Software Architecture Specification

Overview

The Flora of Indiana application is a mobile application which will reside on the users’ physical mobile devices and will allow the user to access information that is stored in a plant api. The source of this api is still unknown, we plan on determining that soon. If not able to, we will resort to just Indiana. The application software will be compiled for both Android and IOS applications. It will be distributed via each operating system’s primary application store. We will most likely be using nativescript but we are still researching.

Subsystem Decomposition

**Database** – The data storage mechanism in our case will be a database of plant species throughout the united states. It will also use geo-location to determine the amount of plants displayed on the device. It will be maintained by other institutions such as herbariums, colleges, and other scientific organizations. We will interact with the databases through Symbiota.

**Determining API** – This system will be a third-party application which communicates with different databases and or one whole database that will retrieve information from those databases. This will have the access point for the data via a defined API.

**Application** – The application will be developed and maintained by our team and will provide the logic which will govern the organization and presentation of data as well as the options for data interaction which will be available to the user.

\*Depending on if we use Xamarin--->

• **Data Access Layer** – This system will be responsible for retrieving data from sources external to the application (the database via Symbiota) and will also be responsible for handing information to the rest of the application.

• **Plant Data Model** – This system will function as a data structure and a packaging mechanism. It will be the organizing container for data retrieved by the data access point. The model will define a valid format to which retrieved data will be required to conform. This will be the “object-maker,” and individual plant data objects will be able to be interacted with by other application subsystems.

• **Controllers** – This system will be responsible for accepting user input from mobile devices and converting that input into commands that are capable of acting upon the data model and views.

• **Views** – The views will provide visual interfaces and affordances to the user, communicating to the user which actions are possible within the system. The views will be the only means through with the user will access or manipulate the model and the data which it represents.

\*Depending on if we use NativeScript

Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that will be imported into apps.

* **Modules:** Declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. It will be stored in a root module typically named AppModule.
* **Components**: Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.
* **Templates**: Combines HTML with Angular markup that can modify HTML elements before they are displayed.
* **Directives**: provide program logic, and binding markup connects your application data and the DOM. There are two types of data binding:
  + Event binding lets your app respond to user input in the target environment by updating your application data.
  + Property binding lets you interpolate values that are computed from your application data into the HTML.
* **Services**: Share across components
* **Dependency Injection**: Lets you keep your component classes lean and efficient.
* **Routing**: Defines a navigation path among the different application states and view hierarchies in your app.

**User Interface** – Users will use touch screens to interact with our system on physical mobile devices. The user interface will be rendered via an Android or iOS operating system and API/libraries, depending upon the device. This system will be designed and maintained by a third party.

Hardware/Software Mapping

**Database** – The “database” in our case will actually consist of either one complete API that is obtained through one or multiple databases depending on what we find. Interaction with these databases will depend on how we get our API, if we get it through multiple different sources it might need to be maintained separately by said third party.

**Symbiota** – This is one of the hosted third parties that the previous group used. This application will probably be hosted on a third party machine or a series of machines and will be maintained by a third party.

**Application** – The application code will be deployed to the Android and iOS application stores, and users will be able to access the code from there and “deploy” the software to their own physical devices by downloading it. Once downloaded, the application will reside on the user’s physical device.

**User Interface** – The user interface will reside entirely on the user’s device, which will be maintained by the user and the third parties that manufactured it and provided the operation system.

Persistent Data Management

Data will be stored in two locations:

• In physical (or virtual) databases, pushed from aPIS or databases from different sources.

• In data objects which will be structured and validated by the application logic and will reside on the user’s device.

The database will contain a large number of details about individual plant specimens and species. The majority of this information is unknown to us currently, as we are still in the process of gaining access to the databases.

The application will store data defined in the database or API that we will be using, this data will be stored for quick use, depending on location.

The Plant Model will require the following information:

• **ID** – The external database appears to use a numeric primary key. If this is the case, we will use the same numeric ID within our system to facilitate simpler communication

• **Genus** and **Species** – The scientific genus and species name, two data points which, together, will provide a functional (and traditional) primary key.

• **Thumbnail or identifying photograph** – A small, representative photograph of the plant species, to aid in identification of the plant.

• **Description** – A description of the plant, to aid in identification.

• **Qualities** – A list of distinct plant qualities upon which the user will be able to filter species.

Access Control and Security

All users of this application will have the same level of access. The application will be downloaded from an application store, and access to that store will be managed by Google or Apple, depending upon the device. No authentication will be performed. Furthermore, our application will only provide read-only access to the user, so data will not be able to be updated from our application. To prevent malicious use of the read-only access, we will limit the number of requests which an individual device will be able to make to the database.

Global Software Control

The system will be event-driven, with the user submitting a series of commands to the system, and the system responding by providing modeled data through a series of views.

follows:

Architecture overview:

* Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that will be imported into apps.
* **Modules:** Declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. It will be stored in a root module typically named AppModule.
* **Components**: Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.
* **Templates**: Combines HTML with Angular markup that can modify HTML elements before they are displayed.
* **Directives**: provide program logic, and binding markup connects your application data and the DOM. There are two types of data binding:
  + Event binding lets your app respond to user input in the target environment by updating your application data.
  + Property binding lets you interpolate values that are computed from your application data into the HTML.
* **Services**: Share across components
* **Dependency Injection**: Lets you keep your component classes lean and efficient.
* **Routing**: Defines a navigation path among the different application states and view hierarchies in your app.

Boundary Conditions

The databases will be maintained by third parties. The raw source code will be uploaded to GitHub and will be accessible and compilable by anyone who wishes to access it. The compiled code will be deployed by the developers to the application stores on which our application will initially be stored. This will likely be a variety of cloud servers maintained by the third parties which offer these application stores. The compiled code will then be downloadable by the users and will be deployed to their physical devices, where the application will be bootable by clicking on the application’s icon on their device. The application will be shut down by whichever operating-system-specific method the user chooses to utilize.

The underlying data will be managed by third party organizations, and no maintenance of the data will be required outside of any maintenance that those third party organizations would have already performed. The same will be true of the application hosting servers and the users’ devices. Any maintenance will be performed by third parties.

Application errors and exceptions will be handled programmatically through code. Physical hardware errors will be handled by the users’ device manufacturers and OS developers.