Submitted by: Vaibhav Rakheja
Email ID: rakheja@ualberta.ca

Proof of Concept
Iversoft

1. **Hours spent to complete the project:** **22 hours**

Time Distribution:

- **MVC Architecture design & setup for services**
  - MVC Architecture setup for services — 4 hours
  - Implementing Unity web requests to generate generic http method templates — 4 hours
- **Logic - Data Consumption & Processing**
  - Logic and consumption of restful API and data processing — 6 hours
- **UI and Screen**
  - Login/Signup screen — 2 hours
  - Home and blog details screen — 2 hours
  - Create blog screen — 2 hours
  - Integration and testing — 2 hours

2. **Tasks completed, status report and highlights**

NOTE: The app is set up according to industry specific MVC architecture to consume the restful services

- **Feature - Loading Screen**
  - Displays animations, as per the requirement
  - The screen shows for at least 5 seconds and disappears when the API is returned
  - A button has been implemented to make the animations cycle through as per the requirement
- **Feature - Home Page**
  - Displays all published blogs in sequence of their creation date
  - Each post will display:
    - Title of the blog
    - Date created
    - Author name
    - Thumbnail image (if user has attached any) which will be downloaded from the server

NOTE: The posts are intractable and will display other details of the blog once the post is clicked on.

- **Feature - Login in Screen**
  - The login screen will collect the following information of a registered user:
    - username
    - password
  - The login details will be JSON serialized and will be sent as parameters to a post request
- **Feature - Signup Screen**
  - The signup screen will collect the following information of a new user:
    - Email ID
    - Password
    - Name
  - The signup details will be delivered to the server by calling a post API request
- **Feature - New Blog**
  - Users can create a new blog by clicking the new blog icon from the home screen
  - On clicking the new blog icon, a new blog dialog box will open and will capture the details of the blog
  - The blog details will be sent to the server using a post API request

- Only logged-in users can create new blog posts
- If the user is not logged-in, the app will prompt the user to login by redirecting the user to login in/sign up page

- **Feature - Blog Details**
  - Users can tap on any blog that is displayed on the home screen to view more details of the respective blog
  - The blog details screen opens with the thumbnail image referred from the home page
  - A simultaneous initiation to download a high quality image will be performed by the system and the quality will be updated automatically when a better quality image is available
- **Feature - Image Select from gallery and Blog Preview**
  - In place of a regular file select, image select from the gallery is integrated and will allow the user to select the image from their native gallery application
  - A preview of the image will be provided instantly on the create new blog screen
- **Feature - Pagination**
  - The application uses client-side pagination to generate data returned from the response on the UI in order to avoid performance issues.
  - Since the service to list all the blogs that have been published returns a list of all the blogs in one go, generating all the content on the user home screen not only causes performance issue, but also utilizes a lot of memory that can be easily saved by providing the user a fixed number of blogs to generate.
  - If the user wishes to scroll more to view additional content, additional fixed set of blogs would be generated on screen.
  - This process will be triggered automatically when the user is about to exhaust the current content (i.e. user is at the end of the page)
- **Feature - Responsive UI**
  - This is a responsive UI design application configured to be responsive to all portrait resolutions
  - The application employs MVC architecture of a rest client interface
- **Feature - Display Validation/Error Dialog Box**
  - Added a lean tween enabled dialog box that can show up messages like - No internet connection/ Incorrect Username Password/ Failed to publish new blog i.e. an animated notification to return API error/app validation errors
- **Feature - Pull to refresh**
  - The application will refresh the homepage content if the user is at the top of the page and pulls the scroll bar down.
  - This feature resembles the pull to refresh of any social media app like Instagram

NOTE: As per the requirement all the features are working

3. **Additional work that I would like to implement and estimated time** **17 hours**
4.

- **Feature - My Blogs & edit blogs** **4 hours**
  - Going through the API documentation, I recognized that there is an API that has been created to edit the blog post. I would like to implement this additional functionality.
  - The approach toward this problem would be to compare the current user authentication to the blog's data before allowing the user to edit the contents and other details related to a blog

- **Feature - Display Validation/Error Dialog Box** **3 hours**
  - Although I was able to add an Error Dialog box that appears when an error is returned via the API, I would like the error dialog box to display the exact error that is returned by the API instead of a generic response from the server
- **Functional Improvements**
  - **Object Pooling:** Object Pooling is a smart solution to accomplish the task of object template creation when the number of object to be generated inside the scene are calculated on runtime. High number of generated object templates cause performance issues in unity. This further causes the app to slow down and lag while generating frames. Object-pooling involves reusing the generated object template in order to avoid excessive memory computation. This allows the developer to cap the memory allocation and computation time spent on template object generation. In our case, though paginated, excessive scrolling would lead to a large number of blogs being spawned as individual objects. This can easily be curbed by using object pooling while reusing the instantiated object to display paginated blogs' details **4 hours**
  - **Interfaces for data models:** The current solution derives from a MVC architecture, where the data processing is done using a class structure approach. This solution is feasible for a small scale POC application (as the current task in hand), however, an even sophisticated solution using interfaces can be derived to provide a better stable MVC structure with secure data processing **4 hours**
  - **Efficient header management:** Due to time constraints, currently, the headers for the APIs are custom built when the API is about to be requested. However, a more sophisticated solution can be implemented to provide the developers quick access to headers and their consumption directly from the inspector. This will omit the need for other developers working on the same project to open the APIHandler class to modify/add headers according to the web request, while enabling them to accomplish this task directly via the inspector **2 hours**

5. Issues

Except the occasional disruption in API, I did not encounter any additional issues

- **Known bugs and issues:**
  - **Buggy Scroll:** Due to time limitations, I was unable to work on a smooth scroll mechanism. The scroll right now is functional, however, it lags and can be further polished.
  - **Generic error messages:** Due to time constraints, I was only able to implement a quick error display dialog-box to inform users about the incomplete web request. This can be further polished to show the exact error returned by the server

6. Wishlist of things that can be introduced in this project and time required **19 hours**

- **Feature - Edit User** **4 hours**
  - The documentation contains a service that allows us to edit the user details. That API can be consumed to provide a personalized page where the user has the ability to update the information provided
- **Feature - Reset password** **2 hours**
  - The documentation contains a service that allows the user to receive a password reset email. Email validation can be done in order to consume this API and provide the user with the ability to reset the password
- **Feature - Personal Blog List** **4 hours**

    - A service can be created which would provide a response comprising all the blogs that have been authored by the current logged in user. This can be used to create a page where all the blogs of the user can be displayed, just like a profile view
- **Feature - View Counter** **2 hours**
    - A service can be created to increment the view count of a blog. This service can be called every time the blog is paginated on the home screen
- **Feature - Like / Impression Counter** **2 hours**
    - A service can be created to increment the view like/impression/engagement counter of a blog. This service can be called every time the blog is loaded on the blog details screen
- **Feature - Player-Prefs and BinarySerialization** **2 hours**
    - A local copy of the user information can be stored in order to automatically attempt login and generate authentication token. This can be done by employing any one approach like player prefs, binary serialization or JSON serialization. Although BinarySerialisation provides much more security, since in this POC, we are already using JSON, JSONSerialization to save the user data would be appropriate
- **Feature - UINavigationManager** **3 hours**
    - A UINavigationManager can be setup. This will control the enabling and disabling of the screen and apply validation wherever required. The UINavigationManager can be called by passing the screen name as a parameter. The Manager will be responsible to update the screen and call the necessary methods when applicable. Since this being a POC, the number of UI screens are less and can be controlled individually, however, in case of a medium sized project, it may become a little hectic to control screen directly from the view controller/UI specific controllers and a NavigationManager class will be a handy class to accomplish the said task.

7. Instructions to use the app

- The development build of the **apk file is placed under the build folder in the main directory**. Install the apk in your device. You would need to provide **file access permission** to the application
- The application launches with Home Screen as the default screen. At first glance, the application will show some animations and friendly tips. A refresh button is provided on the home screen that can be used to cycle through the animations. This animation will be displayed for a minimum of 5 seconds as the API prepares the result (Fig 1.1)
- Once the results of the API (i.e. all blogs) have rendered on screen, home-screen will use the pagination approach to display content on the screen, generating new content every time the user reaches the bottom of the page
- The home screen has additional 3 buttons, Login/Signup (extreme left), Refresh Feed (middle) and the Create-A-New Blog (extreme right) (Fig 1.2)
- Clicking on the login button will redirect you to a Login Page. Clicking on the refresh feed button will refresh the home page to the latest feed. Clicking the Create new button will open the create new button page screen
- Login Page: The login page as the name describes, will provide the user ability to login into the account. The login page consists of 2 input fields and 3 buttons i.e. Login, Sign-Up, and Cancel (Fig 2.1). The login button sends a web request to receive an auth-token from the server. The Sign-up button opens a new screen (Fig 2.2) that allows the user to input sign-up information

like name, email and password. Both Login Screen and Sign-up screen, have a cancel button that will allow the user to return back to home screen without continuing the login/signup process

- Create Blog Screen: The create blog screen allows the user to post a new blog. The screen consists of 2 input fields, an open native gallery input to select images associated with the blog and 2 additional buttons (done and cancel) (Fig 3). The user can input the information and once satisfied, can click on the green tick button to submit the blog or can use the red cross button to cancel the blog and go back to the home page. The create screen will retain the input (draft) by the user in case he returns to the create blog screen again. **The create new blog operation is only allowed if a valid auth-token is provided.** In the absence of a valid auth-token, the screen will redirect to login/signup screen for the user to login/sign-up before posting a new blog. **A user is not allowed to post a blog if the user is not logged in.**
- Blog Details Screen: The Blog details screen provides the user with the detailed information of the blog (Fig 4). The blog details screen will use the thumbnail image provided to show a quick preview of the blog, and would simultaneously start downloading the high quality image. The image quality will update once a better version is available. Some of the blog details that are displayed are author name, title, image, blog content and views

8. Screenshots

Figure 1.1

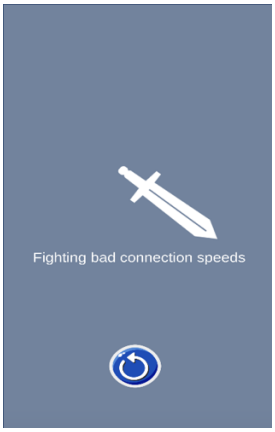

Figure 1.2

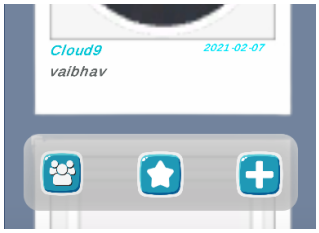

Figure 2.1

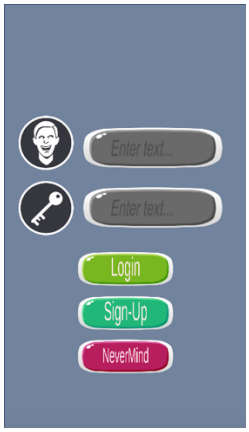

Figure 2.2

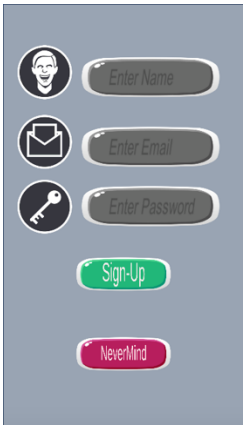

Figure 3

Figure 4