I. **Introduction**

This document is intended to help ease future developers for the Reality Flow mobile app into development process as smoothly as possible. This document is not exhaustive but should provide a succinct picture of what the mobile app looks like now as well as what the team was looking to add at the previous project's close.

II. How the App Works Now

a. Managers

Many elements of the mobile app make use of managers to populate and update the UI dynamically based on information from the server. For instance, the outliner manager will use the objects loaded from the server to create a list of those objects' names. Other managers are used to manipulate the tools in the scene. For example, the camera manager is what moves the camera when a user presses the "left" button.

Managers that populate some UI list, including the project list and outliner, have an accompanying prefab. This prefab is generally a panel with some layout element configurations and a button component to make the panel clickable. These prefabs also usually have scripts attached to hold some data associated with what that prefab is tracking. Each prefab's data is generally populated in that prefab's corresponding manager at runtime.

b. Runtime Gizmo & Selection

Selection is currently handled using a list maintained by the RuntimeGizmo package. The isSelected() function can be used to check if a given object is selected, which should be useful when implementing new ways to manipulate only the objects users have selected in a scene.

The GetTarget() function is used to detect clicks in the viewport. Note the first conditional (line 484 at the time of this writing) in GetTarget(). This conditional had to be modified so that clicks would not "go through" UI panels, but this conditional must be modified again when testing on a mobile device rather than in the editor. The following are the two variants of the last conditional:

```
!EventSystem.current.IsPointerOverGameObject()
!EventSystem.current.IsPointerOverGameObject(Input.GetTouch(0).fingerId)
```

The first conditional is the one that should be used when testing in the editor, while the second should be used when building for a mobile device.

c. Config & Persisting Data Between Scenes

Any data loaded from the server that must persist between scene loads is stored in config.cs, a static class. The default values seen in this file can be used to test whether data has been received from the server. In addition to the config, the project contains a script called DontDestroyOnLoad_RF. Attaching this script to an object will ensure that object persists between scene loads. This script is currently only used for the

FlowNetworkManager object.

III. Planned Features and Features in Development

a. Logout

The development branch for the mobile app currently has some support for logging out, but it's buggy. For this reason all logout buttons in the UI are currently disabled in the main branch. The DoLogout.cs script contains all code currently written to support this feature. For debugging, I recommend checking that config variables are being reset properly and that all objects created by requests sent and responses received from the server are destroyed. In addition, I would recommend working with the server developer(s) to ensure all connections are properly cleared on their end.

b. Creating Objects from Meshes

Support for creating objects from meshes currently exists in a limited form via the OBJ-IO plugin. This plugin allows .obj meshes to be loaded at runtime, which in conjunction with the MeshListManager allows objects to be instantiated using those same meshes. Currently users can only spawn objects using meshes already in the project files when the project is built.

The previous team's plan for this package was to use it in conjunction with a file hosting service, such as DropBox, to load .obj files into the project. This would allow users to spawn objects using those meshes hosted on these services. The viability of integrating such a file hosting service's API with OBJ-IO was not investigated thoroughly, however the example files included with OBJ-IO should give a good sense of how objects are loaded and exported to assess this.

The team was also able to create objects using the Google PolyToolKit, though spawning these objects is limited to the Unity editor plugin.

c. Project Creation, Inviting Users, and Joining Existing Projects

The send() and receive() functions for creating a new project already exist in the mobile client. Like logout, project creation in the mobile app proved to be buggy so all UI elements and scripts associated with project creation are either disabled or removed in the main branch. The user's project list also does not currently refresh when a new project has been associated with them. It would likely be helpful to first add a refresh button before attempting to implement project creation for smoother testing. The current mechanism for joining existing projects is to have a user in the Unity Editor manually add other users.

d. Textures

Limited support for textures currently exists on the server and was supported using the Unity plugin, but the implementation caused severe lag when moving textured objects. The team was advised during the demo that this could be alleviated by storing a reference rather than the image itself in the database and retrieving it using POST/GET

requests. This modification to the server would likely also entail modifications to the send() and receive() functions used to update objects, preferably to only load the texture when needed and reference a local copy when possible.

e. User Experience - Removing the Gizmos & More Camera Control
While the RuntimeGizmo package is convenient for proof of concept, the actual user experience with the manipulator widgets on mobile is poor. The sponsor and the team agreed that removing the widgets and having users manipulate objects by just tapping and dragging on the object itself would have a better feel. A good complement to this change, and even to the current implementation using RuntimeGizmos, would be the addition of more orthographic camera views. The app currently only supports a front view, but a side and top view are necessary to manipulate objects in a truly meaningful way.