Tutorial 4

Disclaimer: In your life as a software engineer, it will be important to know how to use third party software platforms and their materials. As such, **many of the materials we recommend for this course will be third party.** This will differ from some of your other programming courses, where you will need to write most of the software yourself. It is important to develop both sets of skills, that is, using third party materials as well as knowing how to write your own programs.

In this tutorial, we provide information on the final features you will need to know when making your roll call application. These are:

1. The Roll Call Server
2. Working together as a group to build your application

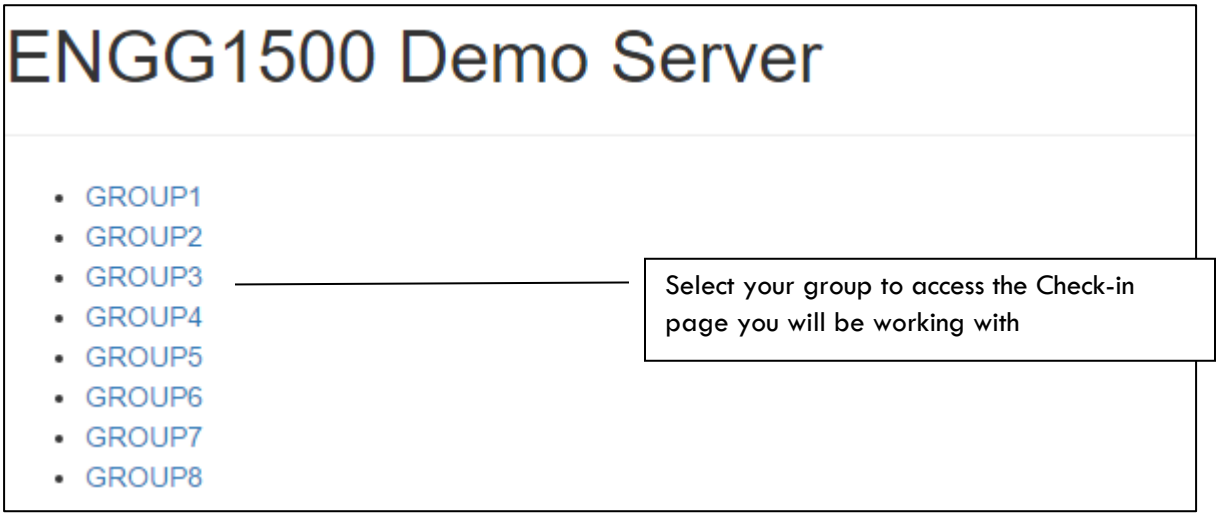
The Roll Call Server

A **server** is a program that will provide functionality to (a) client/s. In your project, you can think of the application you are building as a client, and the server as the program the client will need to communicate with.

The University of Newcastle has provided a simple server application which you can use for your lecture Check-in system. This server consists of a Home page and Check-in Listing page:

Home page

The home page for this server can be found **here**. From this page, each group can access their roll-call/Check-in page.

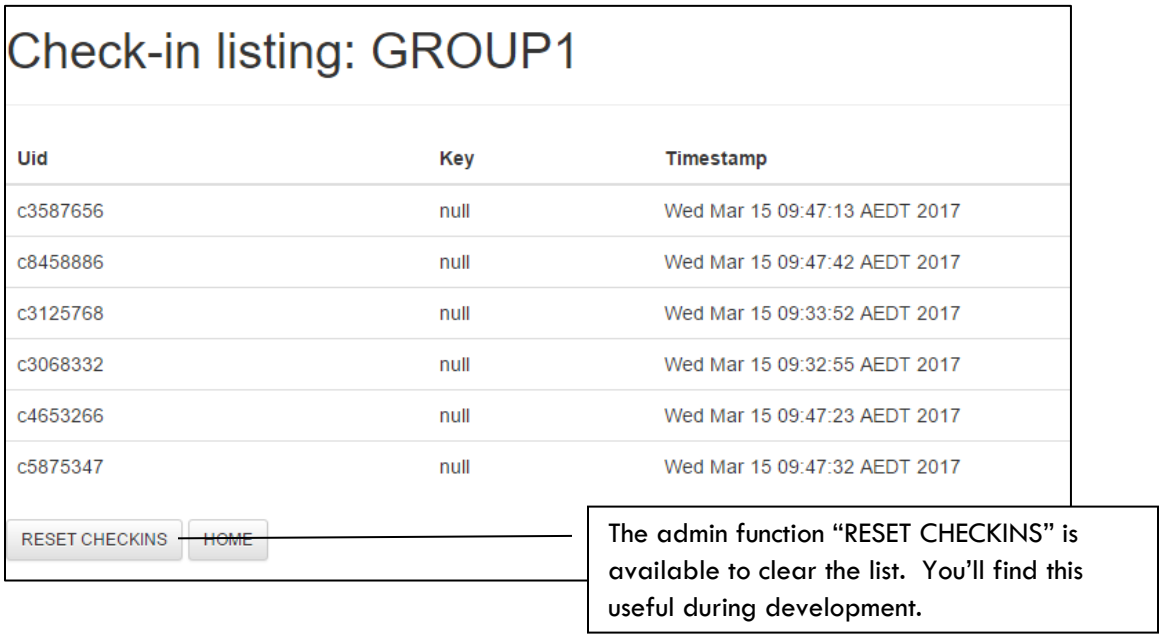


Check-in listing page

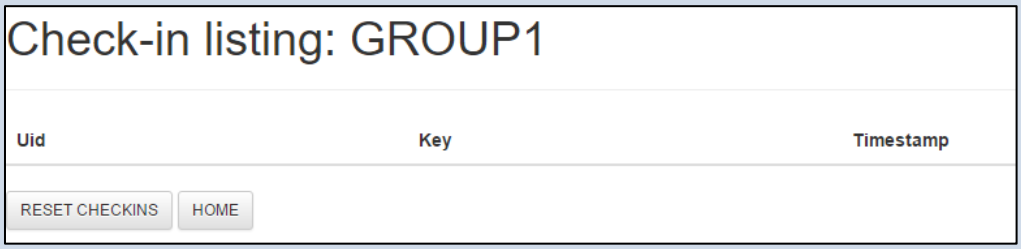
The Check-in listing page can be used to keep a list of students who have Checked-in from your application. This page will allow you up to 3 pieces of information; a unique identifier (UID), an optional key and a timestamp to note the time of Check-in.

Item	Description
UID	**Mandatory:** i.e. whenever a student Checks-in, the UID field MUST have an entry. UID's can be student numbers, or any other unique identifier (such as an IMEI) which you think will work for your application. How you choose to use the field is a design choice for you and your group members.
Key	**Optional:** i.e. whenever a student Checks-in, the Key field does not need to have an entry. This field can be used to supply any additional information which you think will be helpful in your system. For example, if you are logging student numbers as the UID, you may wish to let the student provide their name. OR if you are logging IMEI's as the UID, then you may wish to log an associated student number in the Key field. Don't feel restricted though, you can log anything you think might be useful, such as GPS co-ordinates, a scanned QR code, or whatever works for your application. Remember, this field is optional, so you do not need to use this field if it is not desired. If a value is not provided, this field will store "null"
Timestamp	**Auto-generated**: i.e. whenever the server successfully completes a Check-in action, the time and date it occurred will be recorded.

When using the client application on a number of phones, and storing information, the Check-in listing page might look something like this:



Note: The first time you view this list, it will be empty (as shown here). You will need to build your application so that you can add entries to the list. Read-on for more details about how to do this.



Communicating with the Server

Your client can send information to the server, to let the server know what to do. When the client tries to send information to the server, the server will send back a response. This response will provide information about what the server was (or was not) able to do.

The client will send the information to the server, in the form of a string. For example, here we have **string** which acts as a simple server command.

```
https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=checkin&group=GROUP1&uid=12345
```

This command will make a request to the sever for GROUP1, to log with UID "12345".

Anatomy of the server command

Let's break it down to see how this string works:

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=checkin&group=GROUP1&uid=12345

This part of the call lists details of the server you will connect to. That is, the server's URL.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/**API?**command=checkin&group=GROUP1&uid=12345

Next comes a statement that you will hit the server's API. An API is an **Application Programming Interface**. It's basically just the server's interface (or access point) to the outside world. This is what lets a client communicate with it.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=checkin**&group=GROUP1&uid=12345

Now, you can give the server a "command" for an action. There are two allowable actions for this server; "**query**" and "**checkin**". More information on these commands will be provided in the section below.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=**checkin**&group=GROUP1**&uid=12345

The server still needs to know some more information. For example, you will need to let the server know which group to log to. To do this, include the string "**&group=GROUP*n***" where n is a number between 1 and 8. Check with your tutor if you are not sure which group number to use.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=**checkin&group=GROUP1**&uid=12345**

The server will also want to know about the UID. To do this, include the string "**&uid=xxx**" where xxx is the details of the UID we wish to log/query.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=**checkin&group=GROUP1&uid=12345**&key=xxx**

Finally, extra information can be logged using the key field. To do this, add the string "**&key=xxx**" where xxx is the details of the key data to be logged.

Some interesting properties of the string:

1. The strings the server can accept are **case sensitive**. You will need to format each component of the string exactly as shown.
2. The string cannot contain spaces.
3. Notice that once the command component of the string has been given, the fields you are providing information for are separated by an "&". The "&" is being used as a character, to let the server know where once command ends, and the next begins.

Query, Check-in or Remove?

The server provides three core functions. These are "**query**", "**checkin**" and "**remove**". To use one of these functions, it just needs to be stated as the command in the string you send to the server.

For example:

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=query**&group=GROUP1&uid=12345

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=checkin**&group=GROUP1&uid=12345

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=remove**&group=GROUP1&uid=12345

While these strings look similar, they will do different things:

Query

The query command can be used to see if a UID already exists in the list. If the UID does **NOT** exist in the list, the server will provide the following response:

```
{
   "apiResponse": {
      "code": "SUCCESS"
      "response": "UID not found in checkin list."
   }
}
```

Note: This condition is considered a SUCCESS, because this lets you know that the IMEI/student has not yet logged to the server.

If the UID has already been logged to the list, then the server will provide the response:

```
{
   "apiResponse": {
      "code": "ERROR"
      "response": "UID found in checkin list."
   }
}
```

Note: This condition is considered an ERROR, because this lets us know that the IMEI/student/UID is already present in the list.

Note: The query command will only work for the UID. You cannot query the server based on the contents of the Key field.

Hint: when building the client application, it is logically correct to query the server before running the Check-in command. This is true even if the server is currently empty/has no Check-ins. Why? because the first time you run the query on an empty server, it will give you a "SUCCESS" response. This informs the application it can move onto the next step – Check-in.

Check-in

The Check-in command can be used to add details to the list held by the server. If the string is formatted correctly, and the request is sent to the server using "**command=checkin**", then the server will provide the response:

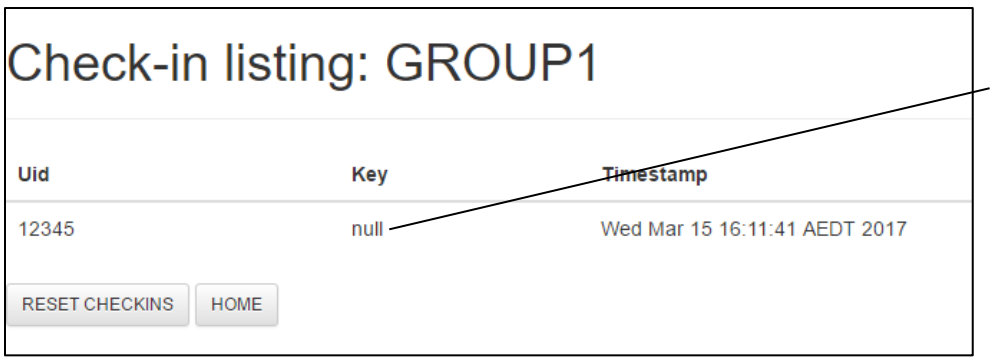
```
{
   "apiResponse": {
      "code": "SUCCESS"
      "response": "Check in successful."
   }
}
```

Note: This condition is considered a SUCCESS, to indicate that the checkin was successful.

Let's say the client used the following string to Check-in UID "12345" for GROUP1:

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=checkin**&group=GROUP1&uid=12345

The Check-in listing page would be updated with the following details:



The command listed didn't have a Key, so this field is listed with "null"

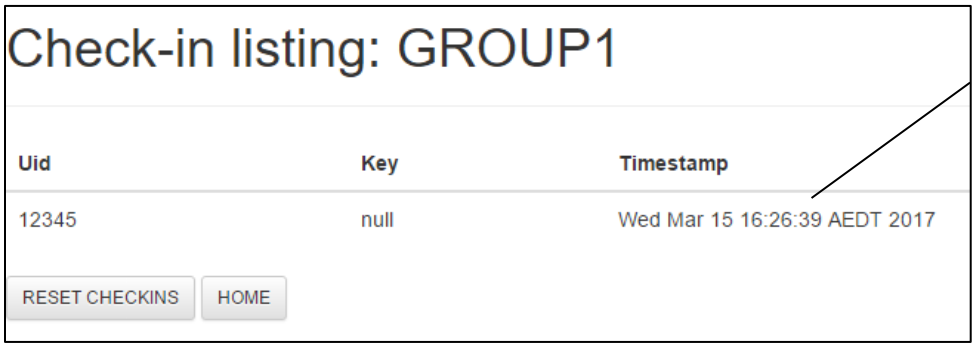
Note: The same details (same string) can be used to Check-in multiple times without error. That is, there is nothing on the server side to stop the client from making a successful command for Check-in using the same details more than once.

Hint: The client you design will need to use the query command to incorporate this logic.

So, just to be clear about the server, and how it will behave in this case: Let's assume the client sent the server the same command again:

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?**command=checkin**&group=GROUP1&uid=12345

When this occurs, the apiResponse code "SUCCESS" is returned to the client, and the timestamp listed for the associated UID will be updated.



The command to Check-in this UID has run twice, but there is only one entry for the data. The server simply keeps a record of the last attempted Check-in time.

Remove

Students who have Checked-in can be removed from the server using the remove command. If the string is formatted correctly, the request is sent to the server using "**command=remove**", and the UID correlates to an entry in the list, then the server will provide the response:

Let's say we try to remove an entry from the list using the following command.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=remove&group=GROUP1&uid=12345

If the UID was present in the list, the server would provide the following response

```
{
   "apiResponse": {
      "code": "SUCCESS"
      "response": "UID removed from check-in list"
   }
}
```

Note: This condition is considered a SUCCESS, to indicate that the removal was successful.

However, if the student was not found in the list, the following response would be provided.

```
{
   "apiResponse": {
      "code": "ERROR"
      "response": "UID not found in list"
   }
}
```

Note: This condition is considered an ERROR, to indicate the entry could not be removed.

Finally, the list can be cleared using a command from your application. This is completed by using the remove command, in conjunction with a **Kleene star** wildcard for the UID.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=remove&group=GROUP1&uid=*


```
{
   "apiResponse": {
      "code": "SUCCESS"
      "response": "The list has been cleared"
   }
}
```


Incorrect String

If the string provided when trying to Check-in is incorrect, the server will return an error message. The response can be used to help define what is incorrect with the string.

Error responses for an incorrect string can be one of the following:

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=checkin&**group=GROUP10**&uid=12345


```
{
   "apiResponse": {
      "code": "ERROR"
      "response": "Unable to access group data."
   }
}
```


Using a group ID which does not exist.

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=checkin&uid=12345


```
{
   "apiResponse": {
      "code": "ERROR"
      "response": "Group is a mandatory parameter."
   }
}
```


Did not provide a reference for group

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=checkin&group=GROUP1


```
{
   "apiResponse": {
      "code": "ERROR"
      "response": "UID is a mandatory parameter."
   }
}
```


Did not provide a reference for UID

https://engg1500.newcastle.edu.au/ENGG1500DemoServer/API?command=checkin

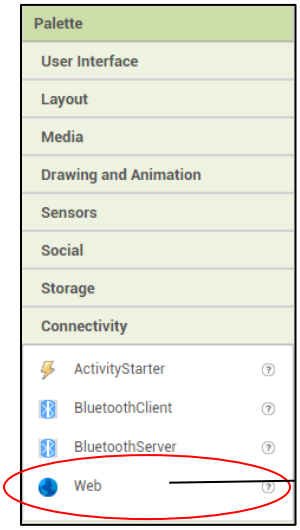

```
{
   "apiResponse": {
      "code": "ERROR"
      "response": "Group is a mandatory parameter. UID is a mandatory parameter."
   }
}
```


Did not provide a reference for group or UID

Sending Information to the Server using MIT App Inventor

This sever is on the cloud, so your android application will be able to communicate with the sever using the **web component** in the MIT app inventor platform.

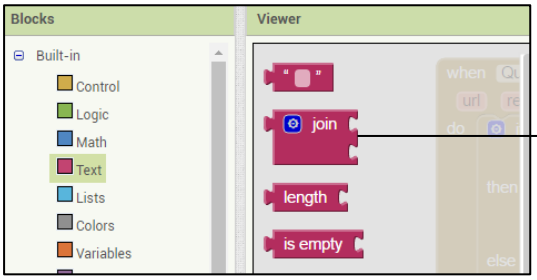
This component is found under "Connectivity" in the palette:



drag and drop the web-component into the viewer window on the designer page.

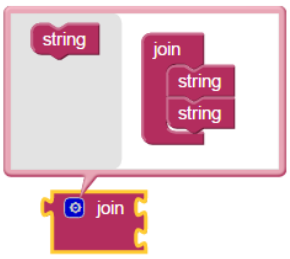
The information will be sent, by creating a **string** which will act as the command for the server

Hint: To build the string, you will need to use the "join" text function. This can be found in the built-in components of the blocks editor:



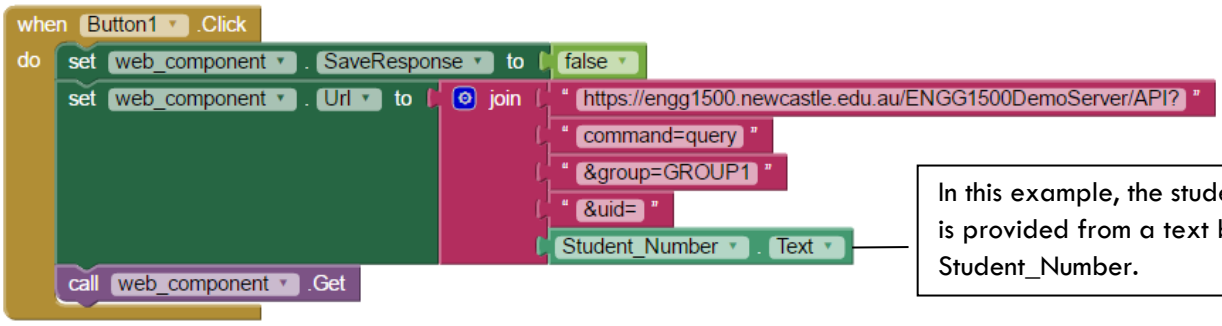
Use the join function to build the strings that will communicate with the server.

If you need to join more than 2 text pieces, you can do this by clicking on the blue settings wheel and adding a place for any more strings.



Okay – so now what?

To make calls to the server use the text join function to build the string you will be sending. This string needs to be set as the URL for the web component. When the string is set, you can use the web_component.Get method to make a call to the server. For example:

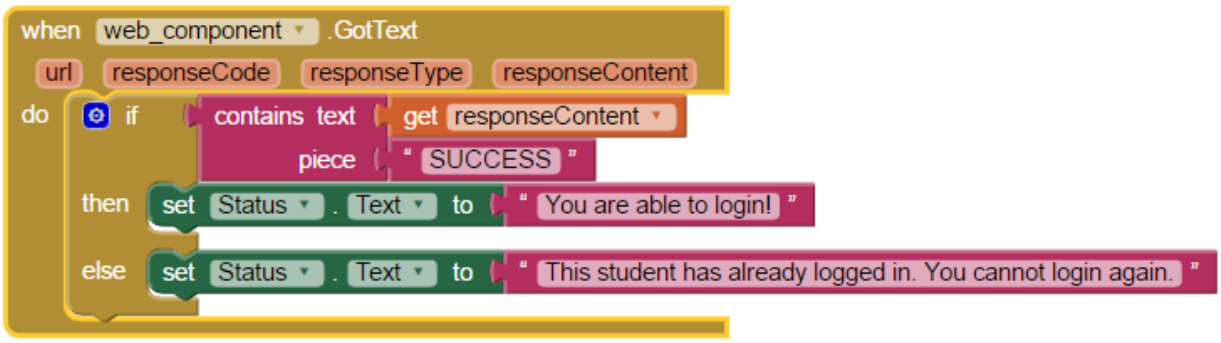


In this example, the student number is provided from a text box called Student_Number.

When button1 is clicked, the web_componenet SaveResponse field is set to false; this will ensure you can use the web_components "GotText" event later in your program.

Once the string is built, you can set it as the web_components URL, then use the web_component.Get function to send the string to the server.

Why do we use a Get request if we are sending info to the server?
The server is going to provide a response to the application you are building, that is, your web_component will be "getting" a response from the server. By using an HTTP get request, you can trigger a "GotText" event, which will hold the server's response as a local variable. Using this response, the client can determine what action will happen next in the application.



When the GotText event is activated, you will have "responseContent" from the server. You can use the response content to see if the query was successful, and update the program with an appropriate message.

Note, if the response was SUCCESSFUL, then you would want to follow up by sending the Check-in command to the server.

Anything Else?

There is one last thing you need to consider when working with the server.

Because the web component calls are asynchronous (this is something you will learn about in SENG3400), **it is recommended that your application uses at least two web components.** One component will be for Querying, while the other will be for Check-in.

Have a go yourself!!!

Build an application, which uses a web component to run the Check-in command. It can log either an IMEI or student number to the server.

1. Add a Check-in button to the application. When this button is clicked, build the string which will be sent to the server.
2. Add another web component which can be used to run the Query command. Add a query button, so that when the button is clicked, the server will be queried about the existence of specific data.

   Hint: for each web component, use the GotText event and ensure it has a conditional statement, to let you know if the call was successful or not.

   Hint: if using the student number, you may wish to include some validation that the student number is correct (for example, check the length of the student number, check that the student number starts with a lowercase c, etc).

Working together as a group

There is a limitation in the MIT App Inventor tool; it is not a collaborative tool in the same way google docs or another typically collaborative tool might be.

But don't be disheartened, in your life as an engineer, you will often have to find solutions to make something fit for purpose. This is sometimes done by using policies. A policy is just a workflow (or defined set of activities) to help make working together easier.

Why is this important?

Because the MIT App Inventor tool is not collaborative, it is possible you could experience syncing issues, missing features or lost work if more than one person wants to work on the same application at the same time.

Sample Policies

We will outline 3 sample policies you could use to to manage the build of your application. For each policy, we will explore the pro's and con's, as there will some key factors to watch out for with each method.

It will be up to your group to work out your own policy so that there is one master application being developed safely (without loss of data).

Policy 1

Assign a group member as the "**developer**" who will be responsible to update the application and perhaps distribute the .apk or .aia files to other group members.

Pros	This keeps development centralized so there is less likely to be any chance of syncing issues
Cons	Only one person is the "developer" Is there a way around this? Perhaps you can change who is in this role each week. Things to consider if you are changing who is the developer: How would you switch developer's week to week? You could simply pass the .aia file to the next person so they can load it to their environment? Or you could keep the .aia file somewhere central so that the whole group has access to it (for example; dropbox, or google drive)

Policy 2

Build the application using the google admin account provided with your Uni phone. Everyone in your group should be able to use this account to easily access the development environment and work on the application.

Pros	Only one account is necessary and there is no need for team members to load/redistribute .aia files
Cons	Even though there might be one account, you cannot have two people making changes to the same project at the same time. This is because, two peopled logged in from different computers, but using the same account can cause syncing issues, and it is possible that your development project will become irreparably damaged. So what can you do instead? Perhaps you can setup some kind of communication standard, such as "**add a post to the Facebook page - when you wish to work on the application**". That way, everyone should be able to know who is working on the application and when. Note that this will work best if you include the obligatory "**add a post to the Facebook page - when you finish working on the application**", so that your group members know when they can login to work on the application again.

Policy 3

Use feature driven development. In this scenario, everyone works on a small feature using their local environment. As a feature is complete, the associated .aia file can be stored in a central repository (google drive, dropbox, etc) so that group members can load features easily for editing.

Pros	Anyone in the group can load and distribute a feature as they please, with little chance of syncing issues.
Cons	At some point, the features will need to be merged into one application, which will require the use of something like policy 1 or policy 2.