ITIS/CS 4180 – Mobile App Development
Midterm Makeup

Basic Instructions:

1. This is the Midterm Exam Makeup, which will count for 20% of the total course grade.
2. This Midterm is an individual effort. Each student is responsible for her/his own Midterm and its submission.
3. During the exam, you are allowed to use the course videos, slides, and your code from previous home works and in class assignments. You can use the internet to search for answers. You are NOT allowed to use code provided by other students or solicit help from other online persons.
4. Answer all the exam parts, all the parts are required.
5. Please download the support files provided with the Midterm and use them when implementing your project.
6. Your assignment will be graded for functional requirements and efficiency of your submitted solution. You will loose points if your code is not efficient, does unnecessary processing or blocks the UI thread.
7. Create a zip file which includes all the project folder, any required libraries, and your presentation material. Submit the exported file using the provided canvas submission link.
8. **Do not try to use any Social Messenger apps, Emails, Or Cloud File Storage services in this exam.**
9. **Failure to follow the above instructions will result in point deductions.**
10. **Any violation of the rules regarding consultation with others will not be tolerated and will result disciplinary action and failing the course.**

Midterm Makeup (100 Points)

In this assignment you will get familiar with using with HTTP connections, JSON parsing, Authorization, and implement a message sharing app where messages are organized into message threads.

API Descriptions

Some APIs will require authorization using an Authorization Token. The signup or login APIs will return a token that should be passed as a header parameter to all the following requests that require authorization. You are provided with a skeleton project file that demonstrates how Authorization can be performed by passing an Authorization header parameter using the Alamofire library. In addition you are provided with a Postman file.

Name:	**Login API**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/login
Method:	POST
Needs Authorization:	NO
Parameters:	- email - password

Name:	**Signup API**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/signup
Method:	POST
Needs Authorization:	NO
Parameters:	- email - password - fname (user first name) - lname (user last name)

Name:	**Get all the threads**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/threads
Method:	GET
Needs Authorization:	YES (Need to pass Authorization header parameter as BEARER Token)
Parameters:	NONE

Name:	**Add a new thread**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/thread/add
Method:	POST
Needs Authorization:	YES (Need to pass Authorization header parameter as BEARER Token)
Parameters:	- title (the title of the new thread)

Name:	**Delete a thread**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/thread/delete/{thread_id}
Method:	GET
Needs Authorization:	YES (Need to pass Authorization header parameter as BEARER Token)
Parameters:	- NONE, but note that the thread_id is included as part of the url.

Name:	**Get all messages in a thread**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/messages/{thread_id}
Method:	GET
Needs Authorization:	YES (Need to pass Authorization header parameter as BEARER Token)
Parameters:	- NONE, but note that the thread_id is included as part of the url.

Name:	**Add a new message to a thread**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/message/add
Method:	POST
Needs Authorization:	YES (Need to pass Authorization header parameter as BEARER Token)
Parameters:	- thread_id - message (the message text)

Name:	**Delete a message**
EndPoint:	http://ec2-18-234-222-229.compute-1.amazonaws.com/api/message/delete/{message_id}
Method:	GET
Needs Authorization:	YES (Need to pass Authorization header parameter as BEARER Token)
Parameters:	- NONE, but note that the message_id is included as part of the url.

Part 1: Login (10 points)

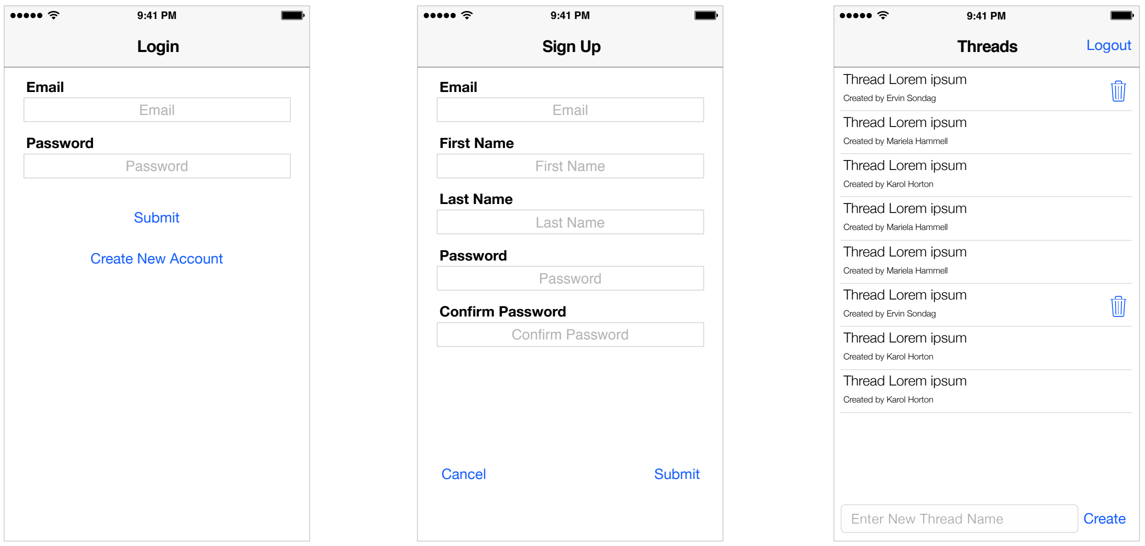
This is the launcher screen the app show in Fig 1(a). The requirements are as follows:

1. The user should provide an email and password. The provided credentials should be used to authenticate the user by using Login API. Clicking the “Login” button should submit the login information to the api to verify the user’s credentials.
   a) If the login API is successful, then store the required user information, and authorization token. Then start the Threads List Screen shown in Figure 1(c) and dismiss the Login Screen.
   b) If the user is not successfully logged in, then show an alert message indicating that the login was not successful.
2. Clicking the “Create New Account” button should show the Signup Screen Fig 1(b).

Part 2: SignUp (10 points)

Create the Signup screen shown in Figure 1(b), with the following requirements:

1. Clicking the “Cancel” button should dismiss this Screen and show the Login Screen.
2. The user should provide their first name, last name, email, password and password confirmation. Perform the required validation (the given password and the confirm password match). Clicking the “Sign Up” button should submit the user’s information to the Signup API.
   a) If the signup API is not successful display an alert showing an error message indicating the error message received from the api.
   b) If the signup API is successful, then store the required user information, and authorization token. Then start the Threads List Screen shown in Figure 1(c) and dismiss the Signup screen.



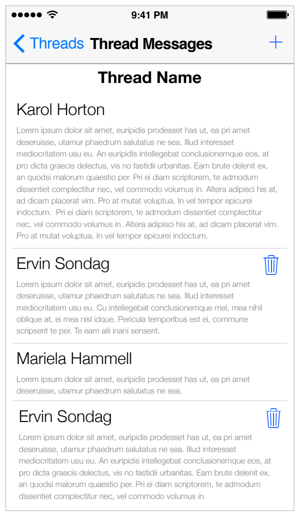
(a) Login (b) Sign Up (c) Threads List

Figure 1, Application Wireframe

Part 3: Threads List (30 points)

This screen shows the list of threads created by all the users in the system and enables the user to interact with these threads see Figure 1(c), The requirements are as follows:

1. Clicking the "Logout" navigation bar button should logout the currently logged in user, dismiss this view controller and show the Login view controller.
2. The app should contact the "Get all the threads" API to retrieve the list of threads and should display the threads in a TableView as shown in Figure 1(c).
   a. Each cell should display the name of the thread, and the creator of the thread.
   b. The trash icon should only be displayed for thread items that were created by the currently logged in user. For example, in Figure 1(c) the currently logged in user is "Ervin Sondag" who is also the creator of the first thread.
   c. If the user clicks the trash icon the app should present an alert asking the user if the selected thread should be deleted. If the user answers "OK" then the selected thread should be deleted by contacting the "Delete a thread" API.
      - After deleting a thread item, the "Get all the threads" API should be used to retrieve the latest list of threads and the list should be refreshed with the retrieved threads.
3. Clicking on a row item should segue to the Thread Messages view controller shown in Figure 2(a).
4. If the user enters a new thread name and clicks the "Create" button, the "Add a new thread" API should be used to add the new thread.
   a. After adding the new thread, the "Get all the threads" API should be used to retrieve the latest list of threads and the list should be refreshed with the retrieved threads.



Thread Name

Karol Horton

Lorem ipsum dolor sit amet, euripidis prodesset has ut, ea pri amet deseruisse, utamur phaedrum salutatus ne sea. Illud interesset mediocritatem usu eu. An euripidis intellegebat conclusionemque eos, at pro dicta graecis delectus, vis no fastidii urbanitas. Eam brute delenit ex, an quodsi malorum quaestio per. Pri ei diam scriptorem, te admodum dissentiet complectitur nec, vel commodo volumus in. Altera adipisci his at, ad dicam placerat vim. Pro at mutat voluptua. In vel tempor epicurei indoctum. Pri ei diam scriptorem, te admodum dissentiet complectitur nec, vel commodo volumus in. Altera adipisci his at, ad dicam placerat vim. Pro at mutat voluptua. In vel tempor epicurei indoctum.

Ervin Sondag

Lorem ipsum dolor sit amet, euripidis prodesset has ut, ea pri amet deseruisse, utamur phaedrum salutatus ne sea. Illud interesset mediocritatem usu eu. Cu intellegebat conclusionemque mel, mea nihil oblique at, ei mea nisl idque. Pericula temporibus est ei, commune scripserit te per. Te eam alii inani senserit.

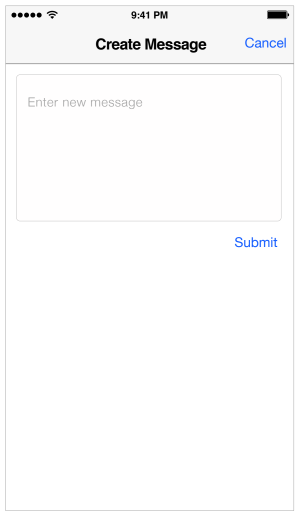
Mariela Hammell

Lorem ipsum dolor sit amet, euripidis prodesset has ut, ea pri amet deseruisse, utamur phaedrum salutatus ne sea.

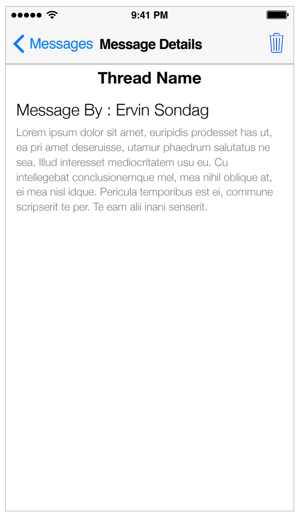
Ervin Sondag

Lorem ipsum dolor sit amet, euripidis prodesset has ut, ea pri amet deseruisse, utamur phaedrum salutatus ne sea. Illud interesset mediocritatem usu eu. An euripidis intellegebat conclusionemque eos, at pro dicta graecis delectus, vis no fastidii urbanitas. Eam brute delenit ex, an quodsi malorum quaestio per. Pri ei diam scriptorem, te admodum dissentiet complectitur nec, vel commodo volumus in.

(a) Thread Messages



(b) Create Message



Thread Name

Message By : Ervin Sondag

Lorem ipsum dolor sit amet, euripidis prodesset has ut, ea pri amet deseruisse, utamur phaedrum salutatus ne sea. Illud interesset mediocritatem usu eu. Cu intellegebat conclusionemque mel, mea nihil oblique at, ei mea nisl idque. Pericula temporibus est ei, commune scripserit te per. Te eam alii inani senserit.

(c) Message Details

Figure 2, Application Wireframe

Part 4: Thread Messages (30 points)

This ViewController displays a list of thread messages for the selected thread as shown in Figure 2(a), The requirements are as follows:

1. The selected thread name should be displayed at the top of the view controller as shown in Figure 2(a).
2. The view controller should contact the “Get all messages in a thread” API to retrieve the messages included in the selected thread and should display the messages in a TableView as shown in Figure 2(a).
   a. Each thread message cell should display the name of the message creator and the full message text. Note: the cells should resize automatically to display the full message text.
   b. The trash icon should only be displayed beside message items that were created by the currently logged in user.
   c. If the user clicks the trash icon the app should present an alert asking the user if the selected message should be deleted. If the user answers “OK” then the forum should be deleted by contacting the “Delete a message” API.
      - After deleting a message item, the "Get all messages in a thread” API should be used to retrieve the latest list of messages and the list should be refreshed with the retrieved messages.
3. Clicking on a row item should segue to the Message Details view controller shown in Figure 2(c) to show the message details.
4. Clicking on the “+” navigation bar button should present the “Create Message” view controller modally as shown in Figure 2(b).

Part 5: Create Message (10 points)

This ViewController enables the user to create a new message in the selected thread shown in Figure 2(b), The requirements are as follows:

1. If the user enters the message text, then clicks the “Submit” button, the app should use the “Add a new message to a thread” API to create the new message.
   a. Upon successfully creating the new message, this view controller should signal the Thread Messages view controller to reload the message list with the latest list of messages. Then this view controller should be dismissed.
2. Clicking the “Cancel” button should dismiss the view controller.

Part 6: Message Details (10 points)

This ViewController enables the user to view the message details as shown in Fig 2(c), The requirements are as follows:

1. The view controller should display the thread name, name of the creator of the message and the full message.
2. The trash navigation bar button should only be displayed if the currently logged in user has created the displayed message.
3. If the user clicks the trash navigation bar button the app should present an alert asking the user if the selected message should be deleted. If the user answers “OK” then the message should be deleted by contacting the “Delete a message” API.
   a. Upon successfully deleting the message using the API, this view controller should signal the Thread Messages view controller to reload the message list with the latest list of messages. Then this view controller should be dismissed.