Assessment 2

Web application

Motivation

The best way to learn is by doing. The best way to learn **how to build** a web application is to **build** a web application.

We will be supporting you throughout the whole process of developing a web application: helping you make decisions, guiding you through your independent learning and teaching you general principles of good design. These general principles will apply not only today but also decades later when you are developing or developing with new technologies that haven't been invented yet.

This assessment is motivated by learning objectives 1, 2 and 3 from the subject outline.

Assessment Item

You will work in groups of four developers to design and develop a modern web application (note: you may be allocated to groups of three or five, depending on your lab size). You will be allocated into groups in Week 5, but your preferences for group members will be taken into account.

You will use JavaScript or TypeScript (running on a Node.js server and in the end-user's web browser) to develop the web application specified below.

You will demonstrate the full **functionality** of your application in the Week 12 laboratory session, but you can continue to make changes to your code for final submission on Monday of the first assessment week (11:59pm on 2 November 2020).

The scenario of this assignment is to build a tool for companies to improve internal collaboration and for social clubs to encourage friendships.

You will build a tracking system for “IOU”s or “favours”. An “IOU” is an abbreviation of "I owe you", and it is an informal acknowledgement of a debt or favor owed. You will build a system that allows groups or teams to log in and record the favors that they owe to each other.

Favor tracking

The purpose of the system is to keep track of small favors among friends or groups.

For example, suppose that Alice and Bobby go out for coffee together and Alice buys Bobby a coffee:

- Bobby logs into the system and records that he owes Alice a coffee
- Alice can log in and see Bobby owes her a coffee (as well as all debts Alice owes others and that others owe Alice)

Suppose that a few weeks later, Alice and Bobby go for coffee again. This time Bobby will pay for coffee:

- Bobby can log into the system to see that he owes Alice a coffee
- Alice can log in and remove the debt when Bobby has paid for the coffee

Favors

The system is designed to be fun, so it only allows favors. It does not allow money to be recorded.

The favors should be low-value items that have an element of "fun". For example, your system might support a choice of coffees, chocolates, mints, pizzas and cupcakes (you can choose any favors you want to support in your system).

When Alice and Bobby go for coffee together, either of them can create or delete the favor. However, to prevent cheating, a photo must be uploaded as proof when performing an action that would disadvantage another user. For example:

- Bobby can add the favor that Bobby owes Alice, without any proof
- Alice can add the favor Bobby owes Alice, only if she uploads a photo as proof
- Bobby can delete the favor that Bobby owes Alice, only if he uploads a photo upload as proof
- Alice can delete the favor that Bobby owes Alice, without any proof

Requests

In addition to recording favors, the system allows users to post public requests with an offer to provide a favor (the reward). For example, suppose that Carol would like somebody to clean the office fridge, she might post a request: "Clean the fridge" with a reward of a coffee and a chocolate. When Greg sees the request, he might also

decide that he wants to see the fridge get cleaned so Greg decides to increase the reward by adding another chocolate.

Finally, suppose that Peter logs in and decides that he is happy to clean the fridge to earn himself a coffee and two chocolates. When he finishes the cleaning, he uploads an image as "proof" and then all the rewards on the request will turn into favors owed to Peter: Carol will owe Peter a coffee and a chocolate, and Greg will owe Peter a chocolate.

Core features

- The website has a "front page" with currently available public requests
- Anonymous users (as well as logged in users) can browse the list of available requests, and search by keywords or by rewards
- The website has a live "leaderboard" showing the "best" users (you can choose what "best" means: e.g., least debts, most debts, most active, or some other criteria)
- New users can create a new account to sign up
- Users can log in and are then able to:
    - View the favors they owe other people (and see the uploaded photo proof, if applicable)
    - View the favors that they are owed (and see the uploaded photo proof, if applicable)
    - Record a new favor owing to another person (the favor should be from a fixed list of at least five choices, but you can choose the possible favors)
    - Record a favor as having being repaid
    - Create a new request that will appear on the "front page" and specify the reward from favors in the system
    - Add to or remove from the rewards offered on existing requests (the request should be deleted if there are no favors remaining)
    - Complete a request and claim the favors by uploading an image as proof
- Any lists that can be very long (e.g., the currently available requests) should show a limited number of items at a time (e.g., you might show 10 items per page or you could use ‘infinite scrolling’)

You should interpret these features in ways that make sense to you (e.g., how you rank users, the types of favors, the criteria for sorting, the user interface).

Party detection

Sometimes cycles or loops will be formed. For example, Alice owes Bobby a coffee, Bobby owes Carol a coffee, Carol owes Greg a coffee and Greg owes Alice a coffee.

The system should detect these cycles. It should then recommend a "party" to all the users, so that they can get together to clear all the debts at once. For example, it would recommend that Alice, Bobby, Carol and Greg all go for coffee together.

You can detect the cycles as soon as they occur or you can use a delayed detection process. The recommendations can be delivered to users any way that you wish (e.g., by notification in the app, by email or in a separate page of the app).

Please note in the marking scheme that party detection is only required to achieve the highest grades: you can pass this assignment without implementing party detection.

Implementation constraints

- You must use React (https://reactjs.org/) or the latest version of Angular (https://angular.io) to implement the front-end logic
- Your application must **not** use AngularJS version 1
- Your application must **not** use jQuery anywhere
- All code must be written in either JavaScript or TypeScript
- All comments, commit messages and READMEs must be in English
- Passwords must be handled in a secure way (it should not be possible to steal passwords even if your database is compromised)
- You must implement the server-side logic yourself using Node.js: do **not** use Firebase or other "Platform-as-a-Service" offerings
- You must use a database: you can deploy a database yourself or use a cloud database
- Your code must be deployed on the public internet using a cloud host (e.g., Amazon, Google Cloud, Azure)
- Your server-side logic must expose a RESTful API that other developers could use
- You must use git to track your changes and you must regularly push your commits to a **private** repository hosted on GitHub or GitLab that is shared with your tutor and @benatuts or benjamin.johnston@uts.edu.au (in addition to your other group members)
- Your application must **not** be offensive (no hate, insults, discrimination, pornography or violence)

Innovation and Inventiveness

If you are aiming to get a perfect score in this subject, you will need to conduct independent research that goes beyond classroom material, to find and implement an inventive solution to one or more of the challenges of this project. For example, while the project can be implemented using standard technologies and components, you will need to demonstrate innovation or inventiveness by enhancing user

experience or scalability through a specialized technology or technique (e.g., Websockets, Load-balancing or clustering, Sharding, Machine Learning, Serverless).

Note, however, that you will only get credit for innovation and inventiveness if you otherwise have achieved a score of 20 in "The quality of your design" of the assessment criteria. For this reason, I strongly encourage you to complete the rest of the assignment before considering **how** or even **if** you will add additional innovation or inventiveness.

Usability

Your tutor must be able to use your user-interface without guidance from you (i.e., its user interface should be self-explanatory).

Costs

Github and GitLab provide free hosting for private projects.

Please use trial periods and free non-commercial licenses when deploying your application or if you integrate your system with commercial services. Reimbursement or financial support will **NOT** be provided if you choose to use paid plans or services that cost money.

Please be very careful when deploying your application. Major cloud hosting providers provide free services in addition to trial periods and free education plans suitable for deploying your application. These free services should be used for this assignment. Please be careful when deploying instances as running additional instances can quickly result in a large bill.

Submission

Your assignment will be demonstrated to your tutor and your classmates during your Week 12 lab session. You will have a chance to polish your source code before final submission by the end of the first Monday of the Assessment period (Monday 2 November 2020 at 11:59pm). The master branch of your GitHub / GitLab repository will be marked.

Before you demonstrate your system, you should thoroughly test your system—attempt to break it using a "hacker's mindset". Your application may be tested with difficult inputs. Some tests you should perform include the following:

- Attempt to log in with an incorrect password
- Enter invalid data and check that validation works on any fields (consider: empty input, incorrect input, too much input, non-English input, no image, large images, corrupt images)

- Check that your system works with concurrent users
- Test all the functionality you have implemented: check that every button and every link works
- Check that you can log out and log in as somebody else
- Attempt to access secure pages without logging in or otherwise "hack" your system

Assessment Criteria

This assessment is worth 40% of your final grade.

You will be receiving feedback and guidance throughout the semester. Many laboratory sessions will be related to this Assessment and provide feedback on your progress.

The final submission of your project will be assessed according to the following criteria:

Correctness/completeness of your implementation (15 marks)

This criteria will be marked in-class during your Week 12 tutorial.

0	Your application does not run, crashes regularly or fails to satisfy several required functions or non-functional requirements.
5	You have implemented most but not all of the core features, you have not satisfied all of the non-functional requirements or your application crashes during demonstrations (or shows user-unfriendly error messages).
9	You have implemented all of the core features and satisfied the non-functional requirements. Your application does not crash even with unexpected inputs.
12	You have implemented **party detection**, in addition to all of the core features and non-functional requirements. Your application does not crash even with unexpected inputs.
15	Your application is reliable, secure, fast, scalable and user friendly. Your application is ready for public use.

The quality of your design (20 marks)

This criterion will be used to mark your final submission. The master branch will be assessed.

0	Your system lacks structure or coherent design.

5	Your code has an appropriate design but there is substantial room for improvement. In particular, there may be significant mixing of purposes (e.g., user interface state embedded in data storage code).
8	Your code is appropriately designed. You have separated your system into layers and you have used libraries where appropriate. Some refactoring would improve the quality of what you have done.
14	Your system is well designed and code is well separated into layers. Components are reusable. The codebase represents the quality expected from professional developers. You have used libraries where appropriate.
18	Your system is beautifully designed and flawless. The codebase represents the quality expected from professional developers and has been designed with scalability, robustness and usability in mind. You have elegantly used libraries where appropriate.
20	Your system is beautifully designed and flawless. The codebase represents the quality expected from professional developers and has been designed with scalability, robustness and usability in mind. You have elegantly used libraries where appropriate. In addition, you have demonstrated elements of **innovation or inventiveness** arising from independent research in solving challenges in your problem.

The quality of your documentation and collaboration (5 marks)

This criterion will be used to mark your final submission. The master branch will be assessed.

0	Your code is difficult to read or it would be difficult for another developer to join your team. You will also receive this mark if your code is messy, there are large amounts of unused/commented-out code, or if you have not used revision control properly as a team.
2	Your code and the documentation is acceptable. Most code has comments. Classes, methods and variables have clear names. A skilled professional developer would be able to contribute to your project without feeling a need to rewrite major parts of what you have done. You have used collaboration tools (such as git) effectively as a team.
5	The quality of your documentation and the clarity of your code (including commits, READMEs and the code itself) is of an extremely high standard. It makes it easy for professional developers of varying skill levels to understand the evolution of the system as well as the current state of the system. Technologies are used appropriately and idiomatically (i.e., using standard coding conventions for the technology).

Assessment Penalties

Teamwork is an important part of this project. If you work alone, submit an individual project or do not collaborate with your group members, you will receive 0 for this assignment.

The system demonstration and final submission are marked separately. A 10% penalty will apply to the relevant criteria for each day your submission is late. Late submissions should be uploaded directly to UTS Online.

If you are not present for the system demonstrations in Week 12 you should contact the subject coordinator to arrange an alternate demonstration. Your demonstration is due at the start of the laboratory session and so is considered late if it is not ready to be demonstrated at the start of your session.

If you have not used git or if you plan to submit the final version of your code late, please ‘zip’ the entire project (including the .git folder) and upload to Canvas. This ensures a reliable timestamp for calculating late submission penalties.

Late submission penalties are calculated based on multiples of 24 hours from the due date.

Group Work and Misconduct

This is a group assignment. You are encouraged to discuss your work with other groups. However, all other work that you submit as part of this assignment must be entirely your own and any assistance properly identified and acknowledged. You are permitted to make use of libraries, sample code, tutorials and other resources found online but you **must clearly identify the source at the top of every single method that you did not personally write**. Failure to properly acknowledge code may result in penalties including a mark of zero for this assessment, failing this subject or even suspension/expulsion from the university, in accordance with university policies.

Your git commit logs should be an accurate reflection of the code that you have contributed. If a team member is not able to use git, teach them how to use git rather than committing their changes. Ensure that you have configured your git "user.name" and "user.email" correctly, that you write clear and accurate commit messages, and that your work appears in the master branch, so that your contributions can be correctly attributed to you.

It is expected that group members will contribute equally to the assignment. The assignment will be marked as a group, and then individual marks are allocated on the basis of a peer assessment process and individual git commits.

Please discuss with your tutor if a group member is not contributing to the assignment (or if you are not given opportunities to contribute). Your tutor may, on the basis of contributed workload, heavily penalize students who do not contribute to the assignment. A student who contributes nothing may receive zero marks for the assignment. The use of revision control is mandatory and will assist your tutor in understanding the contribution of each group member.

Based on consultation with group members and/or an analysis of the revision control logs, your tutor or subject coordinator may override peer assessment to more fairly reflect contributions.

If you are experiencing group problems, do not wait until the “last minute”. Discuss with your tutor as soon as possible so that there is sufficient time to correct any issues or negotiate a solution.

If you are in doubt, please ask your tutor.

Please refer to the universities policies for more information about Student Misconduct:
https://www.uts.edu.au/current-students/support/when-things-go-wrong/student-misconduct