



## Assignment 3 Final Report

**Milestone 0:** Describe the problem that your application solves. (Not graded)

Planning modules is a pain. Planning meetings with your friends is even more painful. mods+ solves this via a combined timetable view with all your friends' timetable, allowing your team to easily find a free time for meetings.

**Milestone 1:** Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?

Timetable planner that has Facebook login functionality to persist data across devices. In this way, if you plan your timetable on your phone, all you need to do is login on our application on your desktop to view the exact same timetable on your desktop.

Our application also allows users to discover classmates with common lessons as long as they also use the application. The team functionality allows users to view combined timetables and can be user to plan for meetings during common free time.

A mobile cloud application makes sense because the data has to be persisted to support the classmate discovery feature and combined timetable view.

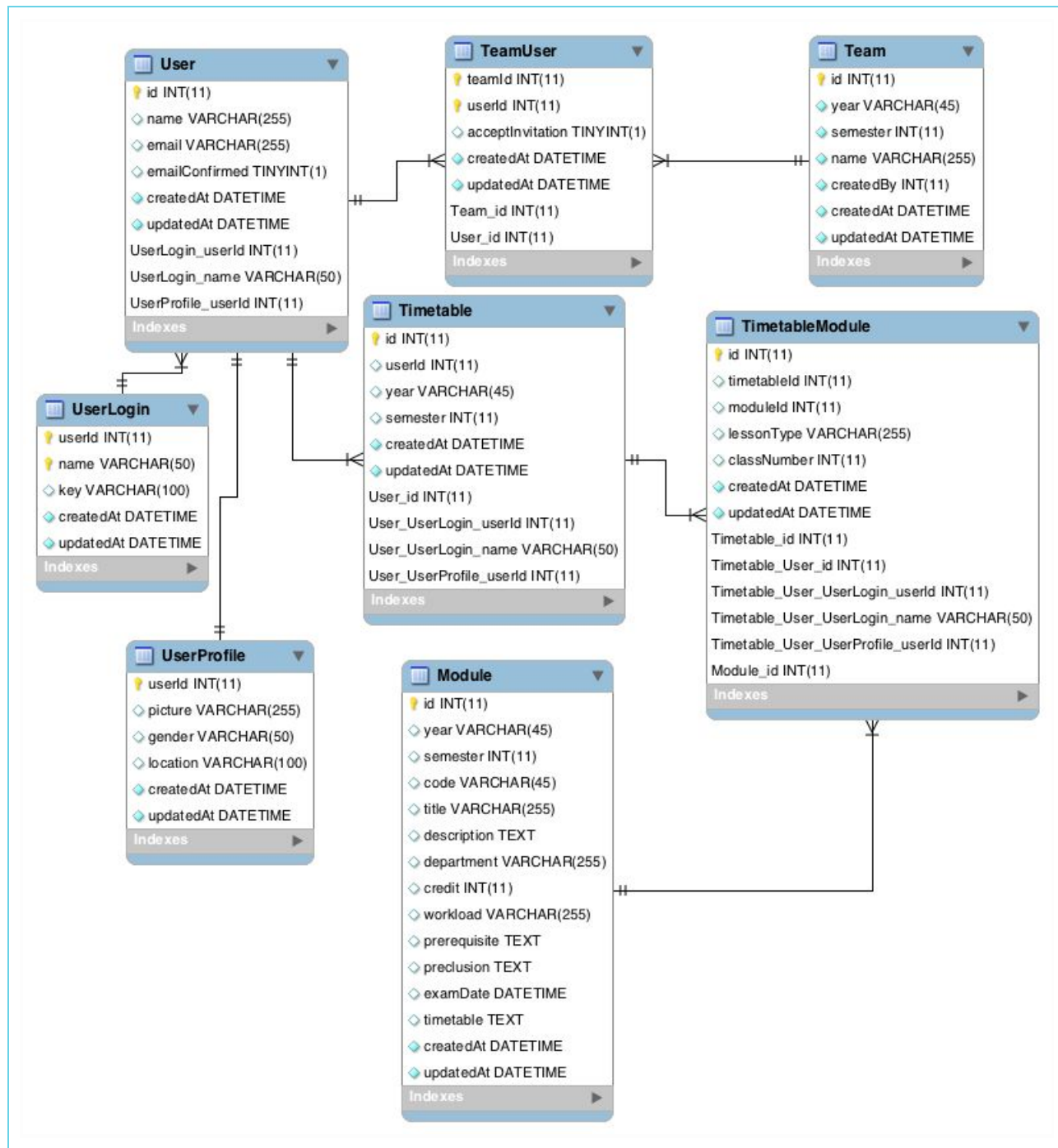
**Milestone 2:** Describe your target users. Explain how you plan to promote your application to attract your target users.

Initially, we will target students in NUS. This is because NUS students are already using nusmods.com and are familiar with the functionality that it provides.

Eventually, depending on an institution's api allowances, we aim to promote our application to students in tertiary education by extending our application to more schools.

We can request the nusmods team to redirect a portion of users from nusmods.com (which already has tens of thousands of users) to beta.nusmods.com.

**Milestone 3:** Draw the database schema of your application.



**Milestone 4:** Design and document all your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief explanation on the purpose of each request for reference. Also, **explain how your API conforms to the REST principles and why you have chosen to ignore certain practices (if any.)**

You can explore our API here: <http://docs.momomods.apiary.io/#>

**Milestone 5:** Share with us some queries (at least 3) in your application that require database access. Provide the actual SQL queries you use (if you are using an ORM, find out the underlying query) and explain how it works.

API 1: /api/team/1

```
SELECT
`Team`.`id`, `Team`.`year`, `Team`.`semester`, `Team`.`name`,
`Team`.`createdBy`, `Team`.`createdAt`, `Team`.`updatedAt`, `users`.`teamId`
AS `users.teamId`, `users`.`userId` AS `users.userId`, `users`.`acceptInvitation` AS
`users.acceptInvitation`, `users`.`createdAt` AS `users.createdAt`,
`users`.`updatedAt` AS `users.updatedAt`, `users.user`.`id` AS `users.user.id`,
`users.user`.`name` AS `users.user.name`, `users.user`.`email` AS
`users.user.email`, `users.user`.`emailConfirmed` AS `users.user.emailConfirmed`,
`users.user`.`createdAt` AS `users.user.createdAt`, `users.user`.`updatedAt` AS
`users.user.updatedAt`, `creator`.`id` AS `creator.id`, `creator`.`name` AS
`creator.name`, `creator`.`email` AS `creator.email`, `creator`.`emailConfirmed`
AS `creator.emailConfirmed`, `creator`.`createdAt` AS `creator.createdAt`,
`creator`.`updatedAt` AS `creator.updatedAt`
FROM `Team` AS `Team`
LEFT OUTER JOIN `TeamUser` AS `users`
ON `Team`.`id` = `users`.`teamId`
LEFT OUTER JOIN `User` AS `users.user`
ON `users`.`userId` = `users.user`.`id`
LEFT OUTER JOIN `User` AS `creator`
ON `Team`.`createdBy` = `creator`.`id`
WHERE `Team`.`id` = '1';
```

This query finds all team members' information as well as the team creator's information. There are 3 main tables used in this query: Team, User and TeamUser. Users can have many teams and teams can have many users. Hence, TeamUser is a junction table. We first query Team based on Team.id = 1 and then left outer join TeamUser where Team.id = TeamUser.teamId. We then left outer join User where TeamUser.userId = User.id. In this way, we have found all users' information (such as name) based on a team's id.

API 2: /api/2016-2017/1/timetable

```
SELECT `Timetable`.*, `timetableModules`.`id` AS `timetableModules.id`,
`timetableModules`.`timetableId` AS `timetableModules.timetableId`,
`timetableModules`.`moduleId` AS `timetableModules.moduleId`,
`timetableModules`.`lessonType` AS `timetableModules.lessonType`,
`timetableModules`.`classNumber` AS `timetableModules.classNumber`,
`timetableModules`.`createdAt` AS `timetableModules.createdAt`,
`timetableModules`.`updatedAt` AS `timetableModules.updatedAt`,
`timetableModules.module`.`id` AS `timetableModules.module.id`,
`timetableModules.module`.`year` AS `timetableModules.module.year`,
`timetableModules.module`.`semester` AS `timetableModules.module.semester`,
`timetableModules.module`.`code` AS `timetableModules.module.code`,
`timetableModules.module`.`title` AS `timetableModules.module.title`,
```

```

`timetableModules.module`.`description` AS
`timetableModules.module.description`, `timetableModules.module`.`department`
AS `timetableModules.module.department`, `timetableModules.module`.`credit` AS
`timetableModules.module.credit`, `timetableModules.module`.`workload` AS
`timetableModules.module.workload`, `timetableModules.module`.`prerequisite` AS
`timetableModules.module.prerequisite`, `timetableModules.module`.`preclusion`
AS `timetableModules.module.preclusion`, `timetableModules.module`.`examDate`
AS `timetableModules.module.examDate`, `timetableModules.module`.`timetable`
AS `timetableModules.module.timetable`, `timetableModules.module`.`createdAt`
AS `timetableModules.module.createdAt`, `timetableModules.module`.`updatedAt`
AS `timetableModules.module.updatedAt`
FROM
(SELECT `Timetable`.`id`, `Timetable`.`userId`, `Timetable`.`year`,
`Timetable`.`semester`, `Timetable`.`createdAt`, `Timetable`.`updatedAt` FROM
`Timetable` AS `Timetable`
WHERE `Timetable`.`userId` = 2
AND `Timetable`.`year` = '2016-2017'
AND `Timetable`.`semester` = '1' LIMIT 1)
AS `Timetable`
LEFT OUTER JOIN `TimetableModule` AS `timetableModules`
ON `Timetable`.`id` = `timetableModules`.`timetableId`
LEFT OUTER JOIN `Module` AS `timetableModules.module`
ON `timetableModules`.`moduleId` = `timetableModules.module`.`id`;

```

This query finds all modules' information based on a certain timetable. Each timetable row has a unique composite key that consists of year, semester and userId. That is, each user should only have one timetable for each semester of each year. Hence, we first select this exact timetable row. We then left outer join TimetableModule to get all modules that are contained in this timetable. These modules are then left joined with the corresponding module to get module details (such as description, exam date etc.).

API 3: /api/2016-2017/1/friends

```

SELECT `Timetable`.*, `timetableModules`.`id` AS `timetableModules.id`,
`timetableModules`.`timetableId` AS `timetableModules.timetableId`,
`timetableModules`.`moduleId` AS `timetableModules.moduleId`,
`timetableModules`.`lessonType` AS `timetableModules.lessonType`,
`timetableModules`.`classNumber` AS `timetableModules.classNumber`,
`timetableModules`.`createdAt` AS `timetableModules.createdAt`,
`timetableModules`.`updatedAt` AS `timetableModules.updatedAt`
FROM (SELECT `Timetable`.`id`, `Timetable`.`userId`, `Timetable`.`year`,
`Timetable`.`semester`, `Timetable`.`createdAt`, `Timetable`.`updatedAt`
FROM `Timetable` AS `Timetable`
WHERE `Timetable`.`userId` = 2
AND `Timetable`.`year` = '2016-2017'
AND `Timetable`.`semester` = '1'
LIMIT 1) AS `Timetable`
LEFT OUTER JOIN `TimetableModule` AS `timetableModules`
ON `Timetable`.`id` = `timetableModules`.`timetableId`;
-----
SELECT `Module`.`id`, `timetableModules`.`id` AS `timetableModules.id`,

```

```

`timetableModules`.`timetableId` AS `timetableModules.timetableId`,
`timetableModules`.`moduleId` AS `timetableModules.moduleId`,
`timetableModules`.`lessonType` AS `timetableModules.lessonType`,
`timetableModules`.`classNumber` AS `timetableModules.classNumber`,
`timetableModules`.`createdAt` AS `timetableModules.createdAt`,
`timetableModules`.`updatedAt` AS `timetableModules.updatedAt`,
`timetableModules.timetable`.`id` AS `timetableModules.timetable.id`,
`timetableModules.timetable`.`userId` AS `timetableModules.timetable.userId`,
`timetableModules.timetable`.`year` AS `timetableModules.timetable.year`,
`timetableModules.timetable`.`semester` AS
`timetableModules.timetable.semester`, `timetableModules.timetable`.`createdAt`
AS `timetableModules.timetable.createdAt`,
`timetableModules.timetable`.`updatedAt` AS
`timetableModules.timetable.updatedAt`
FROM `Module` AS `Module`
LEFT OUTER JOIN `T timetableModule` AS `timetableModules`
ON `Module`.`id` = `timetableModules`.`moduleId`
LEFT OUTER JOIN `T timetable` AS `timetableModules.timetable`
ON `timetableModules`.`timetableId` = `timetableModules.timetable`.`id`
WHERE `Module`.`id` IN (514, 549, 559, 565, 1373, 2325);
-----
SELECT `id`, `name` FROM `User` AS `User` WHERE `User`.`id` IN (1, 2);

```

For this API, we want to get all users of our application who are taking at least one of the modules that the current user is taking.

E.g.:

If user A is taking module X and Y,  
user B is taking module X and Z  
and user C is taking module W and Z,  
then user A is friends with user B but not friends with user C.

Our first query gets all the moduleIds that the user is taking in the current year and semester using his timetable for the current year and semester. This is similar to the `/api/2016-2017/1/timetable` end point besides the fact that we do not need to outer left join with Module as we do not need the modules' details. Instead, we only need the moduleIds.

We then add all these moduleIds into an array. Our next query finds all timetables which contain at least one of the modules in the array. In this second query, we are able to get all the userIds that take at least one of the modules that the current user is taking.

Our final query gets the corresponding names of all these userIds.

**Milestone 6:** Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly. Include an image of the icon and a screenshot of the splash screen in your writeup. If you did not implement a splash screen, justify your decision with a short paragraph. Add your application to the home screen to make sure that they are working properly. Make sure at least Safari on iOS and Chrome on Android are supported.

### Icons

Since Android and iOS have different icon styles, we tried to follow the native style when designing icons for each of them. Android allows icons with a transparent background, so we can use a simple geometric shape for it following Material Design. iOS has completely filled backgrounds with rounded corners (automatically shaped by iOS on home screen), with the latest trend being gradient backgrounds, so we followed suit while preserving our main icon shape and colours.



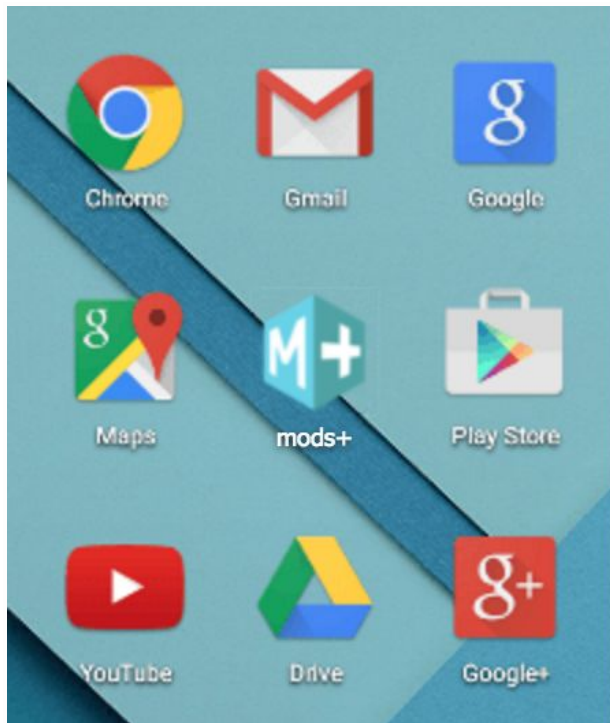
Android



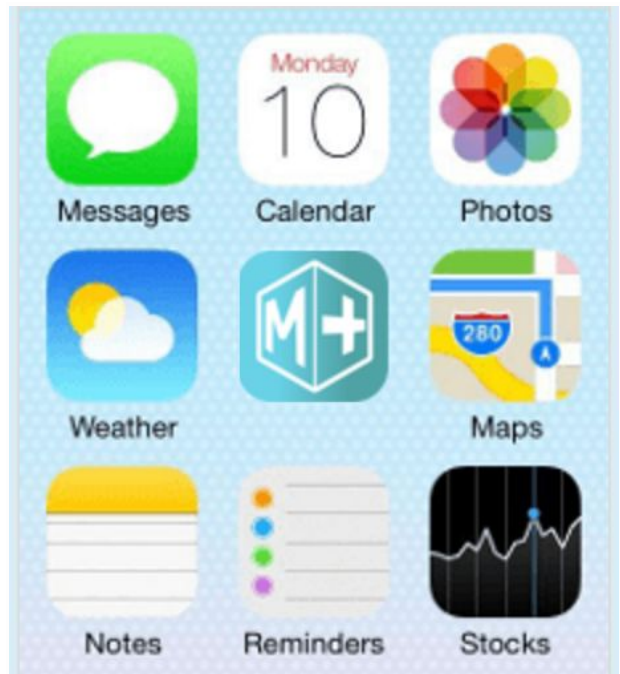
iOS

**Preview of our app when added to home screens of different systems**

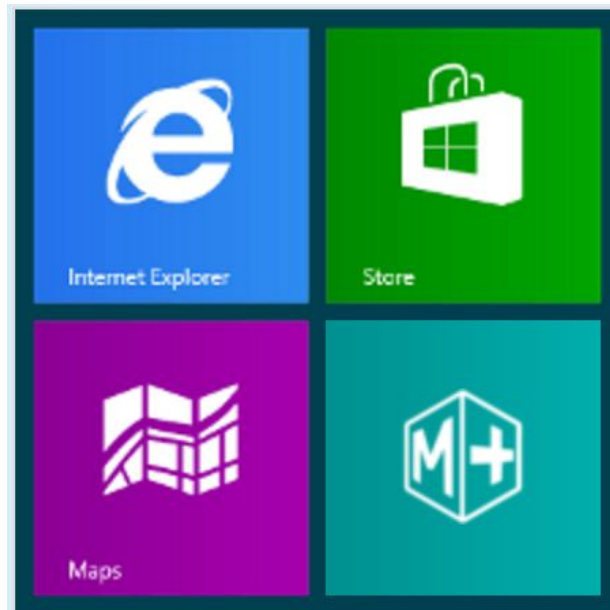




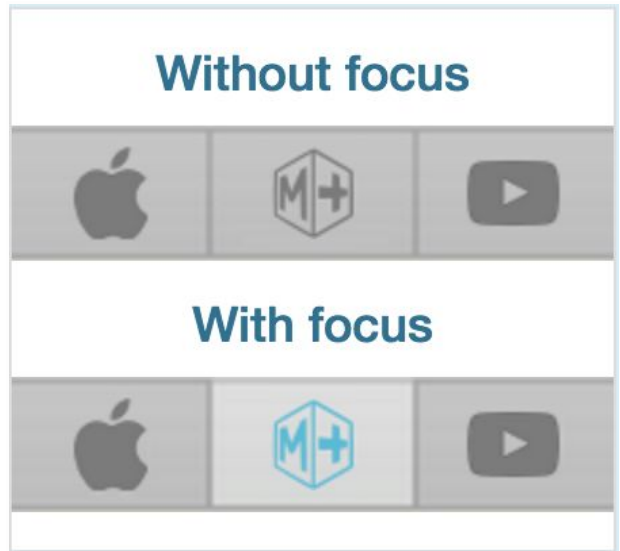
Android



iOS



Windows Metro tile



Safari silhouette

## Splash Screens



mods+

**Milestone 7:** Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application. Choose one of the CSS methodologies (or others if you know of them) and implement it in your application. Justify your choice of methodology.

Our UI design goals are to be mobile-friendly, and have the least cognitive load for the user.

To lessen cognitive load for our users, we aimed to reuse user interactions that have most likely been learned by our target users:

- Vertical timetable, easy to check time and day, similar to Google Calendar
- Display of modules on a grid with coloured blocks, similar to NUSMods

Since users on mobile sites tend to have a goal in mind and want to complete it swiftly, we also placed main features and actions prominently, with most workflows requiring only one to two clicks to finish:

- All main features always shown in navigation bar



- Most options are displayed on page itself and require only one click
- Autocomplete for text inputs so users type less

In terms of style, we used Google's Material Design to build our application, as that would be familiar to mobile users, and used a bright colour that is pleasing to the eye, light blue. CSS methodologies helps to prevent class name collision. In this assignment we make use of webpack and its css-loader's CSS module functionality to help us modularize our CSS. For example, the CSS for our Timetable component is only loaded in the file, and the class names are combined with the component name and a hash to ensure that styles do not accidentally cascade.

**Milestone 8:** Set up HTTPS for your application, and also redirect users to the <https://version> if the user tries to access your site via <http://>. Name 3 best practices for adopting HTTPS for your application. Explain the term "certificate pinning" and discuss the pros and cons of adopting it.

<http://modspl.us> redirects to <https://modspl.us>  
<http://beta.nusmods.com> redirects to <https://modspl.us>

3 best practices for adopting HTTPS

1. Use 2048 bit private key
2. Redirect all http traffic to https
3. Keep your secret key secret

Certificate pinning is when the whole signature hierarchy is ignored, and only \*this\* certificate is trusted. Usually, certificates are shipped with the OS or the system, and these certificates are trusted. However if one of those are compromised, certificates signed by this will be compromised as well. Pinning fixes the CA's public key signature to the certificate, which will be verified by the client.

**Pros:** more secure, compromised cert that is shipped with your OS / browser will not compromise your connection.

**Cons:** more complicated to set up, you can pin to the exact certificate, to the root certificate, or to the SPKI, each presents its own pros and cons.

**Milestone 9:** Implement and briefly describe the offline functionality of your application. Explain why the offline functionality of your application fits users' expectations. **State if you have used service workers, Web Storage, or any other technology. Explain your choice.** Make sure that you are able to run and use the a subset of features of your application from the home screen without any internet connection.

Once the user has used the app once in online mode, module data for the semester is loaded. If the user goes offline now, the module data will still be available, so user can still

plan their modules. This is handled by service worker, caching the `fetch` event for api calls to retrieve modules.

When the user modifies their timetable, we first save the results to localStorage. We run a regular sync call to save the timetable to database. If the user is offline, the localStorage data will be persisted to the database the next time connectivity is restored.

**Milestone 10:** Implement and explain how you will keep your client synchronised with the server if your application is being used offline. Elaborate on the cases you have taken into consideration and how they will be handled.

When the application is used offline, we note when the data is last modified and save the data to localStorage. When we get connectivity, we also know when the data in the database was last saved (using the updatedAt field). We compare these 2 timestamps, and then keep the newer one. If the localStorage version is newer, we save the localStorage version to our database.

**Milestone 11:** Compare the advantages and disadvantages of token-based authentication against session-based authentication. Justify why your choice of authentication scheme is the best for your application.


Since HTTP is stateless, if we authenticate a user, we still do not know the user on the next request. We would therefore need to authenticate again. Token-based and session-based authentications help to resolve this issue.

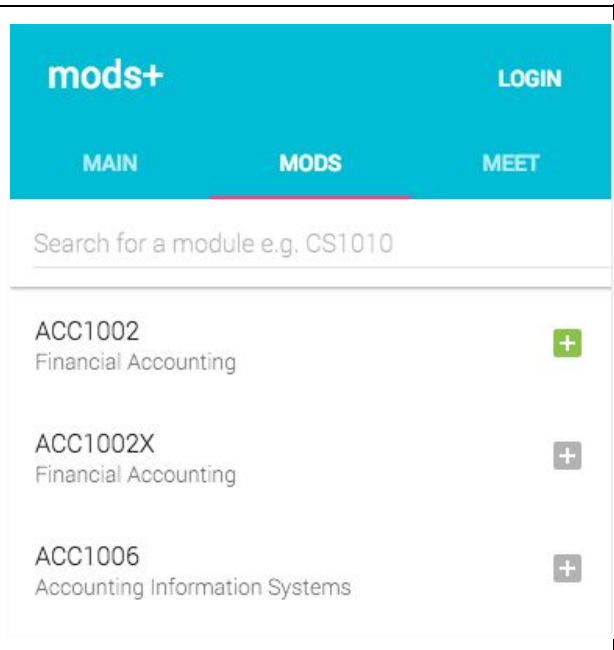
	Token-based Authentication	Session-based Authentication
1	Works well across domains, information is transmitted in the HTTP header	Doesn't work well across domains
2	All information in the token (stateless). Information could include userId, expiry date and secret key. Hence, less memory usage and easier to scale.	Requires a session store to store the session_id (stateful). Hence, more memory usage and harder to scale if using more than one server.
3	Validated against a secret	DB lookup to check if session is valid
4	Token needs to be stored somewhere, commonly local storage, which is only accessible by domains and subdomains, so we get the same problem as 1.	

**Milestone 12:** Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. Lastly, list down some (at least 5) of the mobile site design principles and which pages/screens demonstrate them.

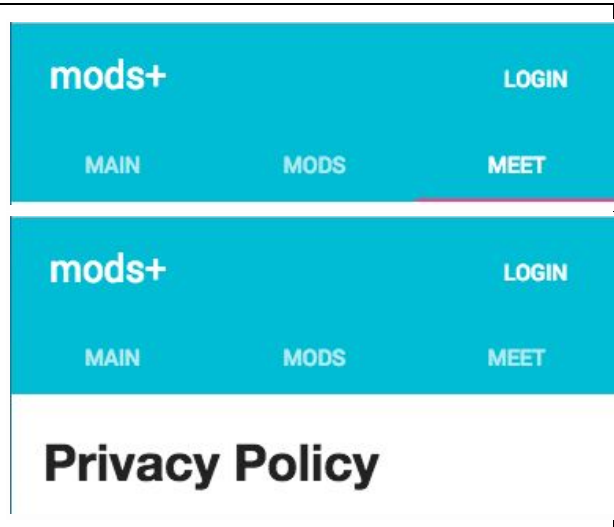
**Choice of library:** Material UI  
**Alternatives:** react-mdl, Bootstrap  
**Reason:** We wanted a mobile-first ui library with pre-made components, and Material UI was the best one. It also follows the latest design trend we wanted to use in our website's design: Google's Material Design.

Based on Google's white paper on Principles of Mobile Site Design, here are some mobile site design principles we fulfilled in our website design.

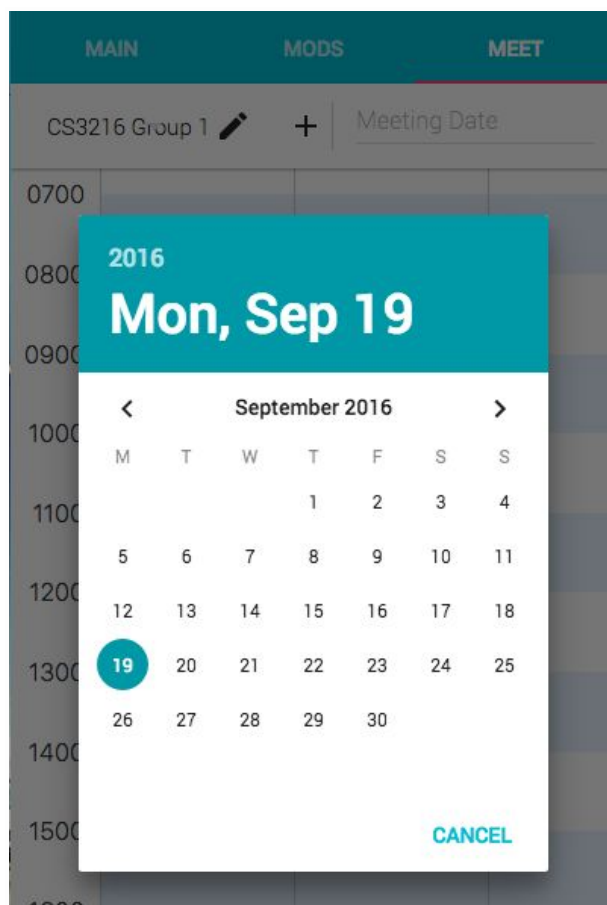
1. Calls-to-action front and center			
		On the Main page, we have a floating action button so users can quickly add modules to their timetable.	

	<p>On the Mods page, each module has an add button for the same purpose.</p>
---	--

## 2. Keep menus short and sweet

	<p>We have our three main functions displayed prominently at the top of the page, and they are always visible even on side pages.</p>
--	---

## 3. Provide a visual calendar when selecting dates



For the Meet page, users select a meeting date using a calendar popup.

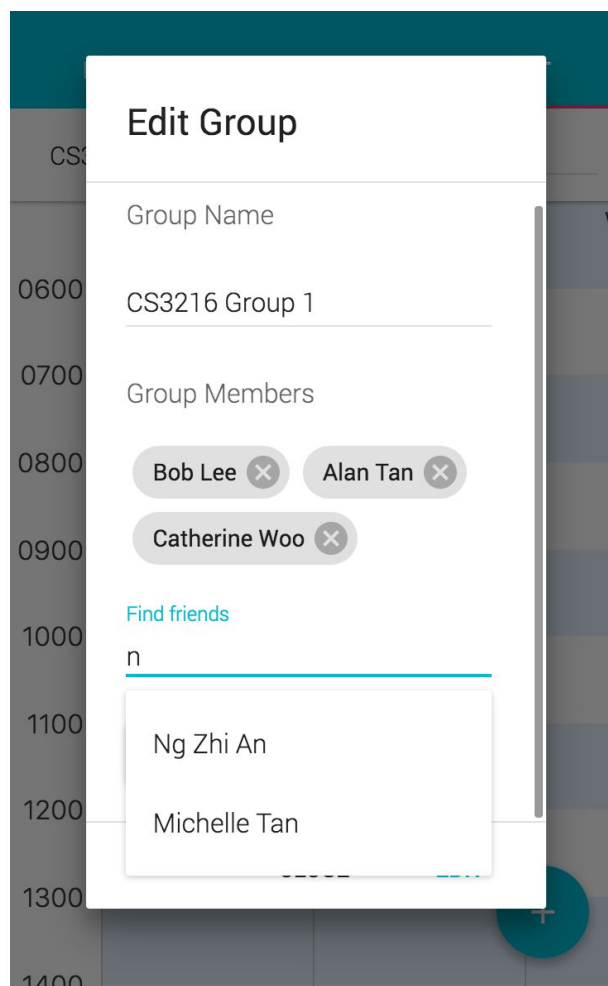
4. Let users explore before they commit

mods+				LOGIN
MAIN	MODS		MEET	
	Mon	Tue	We	
0600				
0700				
0800				
0900	<b>ACC3603</b> SEC [C1] BIZ2-0413A	<b>AR3421</b> TUT [1] SDE-SR12		
1000				
1100			<b>AR3421</b> LEC [1] SDE-426	
1200				
1300				

Users can use mods+ without logging in. They are able to plan their classes and this timetable is saved to local storage.

The only feature they can't access is Meet, which requires users to be logged in. They can use the timetable planning feature first and decide whether to link their Facebook account for the Meet feature.

## 5. Streamline information entry

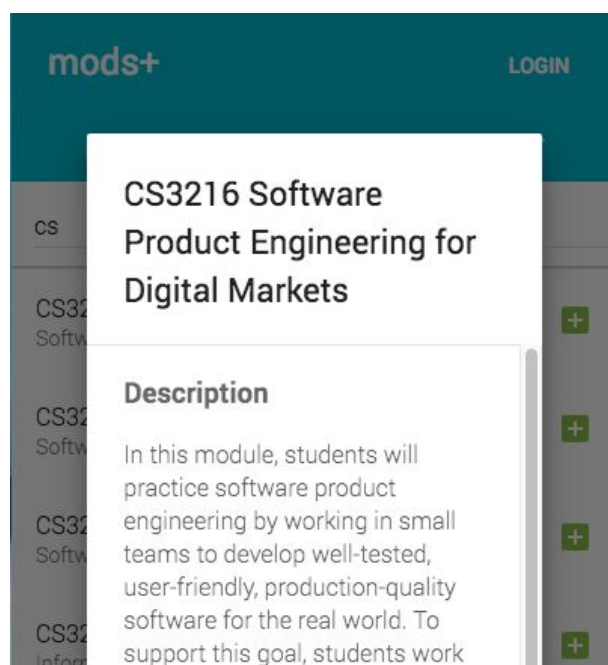


When adding group members in the Meet page, names are suggested to users as they type to make it easier to key in members.

The moment they select the names in the dropdown list, the member is added. Removing the member is also as easy as pressing the 'x' button.

This makes a usually tedious process (keying in full names) much easier for users.

## 6. Don't make users pinch-to-zoom



Information is shown in just the right size for users so they don't need to zoom in.





**Milestone 13:** Describe 3 common workflows within your application. Explain why those workflows were chosen over alternatives with regards to improving the user's overall experience with your application.

### Workflow 1

*Task:* Add a module to timetable

*Steps:*

1. Press on sticky  button on Main page itself
2. Module list with search function displayed, press on the  button next to a module in the list

*Alternative:*

1. Go to Mods page where module list is displayed
2. Press on a module in the list
3. Press on 'Add to Timetable' button

*Improvement:*

- Fewer steps, lesser time taken to complete task
- No need to navigate to a separate tab, everything is done within same tab

### Workflow 2

*Task:* Change selected time slot for a module's activity in timetable

*Steps:*

1. Press on activity to change
2. Alternative timeslots are displayed on timetable, press on time slot to change to

*Alternative:*

1. Press on activity to change
2. Popup dialog displays available timeslots in a dropdown menu, press on time slot to change to

*Improvement:*

- Less cognitive load on user, showing the available choices immediately on the timetable with the time and day context helps them recognise the timeslot that they need much faster, as well as whether the timeslots clash with any other activities

### Workflow 3

*Task:* Check matching free timeslots with group

*Steps:*

1. Press the Meet tab
2. Select the group you want to check from a dropdown list of saved groups
3. Enter the date you want to check

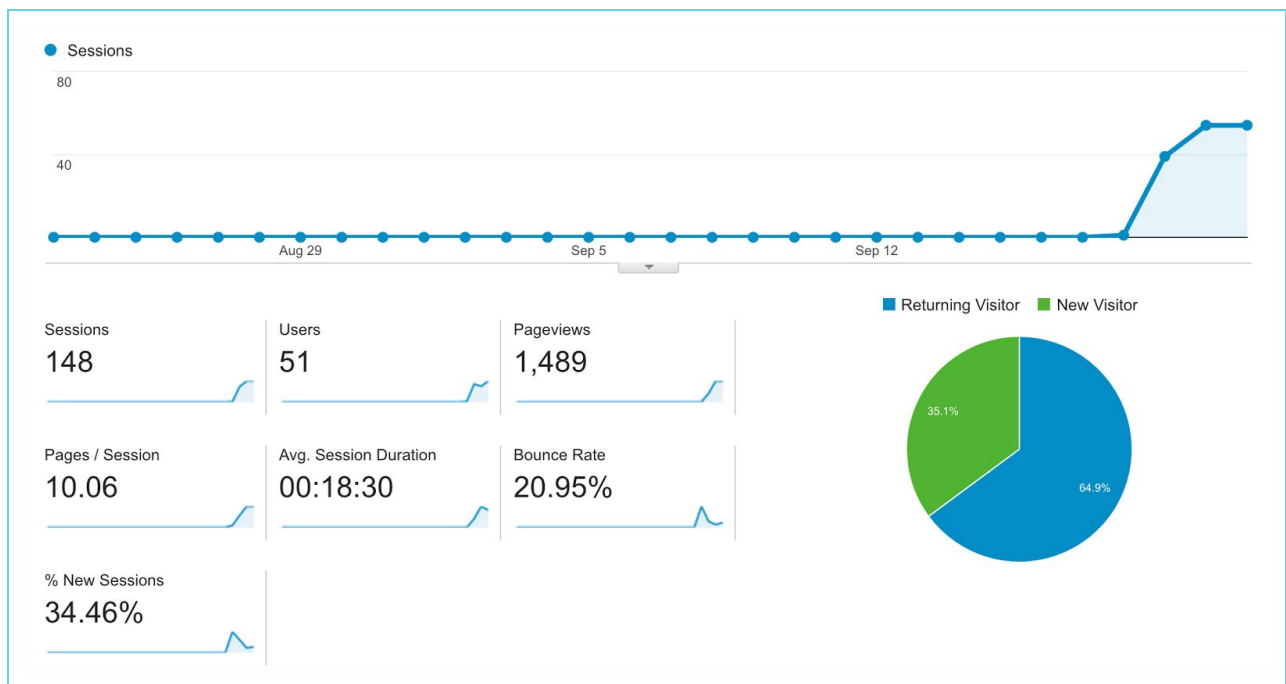
#### Alternative:

1. Press the Meet tab
2. Enter the date you want to check
3. Select the member you want to compare timetables with
4. Repeat steps 2 and 3 until all members are added

#### Improvement:

- Fewer actions needed, since most projects require multiple meetings with the same group of people, having saved groups makes it easier to quickly check for meeting times instead of adding people one by one each time

**Milestone 14:** Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.



**Milestone 15:** Identify and integrate with social network(s) containing users in your target audience. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)

We decided to implement Facebook login. No social plugins were used. Instead, we used passportjs with Facebook strategy in the backend.

During our discussions, we were deciding among Facebook login, Google login and IVLE login. Having multiple login methods makes sense but would involve more development time. Given the tight schedule, we decided that we would choose one of these to implement.

Facebook login makes sense because users may want to have meetings with friends who

do not take any of the same modules as themselves. Using Facebook login allows us to check a user's friends so that the user can also form teams with friends who do not take any of the same modules.

Google login is useful in a different way. Since many people use Google calendar, syncing Google Calendar would help in planning for meeting times, instead of using our current (perhaps naive) assumption that all times without lessons are free timeslots.

IVLE login would also be helpful since all NUS students have IVLE. We might also be able to automatically retrieve a student's modules. However, it would be more difficult to scale to other tertiary institutes if we only implement IVLE login.

Eventually, we decided to do Facebook login as we had implemented it on our previous projects. We have also planned for multiple login methods by having a separate table called UserLogin. This would allow for easier implementation of Google login in the future.