Zachary DeVita
CS 496
Final Exam

The purpose of this document is to take an in-depth analysis of the Space Shooter game which I created for my final project. It will be evaluating the game on the following quality attributes: Portability, Usability, Reliability, Performance, Security, and Interoperability.

Portability

Portability, in regards to software, refers to the ability to transfer an application from one platform, or environment, to another. It is difficult to quantify an attribute such as portability so my goal is to compare the application's portability with other mobile applications, and provide some pros and cons of my chosen framework.

As far as my backend goes, it has absolute portability. I am using a Node.js framework for my server which utilizes a DynamoDB database to query data from, but what's important to realize is that it doesn't affect the portability of the application in the slightest. Because the application simply requests and sends data to the server via POST and GET requests, the server is independent from the user's device or operating system. Accessing data through an API is a standard process which all devices and all operating systems utilize. The header created from an iPhone request looks the same, and is interpreted the same, as the header created from an Android, Blackberry or a Windows Mobile device request.

The frontend of a mobile application is what separates one application from another in regards to portability. iOS applications are most commonly programmed using Objective-C or Apple's new language Swift. Android applications are primarily written in Java. Each of these frameworks comes with their own platform-specific APIs which can only be accessed using one of these languages on the appropriate devices which makes it difficult to port over an application without completely rewriting it in another language. Generally this is exactly what needs to occur in order to make a mobile application available on another platform. This is an excellent example of how an application would have zero portability qualities.

Because I have utilized Unity which is a cross-platform game engine, my mobile application does not have any of these source portability issues. My program is actually written in C# which is not a platform-specific language, and I don't require the use of any platform-specific APIs. I chose to create my application for an Android device due to the fact that I own an Android device, and it was necessary so that I could test the application during development, but switching between platforms is as simple as selecting an alternate platform from a drop-down menu, and selecting the "Build" option. The Unity Game Engine has the build-in functionality to compile the C# scripts used to build the game into the language required for whichever device you choose. I could just as easily have chosen to develop the application for iOS, Blackberry, or Windows Mobile, and I still can in just a couple minutes time. This is absolute source portability; I don't think there is a method for writing a mobile application which could rate higher in portability.

My mobile application is equally as robust when it comes to binary portability. I developed the application so that it can be ran using Android operating systems from Android 2.3.1, “Gingerbread,” forward. This means that 99.9% of all Android devices can utilize my application, and that number will only increase. Again, this is a fundamental feature included in the Unity Engine. I simply had to choose the Android operating system prior to clicking the “Build” button for the application to include all Android platforms from Android 2.3.1 forward. My application is extraordinarily portable in regard to source, as well as, binary portability.

Usability

Usability refers to how easy an application is for user’s to use or learn how to use. Usability is a quality attribute which is solely evaluated based on the frontend of an application; the part of the application which the user interacts with. I will compare and contrast usability in regards to complexity of the application, its responsiveness, aesthetics, and ease of installation and use.

As far as complexity, the application is simple enough that it could be easily used by a child. The application is focused, and, aside from the registration and login, it only provides a single function. The only features of the application pertain to the user’s account information, and not the game part of the application. There are no options or settings regarding the game itself. The user can change their personal account information like their username, password, or the phone number associated with the account but that is it.

The application has some pros and cons regarding its responsiveness. The application absolutely utilizes “coroutines” which are the Unity Engine version of forking off a process. This allows for many processes to be ran asynchronously and it maximizes the performance from a user’s point of view. There is only one part of the application where there is a period of unresponsiveness and this is when the user’s account page is loaded. This is because there must be a complex API request done to retrieve all of the user’s data from multiple tables in the database, parse the data, sort the user’s high scores, serialize the data back into JSON, and then de-serialize the data once more on the frontend of the application. The pause isn’t very long, but it is definitely noticeable. To improve this I would have liked to implement a loading screen with some kind of progress bar so that the user would know, right away, what is going on.

When it comes to aesthetics I spent a considerable amount of time choosing colors which go well together for my buttons and the background of the UI menus. It is very clear what each button does based on the label, and they are structured nicely on each menu page. There are not too many buttons on each page, to the point where it becomes difficult to find whatever action you might be trying to complete. There is nothing distracting on the pages. The layouts of each page are consistent with one another, i.e. the menu headings are in the same location, the buttons are placed in the same place relative to the input or labels associated with them. There is definitely a visual balance of elements on each page with a clear contrast of color between elements. The login screen, registration screen, and account screen all utilize the common convention associated with the layout of elements in mobile apps, so the pages all look familiar to any given user.

The Android App Store makes the install of the application simple. The user simply needs to type in the name of the app in the search bar and press the install button, and the user's Android device will take care of the rest. Because the application is so simple, it makes using the application very easy. If the user has ever registered for an application and used a login to authenticate their credentials, than they should have absolutely no trouble using this application. This is the most complex part of the application and my implementation of it is as simple as possible. I even build error prompts explaining to the user what has happened in the case that they have input invalid data into the system. This makes it so that there is never a point when the user enters invalid data and nothing happens. One of the only improvements that I would have liked to implement was the "Forgot Password" feature, where an email would be sent to the user, in this case, so that they could regain access to their account.

Reliability

Reliability of an application generally is referring to the backend, whether that is the server itself or the database where the data being retrieved is stored. As far as my database goes, DynamoDB has an extraordinarily reliable cloud-based infrastructure. I don't know that there are any comparable non-relational database platforms which can be considered more reliable than Amazon's DynamoDB. With that said I will jump into a description of some of DynamoDB's features.

While DynamoDB offers load balancing via their "Elastic Load Balancing" tool, I personally did not implement it. Much of the features which are available to improve reliability of an application would have been silly to implement in an application where there would never be more than one user accessing data at any given time; this is one of those features.

For my personal backend implementation, I have uploaded my server script using Amazon's AWS cloud service. This service opens up the potential of having many people using my application, or acquiring users in far-away places because AWS stores both my DynamoDB database and my server script in many datacenters across the world. This means the data is stored in multiple places so that if a datacenter goes down, your data will be fine due to the redundancy of data replication. This ensures a nationwide consistency for accessing your data.

One negative of using a non-relational database is that you can't assume that data which was just written to a database will immediately appear in a query. Sometimes it may take a few seconds to read back. I believe this is responsible for why some of my data would not load during my video presentation of my game. Right after a game was finished, and I returned to the account, my user high-scores would not update. This only happened in the video; I was able to get the data to load prior to recording, as well as, the day after the assignment was due. One update which I would implement in future designs would be to automatically use the data being sent to the database in an asynchronous operation opposed to sending the data to the database, and then querying the same data, and then waiting for it to be retrieved.

For the scope of this class, I set up any monitoring system. There are built-in tools provided by Amazon to do this sort of thing, but it seemed extraordinarily unnecessary, even silly to have

set up. Time was also a factor so it would be nice to learn how to use these sorts of tools, but it didn't seem like a very reasonable goal for this assignment. Since only one user would be using this application at any given time, and the server and database only need to be active for a week, it seemed more than fair to assume that there would be no problems associated with server congestion. For similar reasons, I did not consider using a hybrid cloud platform. It would have been time consuming, cost more money, and be absolutely unnecessary. There is not a concern for AWS cloud service to fail, I do not need access to any other platforms APIs, and I did not need further protection of access to my data.

Performance

The performance of my application will be evaluated based on its use of indexes, ability to minimize traffic, and parallelization. DynamoDB automatically will create a new index when a query is received with more than one member which definitely helps to improve performance. For my database implementation I not only use the table key for my "user" table, but I also utilize a local secondary key where an alternate range key is utilized for improving the speed that a user's data is retrieved. The way this works is that the primary key is made up of two attributes, the user's email in addition to the user's username which is used as the range attribute. This allows for direct query requests or making queries against the sorted range index.

There are several methods to minimize traffic to a database, and utilizing some of the available methods will inherently improve the performance of the application. One way of doing this is by only querying the data that is needed at any given moment. This is easy to implement in an application which only has three user interface menu pages, and where the only query request for data by the application is where all the data associated with the specific user is queried. Since there is no option to query less data in any of my requests it sort of limits any improvement in this regard. One thing I did do to improve performance was to implement batch get and write functions wherever appropriate. This ensured that the most efficient query was produced opposed to setting up a series of queries, one after another.

Related to this, I also utilize JSON serialization to transmit data so that the results are concise and many results can be returned in one transmission. One possible improvement might have been using an alternate platform; a platform like MongoDB uses true JSON serialization opposed to the bulkier "pseudo-JSON" which is used by DynamoDB. I don't think the difference would be noticeable, but there is a difference the amount of bytes required for each transmission. I also do all my computation, i.e. sorting the user's high scores and only returning their top ten scores, all on the server side. This helps to decrease the amount of data that must be sent to the application.

The only instance where parallelization is a factor would be what I had mentioned earlier, that I implemented bulk read and write functions when they were appropriate for my queries. That definitely makes sense why that would improve the performance of a server.

Security

As far as security goes, I have utilized different methods to help ensure confidentiality, integrity and availability of the application which, in turn, increase the overall security. I chose to implement my own security which uses a multitude of pre-existing security components that are provided via Node.js modules. Part of this system which I created uses passport.js for the registration and login portions of the application, as well as, session states which continually compares a “Secret”, or key, with one another to ensure that only the authorized user is accessing the account.

The security system that I have set up is simple and minimal. There is only a single login screen, there is only a single class which is responsible for user authorization, and the system only implements the necessary functionality that is needed to make an application secure. As far as the frontend implementation, I only included the minimal set of libraries needed to carry out the functionality of the application. The frontend also utilizes access modifiers where appropriate. Many of the functions which are only used by the class which they are in have been set to private. I could improve this a little by including inheritance and making all classes and functions that are not private set to protected.

In order to improve security between component layers, I have attempted to make each component independent of one another. This is done through a system of data encryption and serialization of data. The user’s password is encrypted before it gets transmitted to the database, and then is decrypted anytime it is being used to compare with the user’s password input at login. Data moved between the server and the database is also serialized before being sent out and de-serialized upon retrieval from the database. This is an important step in improving the security of data.

Injections of database queries are essentially made obsolete with a DynamoDB system. This is because the URIs do not support multiple actions. In order for a query injection the URI would need to allow for an injected, additional action from the single query. Additionally, DynamoDB has the added benefit of not needing data to be sanitized due to everything being parameterized.

The biggest improvement in terms of security that should be done in the application is removing my hard-coded database credentials from my server side script. They are currently available for anyone who can gain access to the server’s source code. This would be a difficult task, to gain access to the server source code, but I’m sure it could be done, and I am currently using the least secure way to store my database credentials. An improvement would be to store the config data in a separate config.json file. I could also set up some kind of data encryption on the frontend, that way the data is encrypted twice, once at each transfer of data.

Interoperability

Interoperability refers to a quality in an application where it consumes data from other services. For this quality I will analyze both the backend and the frontend of my application as they both utilize this quality in different ways.

One excellent way to improve interoperability is to use industry standards for implementing certain functionalities. This allows for these specific actions to be utilized on multiple or all platforms, opposed to methods which are only platform-specific. An example of this is my use of AWS's S3 APIs which are a standard that defines the interface that applications will use to retrieve, update and delete data from their Amazon cloud database. My backend utilizes specific APIs that the clients, or frontend, must access via http calls in order to modify elements in the database. This standard can be used by all mobile platforms, and this allows any device, platform independent, to access the same server and database using a similar method. Because this standard is used, theoretically any application on any platform could utilize my server. It wouldn't matter what kind of application it was; as long as it utilized a registration and login, than it would only need to access the APIs that were desired. It would simply need to have the ability to connect to a URL and post data via a multipart/form-data method; both of these are a cross-platform standard and any mobile platform should be able to do this.

There is no platform-independent way for my client application to communicate with other client applications. This is a very difficult thing to implement cross-platform, but it is also not relevant for the purposes of my application. What is possible is for the user to access his or her account from any mobile platform, and still have all his personal information and scores show up. The user can even use a computer to play the game and have these similar portability benefits.

As mentioned previously, I utilize the multipart/form-data for all POST requests made to the server. This is a standard which is widely used and accepted, and it helps to improve the interoperability of my application. I also utilize JSON objects for every data transfer and at each level. This includes communications between the frontend to the server, and between the database and the server, and vice-versa. This is a cross-platform standard as well. The one major improvement to my application could be the implementation of an oAuth system for authentication. I currently utilize the Node.js Passport module for my authentication which is specific to Node.js, but, as long as I don't plan on porting my backend server to another framework, I don't see where this would create much of a benefit.