Group Progress Report

Overall, a lot of progress has been made in the Implementation Phase. Components such as the User-information Database, Data Gathering systems, Text Translation and SMS sending are all complete, and every component has had significant features implemented. Features still remain to be produced, these features are mainly component-integration based and these can be completed during the next two weeks; the Integration Phase of the project.

Overview of major components and their status

- Data Gathering - Completed
    - This system collects weather and market data. Weather data is collected using Dark Sky and market data with Food Security Portal (for local crop prices), Currency Converter API, and OpenStreetMap Nominatim (for specifying countries from coordinates). Currently Food Security Portal only supports Rice and Maize, so they are the only two crops we will support.
    - Systems were tested by putting in various coordinates and evaluating the responses. Locations tested were Cambridge (which has no crop data), Kampala, Maputo and the North Pole (which has no crop nor currency data because it isn't in a country), and sensible results were produced for all. The data produced was cross checked against other weather forecasts, and compared to the expected crop value from Food Security Portal.
- User Database - Completed
    - This system is responsible for converting Java OOP objects into tabular form for persistent storage in a database.
    - There are several layers of the system: the actual database, which is interfaced using SQL; the object-relational mapping, converting objects into tabular entities; and lastly, the service that abstracts away persistence details and interacts with the rest of the system, providing "store User", "find User", "update User" functionality, as if it is a simple non-persistent Java Collection.
    - The actual database was implemented using H2 and its embedded database. Persistence across restarts is achieved by using a .mv.db file for storage. Object-relational mapping is handled using hibernate. JPA styles were used extensively to use annotations (instead of XML) for configuration of Entities, mappings of Enums to Strings, etc.
    - The Spring Framework is extensively used to reduce the amount of actual code, as well as XML configuration files (or @Configuration-marked classes), written. In particular, Spring Boot's auto-configuration was used. The amount of boilerplate code written was near zero. In the end, no SQL queries were explicitly written.
    - Since the system rests within the Spring Framework and cannot be tested from a static, command-line context, it was tested using @RequestMapping and a browser.
        - Existing Users, when re-registered, are successfully overwritten.
        - Users stored in the database do persist across restarts.

        - Deletion is functional.
        - The UserService class does return User objects ready for direct use across the rest of the system; no further conversion or SQL was required.
    - The original intention was to also make use of database functionality to provide a mapping between country/language codes used in Twilio, Google Translate, etc. But the need for such a mapping was proven non-existent, so this was not implemented.
    - Remains to be done: specialised query methods that return a subset of registered users based on common properties (such as location, language, or crop preferences).
- Message Generation - Partially complete
    - This system currently queries the data gathering system and packages messages and alerts into text. It will need to be integrated with the database so for a given user it can extract relevant information from the database and generate a message tailored to them, but this will be done next as part of the Integration Phase. The current system is being used with part of the SMS sending system to send example update messages (using correct information but without taking input from the user database), testing both this component and the SMS sending component.
    - It will need to be integrated with the website to accept custom messages, and this system will need to store these messages decide whether to send them to users based on their location.
- SMS Sending & Text Translation - Complete
    - In order to send the SMS messages, the interface and classes send messages to phone numbers was completed and there is now a system by which a text message can be sent to any number with any text.
    - This involved connecting the system to the Google Cloud Translate API (which while paid, has a $300 free trial and this is being used)
        - This side is being tested using JUnit unit testing in a few languages to ensure the data is being returned from the server in the format which is expected and the object mapping is working as expected.
    - This system needs to be connected to the side which is producing messages, as well as the database to determine what the language which is required, in order to send the messages once a day.

IVR & Voice Translation - In progress
    - Have decided to change tack and use a Google Text-to-voice API (soundoftext.com) when we found that the text to sound of the Twilio API was quite poor. This was implemented to get an mp3 of the required text and produce a URL
    - Also implemented a basic test system to deal with incoming text and respond with an mp3 (for a basic test). This determined that our API was giving the mp3 with a few incorrect pieces of information. Therefore, a system was created which downloads the mp3 on the server and then gives a URL to Twilio which links back to the server.
        - Concerned this would add latency, so tested this and found acceptable results with long phrases.
    - Must still add all the connection between phrases as well as adding the connection to data delivery system
- Crop Reporting
    - Need to implement the recording system - this has been generally tested but not implemented on the IVR
    - And then deliver this to the web server, though this has been determined to be best done by simply passing a URL - which Twilio provides.
    - This is inherently difficult to test - generally it must be done manually with phrases into known languages and ensuring that the mp3 is correct - avoiding the Twilio phase. This also must be tested by calling up the phone number and see if expected actions occur
- SMS Receiving - Partially complete
    - SMS receiving system has been completed, it just needs to be connected to the configuration system and the prompting of the message generation where required
    - This is hard to test properly as it requires SMS messages to be sent to the system. However, the inputs of the texts for unit testing using JUnit can be emulated.
    - Can also test the responding to HTTP requests through emulating requests in Postman. Finally, we plan to do proper testing by sending messages from an actual phone
- Registration Protocol - In progress
    - The registration protocol is the bridge between the User Database and the Twilio functionality which handles incoming calls and messages. The actual implementation phase can only begin after both has been completed.
    - The User Database is ready for use; significant progress has been made in Twilio integration, and we have conducted tests such as receiving weather update messages in English, Chinese, etc. via SMS.
    - We have discussed and come up with the intended strategy to perform registration, using a Map that stores state for each user in the process of registration, resting within a Spring Controller. Of course, there would be the need to consider the thread-safety of this map.
    - We expect this module to be implemented and tested during the integration phase in the following weeks.

- Crop Report Database - Completed
  - Accepts and contains recordings of crop health reports, providing them to the website.
  - Built using the same database as the user database, which is described above. It stores a URL for each report, which points to an mp3 located on the Twilio server, containing the voice recording of the report. With each URL it also stores a timestamp and a reference to the user who submitted the report.
  - The database was tested by registering mock users, and submitting reports using @RequestMapping and a browser.
- Web-based Interface - In progress
  - Login page built using PHP and MySQL to manage the database of users and the authentication. Testing of this has included adding / removing users and seeing if they are still able to login and testing against vulnerabilities such as SQL injection to ensure protection has been correctly implemented.
  - Next we have to build the page where members can insert a message to be broadcasted to a location at a set time, and then send the information over to SMS sending.
  - We need to connect this to the other server, passing the custom messages and metadata from the web-server to the system handling the generation of update messages.