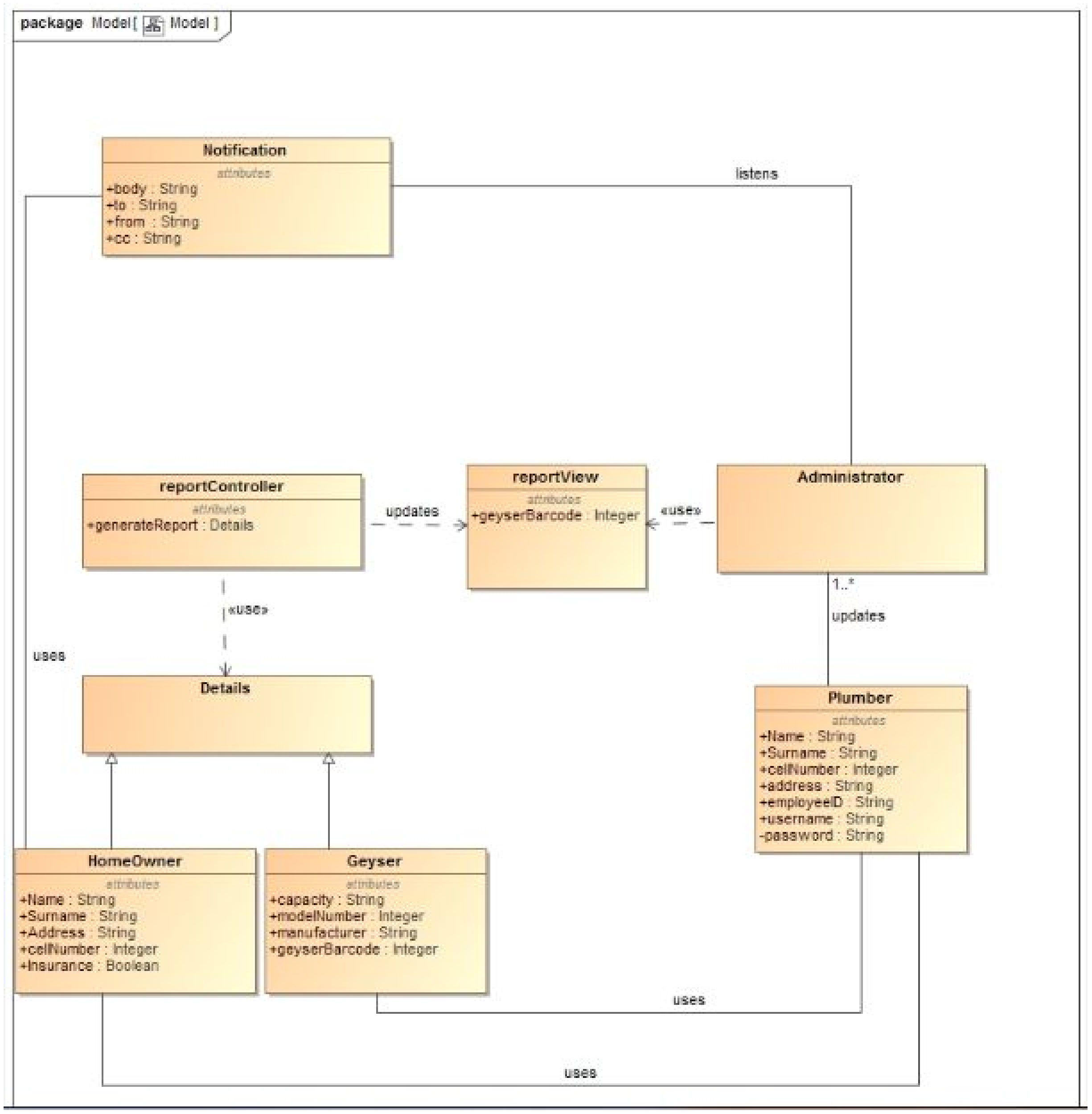
COS 301
HighTech

Theoveshan Naidu (16148861)
Janaki Patil (16006110)
Tristan Sander-Hughes (17071390)
Zi Xin Zhang (15192556)
Alexandros Petrou (15291792)

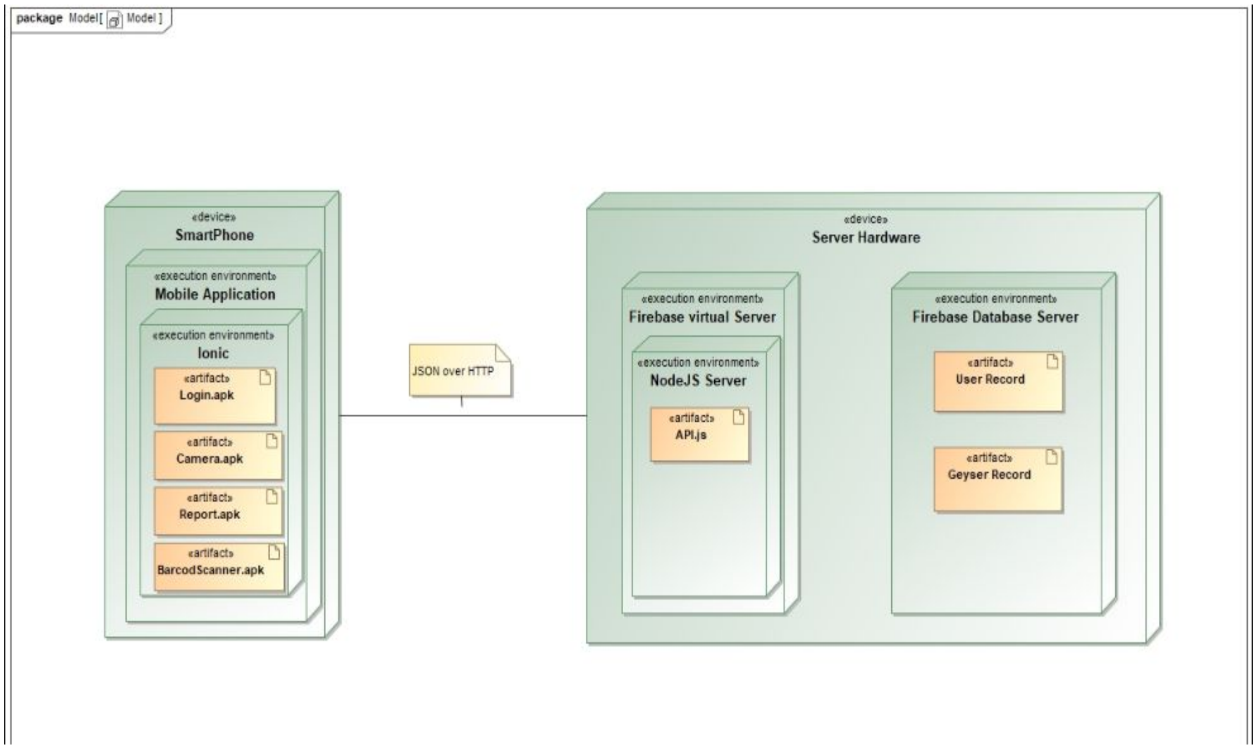
Introduction

In the modern plumbing industry, a continuous source of inconvenience arises for plumbers to log information on the geysers and components installed on various jobs by means of written material on a day-to-day basis. Considering this the goal of this project is to design a new database application system that greatly reduces the inconvenience of having to log information about installed geysers by taking advantage of the prevalence of mobile devices in the modern world. In order to do this, we will shift the logging of details from the paper-based method itself to a mobile application, detailed below.

Corrections to document of demo 1



Deployment model



Technology decisions

Ionic Framework
We chose Ionic Framework because is a free open source mobile development framework that allows us to create cross-platform applications that are native to the platform. Ionic allows us to create applications with Javascript/Typescript, HTML and CSS. We wanted that functionality because it would allow us to not only port our application into a web-app, but also a desktop environment using Electron Framework all using a single codebase.

Firebase
We chose firebase because Firebase provides a realtime database and backend as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase cloud. The database is also accessible through a REST API and bindings for several JavaScript frameworks such as AngularJS, React, Ember.js and Backbone.js.The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Firebase allow allows cloud storage to store files,images, etc. In our case we have to store the installation images of the geyser.

Typescript
Typescript is a typed superset of Javascript that compiles into plain Javascript. We choose this language because it integrates well with Ionic, and it allows us to write typed code which helps us catch errors at compile time, and also helps us debug our application much easier than a dynamic language. Typescript is also free and open source with a large community that would help us get libraries and help that we might need.

Git
We chose to use Github because it is a free centralized area what we can all collaborate using the Git Work-Flow that allows us all to work on the same project effectively and with as less conflict as possible. It has many different tools that allows us to review each other's work, and also help.

Constraints

Android API levels
We are limited to only working with Android devices and because android devices each all have different API levels that allow for different features and functionality, we had to make sure that our application was limited to only using the API level of the lowest common android.

Open Source / Free
We are limited in only using open source or free software and framework to build our application, proprietary software would be too expensive, or a limitation should this application wish to do something that is against their software policies.

Architectural requirements and justification

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 database-centric architecture, through the integration of a centralized database which helps facilitate data storage and sharing between data accessors. 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 n-layer 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.

Quality requirements

1. Performance
   (a) First load must be fast on slow networks (including 3G)
   - Use Lighthouse on a Nexus 5 (or similar) to verify time to interactive less than 10 seconds for first visit on a simulated 3G network.

   (b) The Geyser server system must be able to handle large amounts of traffic without slowing down
   - Tools like Website Grader and Web Page Test will be used to analyze the performance of the system
2. Efficiency
   (a) The Geyser server system must efficiently make use of available bandwidth to ensure performance on slow connections to the server. This will be tested by throttling the network to simulate a slow connection.
   (b) The Geyser server system's mobile application must be implemented as a PWA to ensure that it is multiplatform making the system efficient on Android mobile operating systems.

3. Reliability
   (a) The Geyser server system must be fast loading and work offline. This will be accomplished using service workers to cache data, and will be tested using software tests.
   (b) The Geyser server system must use an OwnCloud backup online software for integration with 3rd-party cloud storage providers and supports data backup and restore for Windows, or Linux. Synchronisation will occur hourly.
   (c) The Geyser server system must support a CDMA (Code-division multiple access) channel as data communication protocol to ensure reliability and quality of data.
4. Security
   (a) Each subsystem must be thoroughly checked and tested to ensure security and error handling .

- Before the system is published it will be stress tested which will also include several ethical hackers attempting to hack into the system and retrieve confidential information.

(b) The system must use a user name password pair authentication for logging in.

- Check that the user is authorized by the app

(c) Mobile app must be immune to malicious code injection

(d) Prevent Brute Force Attacks

- Three incorrect attempts in logging in will result in a timed block which will increase exponentially for each consecutive incorrect attempt after the first three.

(e) The system must notify the correct people, via email, once a logging of details has been performed.

(f) Communication between server and mobile app must use end-to-end encryption

5. Monitorability

(a) The system must use Log4j to log information such as errors encountered by users while they are using the app

(b) The system must use a Datadog to monitor service for cloud-scale applications, servers, database, tools and services

(c) The system shall use SPSS software for editing and analyzing all sorts of data.

6. Integrability

(a) The system must effectively use all necessary subsystems in a logging.

- Git must be used to make branches, track changes and integrate (merge) branches with the master system. Git flow will be utilised to ensure stability.

- Ionic must be used as a platform where modules can be developed and integrated into one system

(b) The system must integrate with the functionality offered by smartphones

- The system must use authentication methods provided by the smartphone to unlock the application
- The system must use hardware like a camera provided by some smartphones

7. Cost

(a) The system must use existing technologies and open source libraries to keep costs to a minimum

(b) The ATM device must run off of low-end, low-power hardware such as a cheap Android phone.

8. Usability
    (a) The system must be responsive. This will be tested using ionic developer tools to record interaction time.
    (b) The system must be user friendly and accessible. This will be record using Chrome developer tools contrast checking and Google's Mobile Usability Report.
        - The system will be tested by a group of first time users varying in age and preferably from a plumbing background to ensure that it can be easily used by anyone
9. Availability
    (a) System server failure must not affect the whole system. Redundancy should be implemented as a fallback for the server.
    (b) The system should continue to operate even during maintenance.
    (c) Mobile app functionality must be available as long as the cell phone is on.
10. Portability
    (a) The Geyser app system will run on a mobile device to make it fully portable