Architectural design & justifications

Our overall architecture is a mix of multiple different architectural styles in order to ensure that our system can allow for separation of concerns, loose coupling, and levels of abstraction. To begin it makes use of a data centric architecture, through the integration of a centralized database which helps facilitate data storage and sharing between data accessors, which are in the web and app servers. These data accessors all use the database as a means of indirectly talking with each other. This type of architecture reduces the overhead for data transfer between data accessors. This also provides the system with the ability to be scaled up easily.

The next architectural style that is in use in our system is a client server architecture, which exists between the device (Android Smartphone, tablets) the web server and the app server. This type of architecture limits the client server relationship to a request-response messaging pattern thereby allowing the system to be easily scalable, such as by adding more clients. This type of architectural style could also be considered N-Tier because of the use of an application server, which is a client of the database. This design is also used because it makes provisions for security concerns.

The server uses an n-layer architecture with a persistence architecture as its final layer (MVC pattern with a REST API as the view). The server architecture at its first level is a REST API, and as such is an interface of the functions available to the clients. This API communicates with the next layers, controlled using Node.js , which control business logic. The final layer is a persistence layer consisting of an interface to a cloud firebase database. By using a persistence architecture here, the admin can change database(such that can perform CRUD functionality) without modifying the entire system (low coupling is achieved). The smartphone application uses an n-layer architecture, as is good for interface design. This is a suitable architecture, as the application is designed using Angular and Ionic, both of which utilize the MVC (Model View Controller) pattern. The first layer is the View, presenting the user with a user interface, rendered to the user using JavaScript, HTML and CSS, the second is the Controller, implemented in client-side JavaScript, which controls client side business logic (any processing which can be offloaded from the server, such as aggregating data), and the last layer is the Model. The model layer is simply an HTTP interface (Angular HttpClient/Browser fetch API) to the server. Using the n-layer architecture with MVC provides low coupling and high cohesion; modification to the user interface of the application can be done without having to modify business logic, and features can be added, removed or altered with ease. This means designers and programmers can work concurrently, and testing of new interfaces (e.g. A/B testing) can be performed without modifying functionality.