COMP2913 24/25: Software Engineering Project Brief

Project Description

1 The product

This project involves the development of an application for car sharing for commuting and/or one-time journeys.

Users must register with the system to use it and book a journey.

The user can:

- Make a booking, select a specific journey, and choose to end the booking provided it has not started yet.
- Converse with the user who proposed the journey for negotiations on (time/pick up location/drop off location/cost/delays and cancellation).
- Search for journeys based on their needs – with pick up locations, number of people, day/times optional.
- Submit a journey to be offered (time/pickup/drop off/maximum number of people/cost)
- Provide additional options for the journey – boot size in case of luggage/pushchair, accessibility.
- Create an account and track their own bookings
- View the pick up point on a map
- Access some statistics about their bookings, durations, costs.
- Be offered a discount for regular commuting
- Be able to customize a commuting booking to be a quicker process each time once they are established in a car share (example Monday/Wednesday/Friday).
- Rate the journey and confirm the journey occurred
- Raise an issue for support regarding a booking

The manager of the system can:

- Produce estimates of revenue based on the registered users
- View data relating to all journeys (available/booked/cancelled)
- Configure cost of the fee for each booking
- Offer a discount for frequent users – e.g. 4 trips a week.
- Provide support if a user needs to raise an issue with a booking – open dialogue/feedback

2 Problem domain

Familiarising yourself with the problem domain would be useful preparation for this project. Spend time exploring how similar systems work. Use some SQIRO investigation techniques, if appropriate.

3 Implementation

Specific functional requirements are enumerated in the product backlog, provided separately.
One important (and hopefully obvious) point to note is that your solution will not be a real system!
It will need to record details of customers, customer data and payments.

Although your solution is subject to the technical constraints outlined below, this still leaves you with a certain amount of freedom to choose tools, frameworks, libraries, and programming languages that suit your team's interests and abilities.

In choosing technology, be realistic about what your team can do in the time available, and keep in mind that a reasonably good mark is achievable for relatively modest implementations provided that the organisation and management of the project is good.

4 Basic architecture

Server:

Your solution should have a client-server architecture and it suggested that you implement a three tier architecture approach using an appropriate Web framework. For the highest marks, your system should be able to cope with simultaneous connections from several clients.

The server should interact with a database that stores details specific to the domain: for example, registered users, journeys, and times and dates. Ideally, this should be an SQL or noSQL database.

A solution in which data are stored as files in CSV, JSON or some other format is acceptable, but will earn you fewer marks.

To simplify development, demonstration and marking, use either an embedded database or a database server that can be run without needing special privileges or installation in system directories. SQLite is a suitable choice of embedded database. There is no preferred language for this project, it is up to your team to choose something that you are all capable of working with.

Client:

There are 2 required interfaces to the system:

- a customer interface;
- a management interface.

You have the following options for your client(s):

C1. A desktop, web or mobile interface that customers can use to register, pay for services, view their data.
C2. A desktop or web implementation of the management interface that a manager would use to maintain and amend accounts, amend details, and perform a selection of other administrative duties.

If you choose a web client, please note that it should provide a responsive, mobile friendly and accessible interface and that it should make non-trivial use of appropriate JavaScript libraries, rather than simply being based on HTML5 and CSS3.

Testing:

Testing should not be an afterthought. Your solution should be tested throughout its development, and testing should be automated (perhaps by a DevOps CI/CD process) where possible. It is worth devoting significant effort to this and we will expect to see a testing strategy and evidence of a range of different types of test in your documentation.

Building and deploying:

You should automate as much of the build, test and deployment processes as possible. Many of the tools to support this are built into Github although other tools may be more appropriate, depending on the nature of your system and your technology choices. Jenkins (https://www.jenkins.io) is an appropriate tool to try if Github doesn't meet your needs.

It should be possible to build, deploy and run your solution using your chosen build tool(s), with a minimal number of commands. For example, a single command to build the whole system, one command to run the server and one command to run the desktop application (assuming you've chosen the latter option) would be a good way of arranging things. If using Github, then investigate how Github Actions (https://github.com/features/actions) can help you to do this.

Note that we do NOT expect you to deploy the server-side components of your solution to real servers in the cloud! You are welcome to implement this as an optional feature if you have the aptitude and the interest in doing so, but it is not required. For marking purposes, we will require any solution to be runnable locally.

As with testing, it is worth starting early and devoting significant effort to this. Build automation can save you a lot of time and pain, so you should aim to get it set up as soon as possible.

5 Approach:

The broad approach to be followed by all teams is a simplified and scaled-down version of the Scrum method.

The process is driven by the Product Owner (PO), who will provide each team with an initial prioritised list of requirements in the product backlog (published separately).

The primary period of development is broken down into a sequence of three sprints. At the start of each sprint, you should have a sprint planning meeting in which you agree on a subset of product backlog items to implement, identify the tasks required to deliver those product backlog items and then allocate those tasks to team members. Remember that at the end of each sprint, you should have an MVP (Minimum Viable Product).

In this simplified version of Scrum, the PO will not be present at these planning meetings. The team will therefore need to check the product backlog before each planning meeting and seek clarification from the PO if necessary, either face-to-face or via Teams.

You should hold between two and four status meetings during each sprint. Status meetings are short—no more than 15 minutes in length—and involve each team member briefly describing what they have done since the team last met, what they will be doing from now until the next status meeting, and finally what issues they are facing that might prevent progress. Discussion of issues should be deferred to other meetings or online communication, not necessarily involving the whole team.

You should finish your sprint with a sprint review meeting. This should begin with, or follow soon after, a brief demo of progress (and your MVP) to the PO. All members of the team are expected to attend this meeting.

In Sprints 1, 2 and 3 you will report to the PO as a whole team at the end of the sprint for a meeting as part of the review process. The meeting will involve a partial demo of the work produced so far, it will involve reflecting on the previous sprint and the status of the items, plus discussing the upcoming sprint. It allows time for questions and clarification from the PO and any feedback. This meeting should also allow the team to discuss possible team issues to help resolve any conflicts early.

You should use the feedback to reflect on the feedback given by the PO and identify aspects of your development process that can be improved for the next sprint.

Scrum master:

You will need someone to take on the role of Scrum Master (SM). The SM should organise and chair all the sprint planning, sprint review and status meetings. The SM should nominate another team member to take notes during planning and review meetings. The note-taker is responsible for uploading their notes to the project wiki (see below). Formal notes are not required for the status meetings.

The SM should keep attendance records for all meetings (including the status meetings), using one or more pages in the project wiki. The SM should also investigate any absences or missed task deadlines, on behalf of the team, via email or other means as appropriate.

Remember that the SM is not the project manager! Decisions affecting the project should be taken by the entire team. The SM is a facilitator, not a decision-maker. In our modified version of Scrum, the SM is also a developer but is entitled to do less development work than others on the team because of these duties.

It is recommended that the SM role be rotated amongst team members, but changes in SM are only permitted between sprints, not during a sprint.

Team members:

Team members should attend all meetings if possible, and must provide apologies for any absences. Team members should use the project's issue tracker to record information about the tasks they are carrying out. Other project documentation should be stored in the wiki.

Team members should use Git version control for all programming activities and should push the changes they have made locally back up to the remote project repository on a regular basis, to facilitate code review and sharing of the new code with others in the team.

Team members can work on tasks alone or in collaboration with others. You should consider adopting a 'pair programming' approach such as that advocated by XP (eXtreme Programming), for instance. This is particularly recommended for complex or critical parts of the system.

6 Project tools

Your team will be allocated a private Github (GH) repository within the module's Github organisation. You MUST use the allocated GH repository and we will expect to see every team member regularly making commits to the repo during each sprint.

A vital first task for every team member is to check whether you can successfully access your team's repo. Make sure you do this before Sprint 1 starts.

Version control:

All source code must be managed using the Git version control system. Each team should have a shared repository, which can be accessed via the HTTPS or SSH protocols.

If you've not already set up SSH access, we recommend that you do this before Sprint 1 begins.
One other important decision that you need to make before Sprint 1 begins is your preferred Git workflow. By default, all team members have been granted administrator permission in GitHub—which means that all team members have permission to push commits into the master branch. However, it is probably best to avoid this and follow a 'feature branch' workflow, whereby individuals or pairs work on each feature in separate branches and then issue merge requests when the feature is complete.

Someone will need to act on merge requests and handle the task of merging features into the master branch, resolving any conflicts that arise. We recommend appointing one or more lead developers to do this. Ideally, lead developers will be the most experienced programmers / Git users on your team.

Issue tracker:

Each team's repository will include an issue tracker. There is documentation at https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues

Your team should agree on some suitable issue labels and set them up before recording any issues in the tracker. GitHub provides a reasonable set of defaults, to which you can add or remove labels as you see fit. You can also prioritise labels if you wish.

In addition to setting up some labels, you should also define milestones. 'Sprint 1', 'Sprint 2', 'Sprint 3' and 'Final Demo' are needed, but you can define others if you wish.

Use the tracker to record the tasks given to each team member. That team member should be recorded in the tracker as the 'assignee' of the issue. Note that the assignee is responsible for

closing the issue when the task has been completed. You should also use the tracker to record bugs found during testing, suggestions for changes to features, etc.

Issues are written using GitHub's own flavour of a lightweight mark-up language called Markdown. Take some time to familiarise yourself with it. More information can be found at

https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

GitHub also provides a project board, which allows you to group issues into different columns to show current status, Kanban-style. You might find this to be useful way of visualising the overall status of the project. More information can be found at

https://docs.github.com/en/github-ae@latest/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards

Wiki:

The wiki is your project's web site and the home for all project-related materials that are not stored in the repository. We will give you more information of what we expect to see in the Wiki documentation.

https://docs.github.com/en/communities/documenting-your-project-with-wikis

Wiki pages are written using GitHub-flavoured Markdown, so take some time to familiarise yourself with the syntax (see links above).