opinion, the most significant methods and classes in the mobile backend.

SERVER

COUNTRY HOLDER

The first is how the backend stores a header “country” in context once received from the application. The CountryHolderStrategy implements the ThreadInformationHolderStrategy interface in the utils package of the project. It appends or deletes, from thread context, the country string that is passed to it.

```
public class CountryHolderStrategy implements ThreadInformationHolderStrategy {
  private static final ThreadLocal<String> COUNTRY_HOLDER =
      new NamedThreadLocal<>("Country holder");

  @Override
  public String get() { return COUNTRY_HOLDER.get(); }

  @Override
  public void set(String country) { COUNTRY_HOLDER.set(country); }

  @Override
  public void reset() { COUNTRY_HOLDER.remove(); }
}
```

The CountryHolder is an enum that has one value (INSTANCE) which returns a thread safe instance of a single request. In Java terms this is a Singleton, a structure designed to restrict the instantiation of the class to a single object. This is a simple class designed to get set the context defined in CountryHolderStrategy.

```
public enum CountryHolder {
  INSTANCE;

  private ThreadInformationHolderStrategy strategy = new CountryHolderStrategy();

  public String getCountry() { return strategy.get(); }

  public Locale getLocale() {
    if (isCountrySet()) {
      return new Locale(getCountry());
    }
    return null;
  }

  public boolean isCountrySet() { return getCountry() != null; }

  public void setCountry(String country) { strategy.set(country); }

  public void setStrategy(CountryHolderStrategy strategy) { this.strategy = strategy; }

  public void reset() { strategy.reset(); }
}
```

The CountryHeaderListener implements the ServletRequestListener class the will listen for servlet requests being made and tidy/destroy them once the lifecycle of the request is done. Any Http request made that contains the header “country” will be saved to that thread and stored in context.

```
public class CountryHeaderListener implements ServletRequestListener {

  public static final String MTS_HEALTH_HEADER_COUNTRY = "country";

  @Override
  public void requestDestroyed(ServletRequestEvent requestEvent) { CountryHolder.INSTANCE.reset(); }

  @Override
  public void requestInitialized(ServletRequestEvent requestEvent) {
    String country =
        ((HttpServletRequest) requestEvent.getServletRequest()).getHeader(MTS_HEALTH_HEADER_COUNTRY);
    if (StringUtils.isNotBlank(country)) {
      CountryHolder.INSTANCE.setCountry(country);
    }
  }

}
```

When the backend processes a request the CountryHeaderRequestInterceptor checks from context whether or not country information has been set. If it has then the header information is appended to the request.

```
public class CountryHeaderRequestInterceptor implements ClientHttpRequestInterceptor {

  public static final String FORWARDING_COUNTRY_HEADER = "country";

  @Override
  public ClientHttpResponse intercept(HttpRequest request, byte[] body,
                                      ClientHttpRequestExecution execution) throws IOException {
    if (CountryHolder.INSTANCE.isCountrySet()) {
      request.getHeaders().add(FORWARDING_COUNTRY_HEADER, CountryHolder.INSTANCE.getCountry());
    }
    return execution.execute(request, body);
  }

}
```

PHRASEBOOKCONTROLLER

The PhrasebookController class retrieves, stores and manipulates the translations for the fourteen stored allergens. showTranslationsForGivenLanguages() retrieves a list of Phrases, pulled from the DynamoDB instance. The first time the server is called after a reboot it builds a cached map to save on response time. If any modifications are made to the data, the map is cleared and propagated again with the updated information.

```
@RequestMapping(method = RequestMethod.GET)
public ResponseEntity<Map<String, List<Phrases>>> showTranslationsForGivenLanguage() {
  if(!cachedPhrases.isEmpty()) {
    return new ResponseEntity<>(cachedPhrases, HttpStatus.OK);
  } else {
    ScanRequest scanRequest = new ScanRequest()
        .withTableName("PhraseBook");
    ScanResult result = client.scan(scanRequest);
    List<Phrases> listOfItems = new ArrayList<>();

    for (Map<String, AttributeValue> item : result.getItems()) {
      listOfItems.add(new Phrases().convertMapToPhrases(item));
    }

    Map<String, List<Phrases>> response = new HashMap<>();
    response.put("listOfItems", listOfItems);

    cachedPhrases.putAll(response);
    return new ResponseEntity<>(response, HttpStatus.OK);
  }
}
```

ALLERGEN ADVICE

The Allergen Advice information is the key piece of work being done by the application. The Allergen Advice model is a simple pojo with the exception of one method “convertItemToAllergenAdvice”. This method maps a DynamoDB Item to the AllergenAdvice model for marshalling the requests to the database.

```
public AllergenAdvice convertItemToAllergenAdvice(Item item) {
  AllergenAdvice allergenAdvice = new AllergenAdvice();
  allergenAdvice.setCountryISO(item.get("countryISO").toString());
  allergenAdvice.setCountryName(item.get("countryName").toString());
  allergenAdvice.setLinkToNationalAllergenHealthWebsite(item.get("linkToNationalAllergenHealthWebsite").toString());
  allergenAdvice.setSpecialInfo(item.get("specialInfo").toString());
  allergenAdvice.setListedHighRiskFoods(item.get("listedHighRiskFoods").toString());
  return allergenAdvice;
}
```

The AllergyAdviceController uses Annotations to inject the repository object as well as the Restful endpoints being hit.

```
@{...}
public class AllergyAdviceController {

  private static final Logger LOGGER = LoggerFactory.getLogger(AllergyAdviceController.class);
  private Table table;

  @Autowired
  public AllergyAdviceController(@Qualifier("prodAllergenInformationDynamoRepository") DynamoRepository repository) {...}

  @RequestMapping(method = RequestMethod.GET)
  public ResponseEntity<AllergenAdvice> getInformation() {...}

  @RequestMapping(value = "/form", method = RequestMethod.GET)
  public ModelAndView getForm(@Param(value = "country") final String country) throws JsonProcessingException {...}

  @RequestMapping(method = RequestMethod.DELETE)
  public ResponseEntity<String> delete(@Param(value = "id") String id) {...}

  @RequestMapping(method = RequestMethod.PUT)
  public ResponseEntity<String> save(@RequestBody AllergenAdvice request) {...}
}
```

The getForm method creates a ModelAndView that populates a jsp file with the requested advice. What this means is that a web form is generated that allows an admin to update allergen advice on the database.

```
@RequestMapping(value = "/form", method = RequestMethod.GET)
public ModelAndView getForm(@Param(value = "country") final String country) throws JsonProcessingException {
  Map<String, Object> itemMap;
  try {
    itemMap = table.getItem("countryISO", country).asMap();
  } catch (Exception e) {
    Item item = new Item()
        .withPrimaryKey(new PrimaryKey("countryISO", country))
        .with("countryName", "N/A")
        .with("listedHighRiskFoods", "N/A")
        .with("specialInfo", "N/A")
        .with("linkToNationalAllergenHealthWebsite", "N/A");

    table.putItem(item);
    itemMap = item.asMap();
  }
  final Map<String, Object> finalItemMap = itemMap;
  return new ModelAndView("/create-allergen-national-information-form",
      new HashMap<String, Map<String, Object>>() {{
    put("country", finalItemMap);
  }});
}
```

The custom RestController class uses the third party rest services library Retrofit. A model is mapped to a http request implemented elsewhere.

COMMUNITYCONTROLLER

Restaurant information is returned based on locality. If “all” is requested, then every restaurant is returned. The reason for “all” is because of a work in progress geo-fencing feature on the app.

```
@RequestMapping(method = RequestMethod.GET)
public ResponseEntity<List<RestaurantAPIModel>> getRestaurantsByRegion(@Param(value = "locality") String locality) {
  Map<String, AttributeValue> expressionAttributeValues =
      new HashMap<>();
  expressionAttributeValues.put(":locality", new AttributeValue(locality));

  ScanRequest scanRequest;

  if(locality.equals("all")) {
    scanRequest = new ScanRequest()
        .withTableName("Restaurants");
  } else {
    scanRequest = new ScanRequest()
        .withTableName("Restaurants")
        .withFilterExpression("locality = :locality")
        .withExpressionAttributeValues(expressionAttributeValues);
  }

  ScanResult result = client.scan(scanRequest);
  List<RestaurantAPIModel> response =
      result
          .getItems()
          .stream()
          .map(RestaurantAPIModel::convertMapToRestaurants)
          .collect(Collectors.toList());

  return new ResponseEntity<>(response, HttpStatus.OK);
}
```

populateCacheMap cycles through all the locations available and Restaurants, then assigns the locality required for the recommendation service used by the app.

```
private void populateCachedMap() {
  if (cachedCountryQueue.isEmpty()) {
    ScanRequest scanRequest = new ScanRequest()
        .withTableName("Locations");

    ScanResult result = client.scan(scanRequest);
    cachedCountryQueue.addAll(result
        .getItems()
        .stream()
        .map(Country::convertMapToCountries)
        .collect(Collectors.toList())
    );
  }
  for (Country country : cachedCountryQueue) {
    if(cachedResponses.get(country.getCountry()) != null) {
      if (cachedResponses.get(country.getCountry())
          .get(country.getGreaterArea()) !=null) {
        if(!cachedResponses.get(country.getCountry())
            .get(country.getGreaterArea())
            .contains(country.getLocality())) {
          Map<String, List<String>> copyOfGreaterArea = cachedResponses.get(country.getCountry());
          List<String> copyOfLocalities = copyOfGreaterArea.get(country.getGreaterArea());
          copyOfLocalities.add(country.getLocality());
          copyOfGreaterArea.put(country.getGreaterArea(), copyOfLocalities);
          cachedResponses.put(country.getCountry(), copyOfGreaterArea);

        }
      } else {
        ArrayList localities = new ArrayList();
        localities.add(country.getLocality());
        cachedResponses.get(country.getCountry())

            .put(country.getGreaterArea(), localities);
      }
    } else {
      Map<String, List<String>> greaterArea = new HashMap<>();
      ArrayList localities = new ArrayList();
      localities.add(country.getLocality());
      greaterArea.put(country.getGreaterArea(), localities);
      cachedResponses.put(country.getCountry(), greaterArea);
    }
  }
}
```

ANDROID

REST SERVICE

The Restful setup on the android application uses Retrofit to create the API, and OKHttpClient as the RestClient.

```
public interface AllergenAdviceApi {

    String BASE_URL = "/aa";
    String RESTAURANT_PATH = "/restaurant";
    String LOCATIONS_PATH = "/locations";
    String ALLERGEN_ADVICE_PATH = "/allergen_advice";
    String PHRASEBOOK_PATH = "/phrasebook";
    String LOCATION_PARAM = "locality";

    @Headers("Cache-Control: max-age=14400")
    @GET(BASE_URL + ALLERGEN_ADVICE_PATH)
    Call<AllergenInformationAPIModel> getAllergenInformation(@Header("country") String country);

    @Headers("Cache-Control: max-age=14400")
    @GET(BASE_URL + PHRASEBOOK_PATH)
    Call<PhrasebookListAPIModel> getTranslations(@Header("language") String language);

    @Headers("Cache-Control: max-age=1440")
    @POST(BASE_URL + RESTAURANT_PATH)
    Call<ResponseBody> addRestaurant(@Body RestaurantAPIModel restaurantAPIModel);

    @Headers("Cache-Control: max-age=1440")
    @GET(BASE_URL + RESTAURANT_PATH + LOCATIONS_PATH)
    Call<Map<String, Map<String, List<String>>>> getLocations();

    @Headers("Cache-Control: max-age=1440")
    @GET(BASE_URL + RESTAURANT_PATH)
    Call<List<RestaurantAPIModel>> getRestaurantsInLocation(@Query("locality") String location);
}
```

The CommonRestConfiguration creates a static RestController, “service”, that can be called by an Activity or Fragment within the application. The JacksonConverterFactory class is used to map JSON formats.

```
public class CommonRestConfiguration {

    public static RestController service() {
        Retrofit restAdapter = new Retrofit.Builder()
                .baseUrl("http://ec2-52-33-187-222.us-west-2.compute.amazonaws.com")
                .addConverterFactory(JacksonConverterFactory.create())
                .build();
        return restAdapter.create(RestController.class);
    }

}
```

PHRASEBOOK

The PhraseBookFragment class implements the service method to instantiate one of the interface methods described in the custom RestController. onCreateView creates a new Rest based request for service().getTranslations(currentLocale, requestedTranslation). What this does is send a request to the app that looks like this:

URL:

http://ec2-52-33-187-222.us-west-2.compute.amazonaws.com/aa/phrasebook?translateTo=${currentLocale}

Header:

“language” = “${requestedTranslation}”

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    view = inflater.inflate(R.layout.fragment_phrasebook, container, false);

    Call<Phrasebook> restControllerCall = service().getTranslations(currentLocale, requestedTranslation);
    restControllerCall.enqueue(new Callback<Phrasebook>() {

        @Override
        public void onResponse(Response<Phrasebook> response, Retrofit retrofit) {
            populatePhrases(response.body());
        }

        @Override
        public void onFailure(Throwable t) { System.out.print("Failed at " + t); }
    });
    view.setTag("activeTag");
    return view;
}
```

RESTAURANT RECOMMENDATION

Google’s AutoComplete Place Picker is used to add restaurants to the Restaurant store.

```
if (containsEatery) {
    List<String> allergens = new ArrayList<>();
    for (String allergen : prefs.getStringSet(Constants.ALLERGENS, new HashSet<>())) {
        allergens.add(Allergens.values()[Integer.parseInt(allergen)].toString());
    }

    RestaurantAPIModel restaurant = new RestaurantAPIModel();
    restaurant.setId(place.getId());
    restaurant.setEatery(place.getName().toString());
    restaurant.setAddress(place.getAddress().toString());
    restaurant.setPriceLevel(place.getPriceLevel());
    restaurant.setRating(place.getRating());
    restaurant.setTelephoneNumber(place.getPhoneNumber().toString());
    restaurant.setWebsite(place.getWebsiteUri().toString());
    restaurant.setLatitude(place.getLatLng().latitude);
    restaurant.setLongitude(place.getLatLng().longitude);
    restaurant.setAllergensOfRecommendingUser(allergens);

    Call<ResponseBody> restControllerCall = RestClient.getApi(this).addRestaurant(restaurant);
    restControllerCall.enqueue(new Callback<ResponseBody>() {
        @Override
        public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
            Snackbar.make(view, "Eatery successfully added to service. Thank you!!", Snackbar.LENGTH_LONG).show();
            Intent restartCommunity = new Intent(RestaurantSelectorActivity.this, CommunityActivity.class);
            startActivity(restartCommunity);
        }

        @Override
        public void onFailure(Call<ResponseBody> call, Throwable t) {
            System.out.println("Throwable:" + t);
            Snackbar.make(view, "There was an error. Please try adding the eatery again.", Snackbar.LENGTH_LONG).show();
        }
    });
} else {
    Snackbar.make(view, place.getName() + " is not a recognized eatery.", Snackbar.LENGTH_LONG).show();
}
```

The request validates that the returned data is an eatery. It then goes on to populate the card.

TESTING

I decided to design the mobile backend from a test driven approach, meaning I wrote the tests first and then developed the “how to” after. In order for Test Driven Development to be used as a design tool, it is very important to use a human readable naming structure. Below is an example from the AllergenAdvice model test. This unit test is asserting that the conversion from DynamoDb Item to AllergenAdvice model mapping is correct. The name of the test class is AllergenAdviceShould and any additional methods should be named after the specific action. So a class + test method should read ClassNameShould -> do_this_action

```
public class AllergenAdviceShould {

  private AllergenAdvice allergenAdvice;
  private Item allergenAdviceItem;

  @Before
  public void setup() {...}

  @Test
  public void
  convert_item_to_allergen_advice() {...}
}
```

The setup method creates a new AllergenAdvice object and a DynamoDb Item, populating them with the same data.

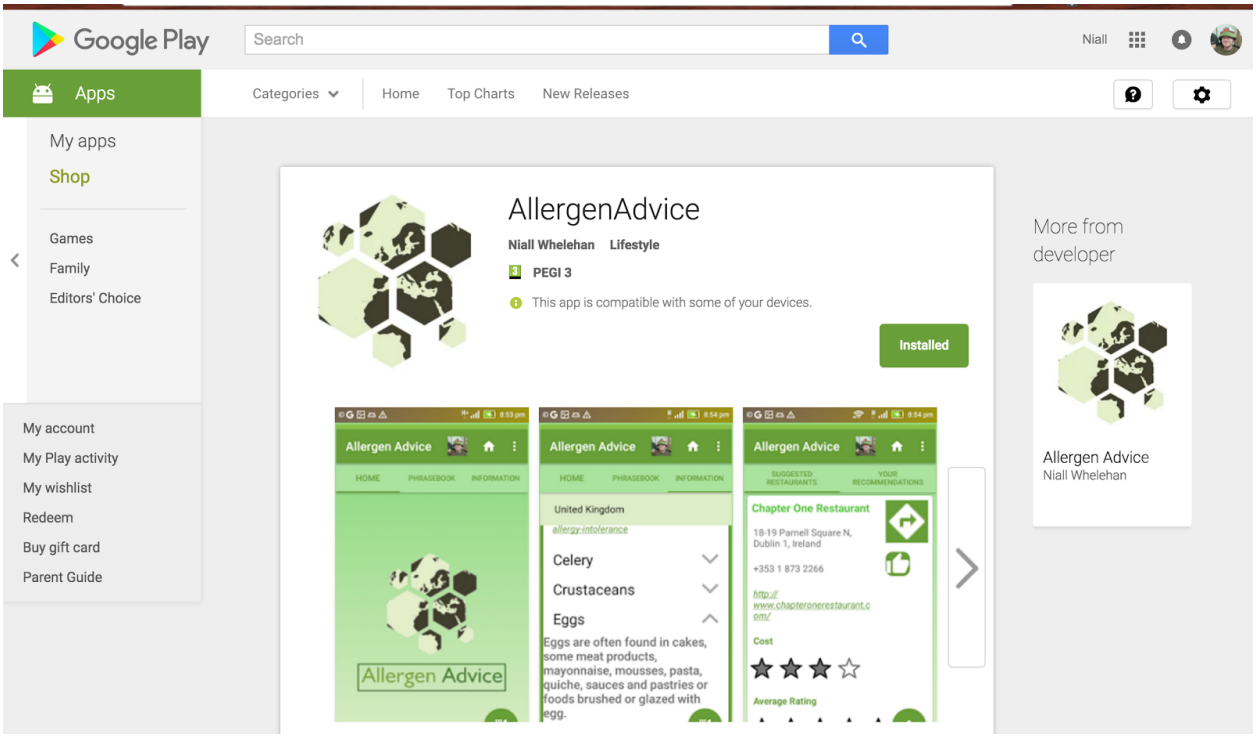
```
public class AllergenAdviceShould {

  private AllergenAdvice allergenAdvice;
  private Item allergenAdviceItem;

  @Before
  public void setup() {
    allergenAdvice = new AllergenAdvice();
    allergenAdvice.setCountryISO("IE");
    allergenAdvice.setCountryName("Ireland");
    allergenAdvice.setLinkToNationalAllergenHealthWebsite("www.fsai.com");
    allergenAdvice.setListedHighRiskFoods("Fish, Peanuts");
    allergenAdvice.setSpecialInfo("blah, blah");

    allergenAdviceItem = new Item()
        .with("countryISO", "IE")
        .with("countryName", "Ireland")
        .with("linkToNationalAllergenHealthWebsite", "www.fsai.com")
        .with("listedHighRiskFoods", "Fish, Peanuts")
        .with("specialInfo", "blah, blah");
  }
```

The test method, convert_item_to_allergen_advice, asserts that when a new AllergenAdvice object is instantiated using the DynamoDb Item created in setup, then it should match the AllergenAdvice object also created in setup.

```
  @Test
  public void
  convert_item_to_allergen_advice() {
    AllergenAdvice allergenAdviceFromItemConversion =
        new AllergenAdvice().convertItemToAllergenAdvice(allergenAdviceItem);
    assertThat(allergenAdviceFromItemConversion.getCountryISO(),
        equalTo(allergenAdvice.getCountryISO()));
    assertThat(allergenAdviceFromItemConversion.getCountryName(),
        equalTo(allergenAdvice.getCountryName()));
    assertThat(allergenAdviceFromItemConversion.getLinkToNationalAllergenHealthWebsite(),
        equalTo(allergenAdvice.getLinkToNationalAllergenHealthWebsite()));
    assertThat(allergenAdviceFromItemConversion.getListedHighRiskFoods(),
        equalTo(allergenAdvice.getListedHighRiskFoods()));
    assertThat(allergenAdviceFromItemConversion.getSpecialInfo(),
        equalTo(allergenAdvice.getSpecialInfo()));
  }
}
```

For the mobile application I have used smoke tests only. My knowledge and familiarity of Android's UI means that my tests did not suit the purpose. Instead I manually tested the phone.

I opened the application up to a focus group in the Google Play Store.



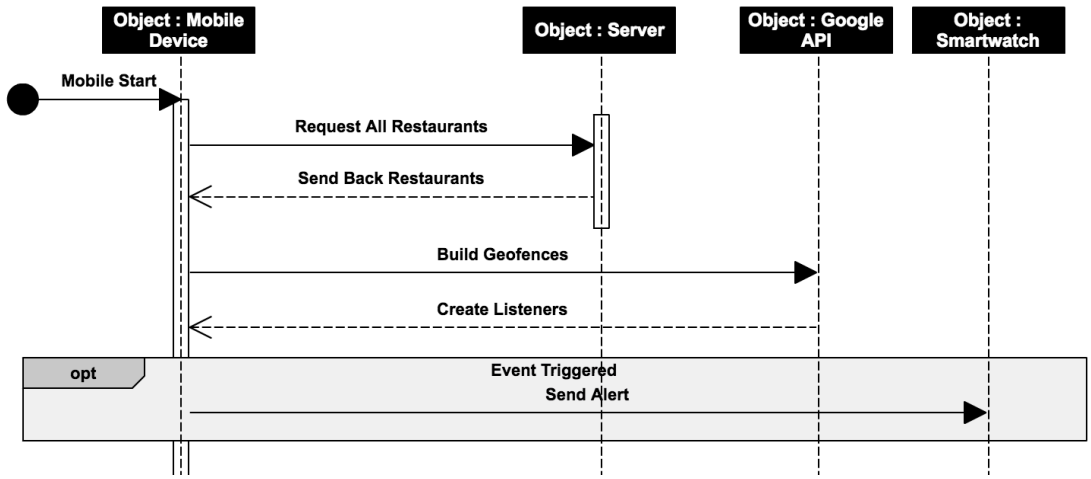
Members of the group could download the latest builds. In the Google+ Community that was setup along with the Beta I included a feedback and bug report tracker where friends and focus group could leave their thoughts and point out what was broken.

I found the focus group the most useful part of this. A lot of the manual tests I was carrying out worked like a charm, but only because I had an idea of what I wanted to do each time. It was amazing to see how many critical bugs were found after the first User test period. I have included the Spreadsheets in the Appendix.

WORK CURRENTLY IN PROGRESS

GEOFENCE ALERTS

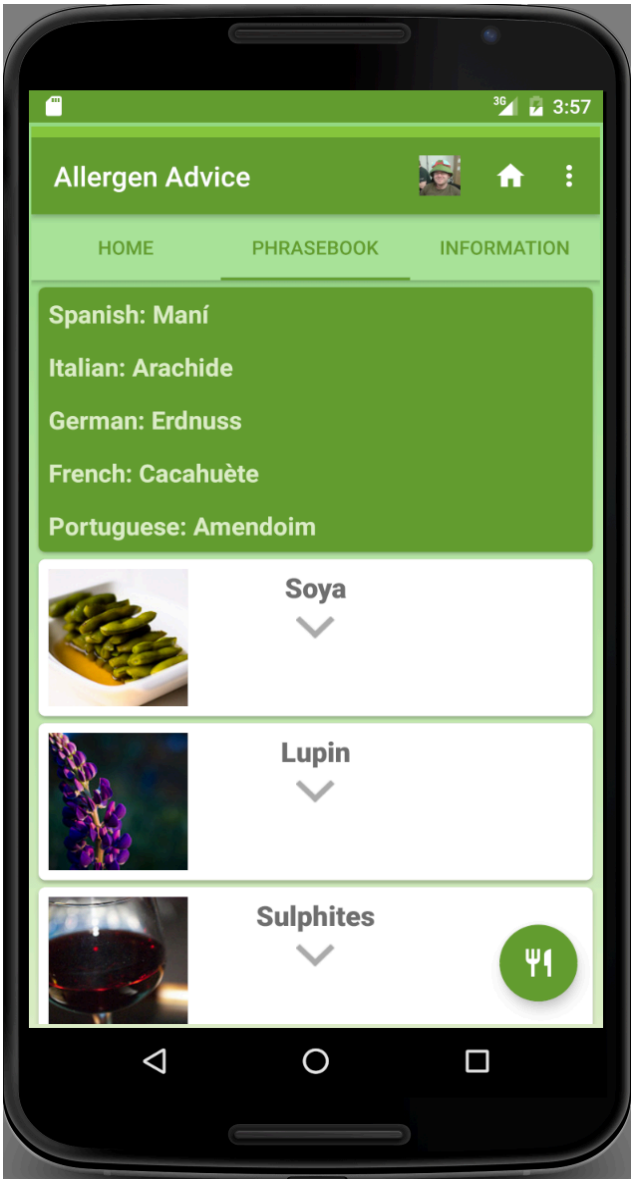
Implementing the Geofencing alerts in the application to notify the user when they are close by a recommended restaurant. If the user has a smart watch synced with their mobile the alert should be sent there.



USER FEEDBACK

As mentioned earlier, the best tests I had were the focus group I opened the application to. If the app were released as a Production deployment then I would like to have users be able to have their voice, in an effort to better the app and the user experience.

GRAPHICAL USER INTERFACE (GUI) LAYOUT



The app makes use of Material Design to maintain a sense of flow. There are no major clashes in the style and layout because there is one central color palette being used.

Navigation is handled using tabs and a FloatingActionButton to go from the Dashboard to the Community (Restaurant) area of the app.

In order to keep the size of the app down the Phrase images are added dynamically from the server.

All of the icons used in Allergen Advice are Creative Commons and freely available from https://design.google.com/icons/.

The Phrasebook images were taken from Creative Commons searches on Flickr. Each image did not require any credits to be given so I have left them out.

CONCLUSIONS

WATERFALL VS AGILE

In terms of planning and forecasts one thing is certain when it comes to mobile applications… the market moves too fast for Waterfall. Having hard deadlines and dates set months in advance is not going to work when there are so many factors and changing trends. I tried to apply an Agile approach to this project whereby I had a roadmap of things I'd like to get done on this project as well as a rough idea of what order and when. However, I worked in sprints of two weeks, whereby at the end of a two-week period I reflected on the work done compared to what I planned. As my knowledge of some of the technologies improved my vision of the project changed. Not having hard targets allows for these kind of changes and I think the application will be much better because of this.

ACTIVITIES VS FRAGMENTS

This is the first time that I invested so much into an Android application. So up until now I was only ever aware of Activities. When you open an Android application an Activity is launched and that is what you see. On top of that Activity can be multiple Fragments similar to the idea of a placeholder in Ruby and .NET. Activities are heavier on processing so should only be used when the view of the application should completely change. If, however, a button like Phrasebook were pressed that only needed to display some translations then it makes more sense to load that within the currently running Activity.

AMAZON SERVICES ARE YOUR FRIEND

After using other cloud providers in the past I was more than impressed with Amazon. DynamoDB is a fine solution for a hosted NoSQL database, although not as user friendly or support ready as MongoDB. The developers console is easy to use. Unless an app is incredibly successful, the cost of using the Amazon services is very fair.

ACRONYMS AND PHRASES

According to Google Dictionary - "A damaging immune response by the body to a substance, especially a particular food, pollen, fur, or dust, to which it has become hypersensitive."

App/Application/Mobile Application/Client/Front End - The Allergy Advice Android application

Backend/Mobile Backend - The Java instance communicating between the application and the database.

Play Store - Official download resource for Android Applications

Cloud Solution - A solution that uses hardware or software that is not physically attached or installed to your device and can be accessed via the web.

Philanthropic - A venture that is not motivated by monetary gains.

Rest Services - Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. (Docs.oracle.com)

Spring Frameworks - A popular solution for using Dependency Injection and Aspect Orientated Programming

War File - A Web Application Archive is a file that wrap all java classes, servlets, web formats and properties together for ready distribution to a server.

Hotfix - An unplanned release of an application to address a critical bug in the system

GPS = Global Positioning System used to get coordinates on a global scope.

Thread (Java) = In Java a thread is a flow that is independent of other current interactions within the program.

REFERENCES

Keane, John. Nut Allergy. 2009. Web. 28 Jan. 2016.
Docs.oracle.com,. "What Are Restful Web Services? - The Java EE 6 Tutorial". N.p., 2013. Web. 24 Jan. 2016.

APPENDICES

APPENDIX (I & II) - PROJECT PROPOSAL & PROJECT PLAN

Project Proposal & Project Plan

Niall Whelehan

x12102954

niall.whelehan@student.ncirl.ie

BSc (Hons) in Computing (Evening)

Allergy Ambassador

Travel Companion and Advisor for

persons affected by food allergies

https://mrgnx3@bitbucket.org/mrgnx3/allergenadviceandroid.git

https://mrgnx3@bitbucket.org/mrgnx3/allergenadvicebackend.git

Background

Food allergies are becoming more and more prevalent in modern society. The shock of the first major attack is enough to make most people research the information relevant to their condition. One thing that might not be taken into consideration is that the laws that govern food allergies and general health are not universally set. In the EU there are 14 food types that must be displayed in bold

if they are present in the product whereas in the USA there are only 8 compulsory food items that must be stated.

This application will act as an advisor and educator. The target audience is parents of children affected by food allergies who intend on travelling, backpackers who find themselves in exotic locations and allergy sufferers with lifestyles that require a lot of travel.

Technical Details and Approach

The three major components in this project will be:

- A java backend using a number of Spring framework's, a groovy + cucumber based test suite and some jsp + spring web frameworks for any wrapped external pages that require data input.
- An Android front end. Having less experience with front end development I intend to use some time ramping up on Android standards and suggested implementations.
- Research. I do not intend or expect to research every country's laws and policies regarding allergens in a single year, but I do intend to create a transferable model that can easily be updated once more information is sourced.

Using Spring contexts, I will create profiles for development, mock and production. My knowledge of Spring is reasonable but I will also take some time to improve my current understanding.

The main body of code will be written in Java while the tests will be written in Groovy (due to its nicer integration with the Cucumber-jvm and the build tool gradle).

The server should have taken care of most of the work to the point that the App should just need to call the correct endpoint and the API will return the information it needs. The major requirements of the application will be establishing the connection and, if I decide to implement it, include a serializer that maps gps coordinates to a specific country iso.

The information will be displayed using a template that will be constant between each country. A default placeholder will be set explaining that information is not yet available for selected country. Once research has been carried out on said country the information should be stored in one place, be easily accessible and editable.

Resources required

In order to successfully carry this project out I will need:

- A better understanding of Android development.
- A better understanding of Spring Frameworks.
- A Database, most likely mongoDB.
- The correct IDE. Intellij for Backend, Android Studios for Client.
- Information and a maintainable resource to store information.
- A professional and attractive presentation tool for showcasing the process. Eg, Cacoo.
- Time and advice.
- An Amazon account + AWS boxes to host the backend.
- Wunderlist will be used to keep track of work left to do.
- Github will be used to track work done.

Evaluation

I intend on designing a test suite in the mobile backend to ensure that any changes made to the service will not break the current implementation.

For unit tests and integration tests a library such as Hamcrest should suffice.

For functionality testing I will use Groovy, Cucumber (or some other Gherkin language) and gradle build scripts. If mocks are required I will include Wiremock.

For the front end development, I might include some unit tests but, due to the main workload being done on the backend, most of the testing on Android will be smoke testing and UAT(User Acceptance Testing) sessions with friends and family.

Project Plan - Milestones and Timeline

Note: I intend to implement an agile approach to this development. “Sprints” are bodies of work carried out over 2 weeks. A Sprint can be planned in advance, and can change as late as the day before, but once the Sprint starts there is an agreement that only the planned work is carried out. This allows for better focus, especially in larger projects. Any mentioned deadlines are purely based on a high level estimate that is likely to change from one Sprint to another. At the end of every Sprint there will be an updated planning and a revision of work done/required. This is known as Sprint Planning and Sprint Retrospective.

Week 2 - 21/09

Planning: Project Proposal approved.

Design: Initial wireframes and feature planning

Week 3 - 28/09

Server: Sign up for amazon account and setup an aws instance to work off of. Set up github repo for project. Create Spring/tomcat Intellij Project and configure required contexts

Sprint 1 - 05/10

Story: Create Dashboard (Mocks + Android)

Task: Research at least 2 different countries per Sprint

Sprint 2 - 19/10

Story (Continued)... Create Dashboard (Mocks + Android)

Task: Document API and begin planning server work for Dashboard

Task: Research at least 2 different countries per Sprint

Sprint 3 - 02/11

Story: Implement initial API calls for Dashboard interaction (Backend)

Task: Research at least 2 different countries per Sprint

Sprint 4 - 16/11

Story (Continued)... Implement initial API calls for Dashboard interaction (Backend)

Task: Research at least 2 different countries per Sprint

Sprint 5 - 30/11

Story: Allergy information Template (Android)

Technical Task: Investigate gps coordinates to country iso serialization

Task: Research at least 2 different countries per Sprint

Sprint 6 - 14/12

Story: Allergy Information Template (Backend)

Task: Research at least 2 different countries per Sprint

Sprint 7 - 28/12

Xmas Break/ Exam Study

Task: Research at least 2 different countries per Sprint

Sprint 8 - 11/01

Technical Task: Investigate cms solution for country information.

Story: Implement allergen translator(Android)

Task: Research at least 2 different countries per Sprint

Sprint 9 - 25/01

Task: Research at least 2 different countries per Sprint

Story: Implement allergen translator(Backend)

Sprint 10 - 08/02

Task: Research at least 2 different countries per Sprint

Story Continued... Implement allergen translator(Backend)

Sprint 11 - 22/02

Task: Research at least 2 different countries per Sprint

Story: Add alert when entering a new country(Android)

Sprint 12 - 07/03

Task: Research at least 2 different countries per Sprint

Story Continued... Add alert when entering a new country(Android)

Sprint 13 - 21/03

Task: Research at least 2 different countries per Sprint

Story: Add alert when entering a new country(Server)

Sprint 14 - 04/04

Task: Research at least 2 different countries per Sprint

Story Continued... Add alert when entering a new country(Server)

Sprint 15 - 18/04

Buffered period for inevitable delays due to bugs/ unforeseen circumstances

Sprint 16 - 02/05

Buffered period for inevitable delays due to bugs/ unforeseen circumstances.

Objectives

Promised Features

- A template, that can be used to populate at least a country's information about allergen food law and policies.
- At least 2 dozen countries populated in the database using the above template.
- A context controller to compare the differences between the user's country of origin and current/selected locale.
- A dictionary of known allergens in a minimum of 3 languages.

Desired Features

- A personal profile where users can keep a note of potential food sources that triggered a reaction in themselves.

- A Content Management Service to store all translations and information.
- A controller that manages gps to iso conversion, recognizing the country the user is currently in.

Summary

The Allergy Ambassador (working title) is an ambitious project due to the amount of work required. Every sprint will need to be managed in a way that allows time for both development, planning, design and research. By using an Agile approach, the task will be less rigid and allow for changes to the roadmap where necessary. The intention of the app is to spread awareness of differences in food laws and policies between countries. There is no intention of making the app a profitable campaign.

APPENDIX(III) PROJECT JOURNALS

Software Journal September

Niall Whelehan

x12102954

niall.whelehan@student.ncirl.ie

BSc (Hons) in Computing (Evening)

Allergy Ambassador

First journal for the ongoing project

Month One (Jan 12th - October 5th)

As the heading states, the importance of this project to the overall mark has not been lost on me. I created a list at the start of the year of potential final year project ideas, adding to and removing from the list as the year went on.

The first phase in this process was to get as many ideas (good or bad) on paper. There is a new philosophy on Software Development called the Hammock approach whereby the developer thinks of a solution to an idea and then sits on that idea for a couple of days. With any luck there is a eureka moment, a better implementation for the solution is thought up and the developer has saved themselves and their team some unnecessary pain in refactoring a piece of work that was rushed into. Once I had enough ideas on paper I looked into phase two.

The second phase of the pre-project was to start considering how feasible the idea would be for a single developer, with part time availability, to implement the project inside of 7 months. Were there too many dependencies on third parties? Was there a hardware constraint that might involve unavailable funds? Will the final result be easy to pitch to someone hearing about it for the first time? Using these criteria and more I slowly dwindled the list down to eight, and then six, and then four, and then two …

The third phase was the beginning of the semester. Asking Eamon to fill in any gaps in my understanding of what was required of the project. Knowing that I would not get the all clear to proceed with my idea straight away I began the first couple of weeks with generic admin tasks that would universally fit no matter what project I ended up with, e.g., setting up an Amazon instance, prepping the tools that I would require, installing software needed, beginning Android Tutorials.

In the end I chose the Allergy Ambassador idea, which at the time of writing this has yet to be given the go ahead, for a couple of reasons.

- I wanted to experience working with native front end development. I am not interested in doing Android in a Java way, I'd prefer to pick up the standards required of an Android Developer to get a better view of how this side works.
- I felt that I could achieve the plan that I had laid out in the time available.
- My son has a peanut allergy that is as serious as they get. The thought of travelling to a country and being unprepared for their customs and laws is a terrifying thought that I am sure I share with a lot of parents.

Setting up the basics

The first task I undertook was planning how I should proceed and deciding on what was the main deliverables that needed to be in the application. As well as that what were the dependencies of those deliverables. Once I had that decided I started with the admin stuff.

My plan from the start was always going to involve taking advantage of the great offers with Amazon cloud servers. It takes an entire field out of this project in Dev Ops, a discipline I am interested in but felt would be one too many new things to explore in a project. Instead I will now how an AWS instance and a Github web hook that will do all the heavy lifting, in terms of deployment, for me.

The next process was to make some basic wireframes to think about. This is a simple throw away process that means I have not invested too much effort into anything more should the application idea get declined.

Putting the tutorials to the test

After going through two of the introductory tutorials in Pluralsight for Android development I decided to implement the basic dashboard that I had created in wireframe form previously. If the project proposal is accepted then great, I have a good start on the development, if not I have gotten a good practice session on Android development.

The implementation and general layout of an Android project is a bit different that anything I've worked on before. It is similar to some of the Model View Controller developments I have worked on in the past but the view is very different. The resources are extensive in terms of having global xml files and reference ids for every property defined in the application. I work with property files everyday but the volume of properties that get stored in these shared files is a lot more extensive than I have found on server side development.

Agile Development and Planning

I intend to use this monthly journal to reflect on the most recent Sprint (2-week body of planned work). Obviously for the first report I have not got a retrospective to mention so I shall skip the first part and go straight to the planning for the next 2 weeks. After two weeks I'll update the first half of the next journal with Sprint 1 Retrospective, and Sprint 2 Planning.

Sprint Planning

The goals of the next sprint are:

- To continue the development of the basic Dashboard on Android.
- To continue the tutorial series on Pluralsight.
- Design a template to work with for highlighting basic laws and governance around allergen laws.

Some tasks I intend to carry out and a rough estimate as to how long I expect each task to take.

Estimated available time: 15 hours (7.5 hrs. per week)

Story: Develop Basic Dashboard on Android

Acceptance Criteria

Given: I navigate to the home Dashboard

As a user: I want to be able to select my country of origin

Then: The background image should change

And: The buttons on screen should show information in context

- Task: Investigate best approach to implement dynamic contexts on dashboard (3 hours)
- Task: Implement dynamic context logic in Dashboard (6 hours)

Story: Display Allergen Laws Information

Acceptance Criteria

Given: I navigate to the home Dashboard

When: I press the Allergen Laws button

Then: The information of the country set in Context is displayed

- Task: Design basic form for adding and updating allergen information (2 hours)
- Task: Investigate and document 2 countries/regions information around allergen laws (2 hours)

Ramp Up: Continue Android Tutorial Series (2+ hours)

Agile terms and techniques

- Agile: A conceptual framework made up of many sub components intended to drive reactive software projects.

- Sprint: A two-week period of time devoted to a predetermined set of work goals.
- Story: A description of a feature from the perspective of the end-user.
- Acceptance Criteria: The functional requirements of a story.
- Ramp Up: Period of learning set aside to learn a skill that a story has a dependency on.
- Sprint Planning: Review of hours available and deciding the work involved in the next sprint.
- Sprint Retrospective: Reviewing the current sprint on its last day to reflect on good and bad decisions made and how to improve.

Software Journal October

Niall Whelehan

x12102954

niall.whelehan@student.ncirl.ie

BSc (Hons) in Computing (Evening)

Allergy Ambassador

Grinding Through the Early Pains

Month Two (Oct 6th - Nov 3rd)

I guess the best way to follow on from the last report is to give an update on how the previous goals went.

To continue the development of the basic Dashboard on Android.

I have finished this to the point that I was happy to leave it and begin developing the backend in the sprint that followed.

To continue the tutorial series on Pluralsight.

I have done 2 four hour tutorials on Android from Pluralsight as well as an introductory free online course recommended from a colleague. I do not intend to do any further tutorials until I progress with the backend development and return to Android.

Design a template to work with for highlighting basic laws and governance around allergen laws.

I have created a mongo entry that covers

- The two letter ISO describing the country, e.g. “IE” for Ireland
- The country name
- A list of strings stating the mandatory foods that are required to be specified on food packaging for said country.
- A link to the country’s official food authority

Retrospective

I found the hardest side of working on Android is breaking the link between it and Java because the correct implementation is very different. Also it is even more important than any other language I’ve used to decide on a naming convention early, because when you have variables named “DASHBOARD_TEXT” that actually points to the Textbox object and not the text value you very soon run into problems.

Going forward I think I will need to be looser with my estimates. When I planned how long a task would take I did it with the same mentality I use when planning in work. The major difference between working on a 15-hour piece of code in work is that I will probably get a good 4-5 hours in one stretch.

From Front to Back and Front Again

Once I was satisfied with the basic layout of the Android Dashboard I began work on the mobile backend that will support the app. The intention is to do any “heavy lifting” there so it is very important to have a good API design to work with. As a result, I kept going between Client and Server while thinking about what I wanted and needed from the api as well as the dialogue that would pass between phone and server.

Test Driven Development

I was very lucky to have gotten the chance to attend JAX London this month, a Java conference hosting a lot of exceptional thinkers and speakers. One such speaker was Sandro Mancuso who is an advocate of TDD, or Test Driven Development. I left the conference wanting to test out some of his teachings and thought this project is the best chance I will get.

I decided that I would first tackle the main functionality, which is to retrieve Allergen Laws for the country selected. So when I began to form the design I began at the highest layer of abstraction, the AllergenAdviceController. From there I created a test using JUnit and Mockito that slowly built into what I have now implemented.

So using TDD and only knowing that I wanted to make a controller to call Allergen information I now have an interceptor that stores a context based on what country the phone has requested information from, a mongo repository class, a set of tests and all the required Spring configurations needed to implement this. Instead of trial and error development I feel like I have saved myself an extra 2 weeks using this method.

Agile Development and Planning

Sprint Planning

The goals of the next sprint are:

- To return to Android and design correct dialogue between the allergen advice controller implemented in backend.
- Continue studying android

Some tasks I intend to carry out and a rough estimate as to how long I expect each task to take.

Estimated available time: 15 hours (7.5 hrs. per week)

Story: Develop Allergen Advice functionality on Android

Acceptance Criteria

Given: My country context is set

As a user: When I press the Allergen Advice button

Then: I retrieve information about the country set in context

- Task: Create Allergen Advice implementation (12 hours)
- Task: Research unit testing for Android (3 hours)

Agile terms and techniques

- Agile: A conceptual framework made up of many sub components intended to drive reactive software projects.
- Sprint: A two-week period of time devoted to a predetermined set of work goals.
- Story: A description of a feature from the perspective of the end-user.
- Acceptance Criteria: The functional requirements of a story.
- Ramp Up: Period of learning set aside to learn a skill that a story has a dependency on.
- Sprint Planning: Review of hours available and deciding the work involved in the next sprint.
- Sprint Retrospective: Reviewing the current sprint on its last day to reflect on good and bad decisions made and how to improve.

Software Journal November

Niall Whelehan

x12102954

niall.whelehan@student.ncirl.ie

BSc (Hons) in Computing (Evening)

Allergy Ambassador

Backend Looking Good

Month Three (Nov 4th - Dec 1st)

I started this month by slightly off roading to investigate how to automate my deploy scripts. The Amazon EC2 box that I am using is up and running (with relatively few bumps and bruises). The problem I ran into was the physical deployment. Being blessed with having professional operations engineers in my work life, I completely forgot that a war does not build and deploy itself to a live instance. So with that realization I replanned my time to include a spike in that area to get a better understanding of the work involved.

Once I got passed that I managed to finish off work on the backend server and resume work on the Android App. I spent more time studying than implementing but feel like it was time well spent. My understanding of fragments vs actions and the different type of layers has improved enormously. The functionality of the app has not caught up to the backend like I had planned but it is clear to see why. I now know how to properly display my layouts and even better I know how to deploy my projects to run on a cloud instance.

Backend Is Working, Tests Up and Running

The backend can now be called from my local machine with a status endpoint and allergen advice response. The integration with mongodb is tested, the country context header is tested, the status response is tested and the Allergen Advice Controller itself is tested.

Agile Wizardry

I intend to continue my ramp up on all things Android in the next sprint, including setting up unit tests to check if functional code is working. Another goal is to correctly incorporate the async function in Android to interact with multiple Rest endpoints, or else look for a better way to do this.

Retrospective

In truth I probably spent too much time researching the deployment of the war file, but it was a very interesting topic that tied in with something I was working on in my day job. This month as a whole was quite difficult because of family and work commitments. I did not get to put the time in that I wanted, but still managed to modify the plan accordingly. I watched more tutorials because I had more time on buses etc...

One thing I need to improve on is keeping focus when researching a topic. I started working on Android async calls and ended up researching frame layouts. Maybe this was a planning issue from the start or maybe it was my wandering brain, either way it probably cost me a couple of hours of indecision.

Agile Development and Planning

Sprint Planning

The goals of the next sprint are:

- To complete the functionality on the app in relation to Allergen Advice call and template
- Research using GPS to identify locale.

Some tasks I intend to carry out and a rough estimate as to how long I expect each task to take.

Estimated available time: 15 hours (7.5 hrs. per week)

Story: Develop Allergen Advice functionality on Android

Acceptance Criteria

Given: My country context is set

As a user: When I press the Allergen Advice button

Then: I retrieve information about the country set in context

- Task: Create Allergen Advice implementation (8 hours)
- Task: Add unit tests (3 hours)

Technical Task: Investigate using GPS tracking to identify locale

- Task: Research unit testing for Android (3 hours)

Agile terms and techniques

- Agile: A conceptual framework made up of many sub components intended to drive reactive software projects.
- Sprint: A two-week period of time devoted to a predetermined set of work goals.
- Story: A description of a feature from the perspective of the end-user.
- Acceptance Criteria: The functional requirements of a story.
- Ramp Up: Period of learning set aside to learn a skill that a story has a dependency on.
- Sprint Planning: Review of hours available and deciding the work involved in the next sprint.
- Sprint Retrospective: Reviewing the current sprint on its last day to reflect on good and bad decisions made and how to improve.

Software Journal December

Niall Whelehan

x12102954

niall.whelehan@student.ncirl.ie

BSc (Hons) in Computing (Evening)

Allergy Ambassador

Month Four (Dec 2nd - Jan 5th)

Due to a sudden loss in the family my December was did not reflect the plan that I had made for it at all. I did not work on the project at all until the middle of the month. But this had a positive impact because when I came back to the project I saw it fresh. I picked three key things that needed to change and revised my plan for the remaining months.

Data is the core of this project

The project data was being stored in a mongo repository deployed locally to the server. It occurred to me that if Amazon were to pull down the server then my code would be fine, but all of my data would be lost. This completely scattered my brain and I began devising a solution to avoid this.

I considered backing up my Mongo using scripts that were run on a timer. This would be costly in terms of time and testing.

I considered hard coding the data to the server. This would mean having to restart the server in a production environment every time some table needed to be updated.

In the end I decided to migrate the current data implementation to work with Amazons own DynamoDB. The similarities with a Mongo are obvious: it is a NoSQL database, it uses Items instead of Documents but both are represented as JSON format, they are both easily scalable. Unlike a relational database, the schema does not need to be as rigid so additional fields can be added as the application evolves.

DynamoDB, an obsession

There was a point in the middle of the holiday where I found myself awake at 3 AM studying the source code for the Java code required to converse with the DB. I already had unit tests for how I wanted my features to work so I decided I would leave the acceptance criteria exactly the same and make the code do what I wanted.

In all the migration took about 12 hours, from conception to implementation but I now have a database that is hosted by Amazon. With it I have a very useful developers console and the promise of greater security and backup.

The features

The two additional changes I felt needed to be made were the core functionality of the application. I did not want to saturate the application with unnecessary features but also did not want to leave it too simple. I decided the two core features would now be Allergy Advice and a Phrasebook of common terms that could be translated from one language to another.

In order to extend the functionality my second other decision was to invest in a Smartwatch that I will include the Phrasebook feature. With no experience of Smartwatches and very little experience of Android it is my plan to finish out the backend, database and build environment by February so that I can focus purely on the client for the final months.

Retrospective

This was a funny sprint in which I achieved none of the goals I had set previously. Usually this would be a bad thing, but instead the step back has left me with a clearer vision of where I am and where I want to be. Had I chosen a hard target, rigid business plan like Waterfall my project would be in flames right now, but because of the adaptive nature of the Agile process I can replan and carry on with very little impact.

Agile Development and Planning

Sprint Planning

The goals of the next sprint are:

- Finish testing DynamoDB
- Assure that logs and build scripts are correct in the Amazon instance
- Finish implementation of Phrasebook feature in the backend

Some tasks I intend to carry out and a rough estimate as to how long I expect each task to take.

Estimated available time: 15 hours (7.5 hrs. per week)

Technical Task: Finish testing DynamoDB

Acceptance Criteria

Given: I run the requested put/delete/retrieve/update command.

Then: The expected behavior is carried out.

- Task: Work with developer console and test java class (5 hours)

Technical Task: Inspect Amazon EC2 instance to ensure logs and build are deployed correctly

- Task: Review shell scripts for deployment (local and Production) (2 hours)
- Task: Review logs of expected errors and HTTP 200 requests to ensure expected behavior is being recorded. (2 hours)

Story: Complete Phrasebook implementation on backend

- Task: Define and implement Phrasebook model (1 hour)
- Task: Create Basic Controller to get, add, update and delete phrasebook entries(5 hours)

Agile terms and techniques

- Agile: A conceptual framework made up of many sub components intended to drive reactive software projects.
- Sprint: A two-week period of time devoted to a predetermined set of work goals.
- Story: A description of a feature from the perspective of the end-user.
- Acceptance Criteria: The functional requirements of a story.
- Ramp Up: Period of learning set aside to learn a skill that a story has a dependency on.
- Sprint Planning: Review of hours available and deciding the work involved in the next sprint.
- Sprint Retrospective: Reviewing the current sprint on its last day to reflect on good and bad decisions made and how to improve.

Software Journal January

Niall Whelehan

x12102954

niall.whelehan@student.ncirl.ie

BSc (Hons) in Computing (Evening)

Allergy Ambassador

Starting Client Development

Month Five (Jan 6th - Feb 2nd)

This month I finished working on the backend and environments for now. My Android watch arrived and I got to play around a bit with it, so getting excited about the prospect of designing an application for it.

My reality check this month was learning what Android development was about compared to what I thought it was about. There are a lot of pitfalls like strings overwriting other values when naming conventions fail, or how too many activities is the death of an application.

Making Up for Lost Time

I managed to wrap up all the DynamoDB testing, environmental monitoring and Phrasebook implementation within this Sprint with time to spare.

Naming Conventions

Everything in Android is referenced using Strings and int id's. From layouts to class placeholders, everything should be given an id to define within View states in the application main flow. If the naming convention is not completely transparent you run the risk of not being able to tell what the difference is between a String reference is to a ListView.

In certain cases, Compile time errors are not thrown to signify a compatibility issue and so you need to navigate through the application to find problems.

Activities and Fragments

When an application is started the main Activity is what is opened. Within the activity can be multiple Fragments. Fragments can be injected into Activities the same way that Ruby and .NET use placeholders.

The problem with too many Activities is how expensive they are to system memory and therefore slow the application down. Once I finished the server tasks from the last planning, I spent a lot of time researching Fragments. My plan now is to use as few Activities as possible and instead use Fractions for simple form displays. If a View is in some way complex, then it probably warrants an additional Activity.

User Interface and User Experience

This month I reviewed my original View of the application and now believe that instead of using multiple images based on what country is selected, a color palette and theme should be applied. The motivation should be health and all things therapeutic. So calm, warm and living colors should be present throughout the application.

Retrospective

This Sprint was very productive. I managed to make up for some lost time. The environments, database and mobile backend are all in a good state. I have started on the application planning and development. One thing I could have done better was to assess the work required more accurately, because I finished the backend work very early.

I am excited to work with Android. It is a new experience for me to try to apply the professional standard I have now acquired to a front end development. I have lots of ideas but also lots of knowledge gaps, so I think I need to hit the books next month.

Agile Development and Planning

Sprint Planning

The goals of the next sprint are:

- Create a development plan for front end
- Investigate http request solutions
- Implement Phrasebook and Allergen Advice models

Some tasks I intend to carry out and a rough estimate as to how long I expect each task to take.

Estimated available time: 15 hours (7.5 hrs. per week)

Planning Task: Create a development plan for front end (6 hours)

Discovery Task: Investigate http request solutions

- Review 3rd party http library “Retrofit” (3 hours)

Story: Implement Phrasebook and Allergen Advice models

- Task: Create Allergen Advice model (3 hours)
- Task: Create Phrasebook model (3 hours)

Agile terms and techniques

- Agile: A conceptual framework made up of many sub components intended to drive reactive software projects.
- Sprint: A two-week period of time devoted to a predetermined set of work goals.
- Story: A description of a feature from the perspective of the end-user.
- Acceptance Criteria: The functional requirements of a story.
- Ramp Up: Period of learning set aside to learn a skill that a story has a dependency on.
- Sprint Planning: Review of hours available and deciding the work involved in the next sprint.

- Sprint Retrospective: Reviewing the current sprint on its last day to reflect on good and bad decisions made and how to improve.

APPENDIX(IV) PLAY STORE FOCUS GROUP BUG TRACKER SPREADSHEET

Bug Tracker *4/15 completed*

✓	Date	Issue	Logged By	
*	~~19/4~~	~~Infitite Loop causing phone to crash~~	~~Niall~~	~~Critical~~
*	~~20/4~~	~~Peanut + milk image not displaying~~	~~Niall~~	~~Major~~
*	~~20/4~~	~~When translation is the same in multiple languages the language is repeated as the label (example, 2 spanish translations for lupin)~~	~~Niall~~	~~Major~~
	21/4	Cannot exit the app using back button (restarts the app)	Niall	Critical
	21/4	Settings does not auto update when something is changed	Niall	Minor
	21/4	Only Ireland has full allergen information so far	Niall	Minor
	21/4	tried to access settings button and app stopped and forced closed	Paddy	Critical
	21/4	Phrase book scroll to bottom and back up results in crash	Conor	Critical
	21/4	in information tab it is not clear country is selectable	Paddy	Minor
	21/4	app allows user to back out to gmail login screen resulting in the app crashing	Conor	Critical
	21/4	Clicking into settings is crashing the build :(	Conor	Critical
		~~Not sure if its because I've moved to multiple~~		

APPENDIX(V) PLAY STORE FOCUS GROUP FEEDBACK SPREADSHEET

Feedback *0/7 completed*

✓	Date	Feedback	Logged By	Weight
	19/4	Cards would be better than long text for allergen advice. At the moment it is too much to read	Niall	Greatly Improve
	19/4	"Phrasebook" is not iterative and should be reviewed as a name	Natasha	Nice Touch
	20/4	Phrasebook translations should be in appear in a drop down view instead of a scroll view	Niall	Nice Touch
	20/4	More countries info needs to be added	Niall	Necessity
	20/4	By changing language in settings all text should be translated	Niall	Necessity
	21/4	from the main screen I can swipe right but see no images in phrasebook until the load, should be some kind of loading pop up as its confusing for the user	Conor	Nice Touch
	21/4	kind of like above if you swipe back left before phrasebook loads the data the app crashes	Conor	Nice Touch