COMP2913 Software Engineering Group Project

Group Project Description

1 Introduction

This project involves the development of a software system for hiring electric scooters in Leeds City Centre.

The system will have a set of requirements that are provided in a separate document.

You have been grouped into teams to work on this. Your team should arrange a meeting once groups have been confirmed and use this initial meeting to exchange contact details, discuss each other's skill sets, review this document, and consider technology choices for the project. Each of you should also check that you are able to access your team's GitLab project area (see Section 6).

You will have three full sprints, that are each two weeks in length, during which we expect most of the functionality to be implemented. The remainder of the time should be used to implement any outstanding low-priority features, fix bugs, polish the implementation and finalise deliverables.

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 the hire bookings and payments, but it will not actually make those bookings using real money. The payment process should be simulated, but should be realistic. You should, however, use real data where possible. The data can be created manually; we do not expect your solution to acquire such information automatically. You will need to collect a realistic set of data to demonstrate your system.

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

4.1 Server

Your solution should have a client-server architecture. It will be easiest to use HTTP as the base protocol for communication between clients and server, and this will also provide you with the most flexibility, but a TCP-based solution is acceptable. For the best marks, the server should support multiple simultaneous connections from clients.

The server should interact with a database that stores details specific to the domain: for example, registered users, available scooters and locations, prices, and bookings. Ideally, this should be an SQL 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. If you are using Java on the server side, Java DB comes bundled with the JDK and can run either as an embedded database or a database server—either of which would be suitable for this project.

4.2 Clients

There are 2 required interfaces to the system: a customer and an employee/manager.
A basic solution will provide only a single client for the user, whereas an advanced solution will implement two different solutions for the user. You have the following options for your client(s):

1. A web interface that customers can use to view availability and book a scooter hire session.
2. A basic mobile app, for Android or for iOS, that customers can use to view availability and book a scooter.
3. A desktop or web application simulating the interface that an employee would use to

make bookings, take payment, send email confirmation.

4. A desktop or web application that the manager can use to perform selected administrative functions such as calculating money taken for the week, reviewing usage of the different scooter hire periods, updating details.

The mobile app option is aimed at more ambitious teams who are prepared to learn some app development on their own initiative (or at teams containing members with some relevant prior experience). 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 JavaScript, rather than simply being based on HTML5 and CSS3. If you choose to implement a desktop UI in Qt, this does not constrain you to using C++; an implementation in Python, using the PyQt bindings, would be perfectly acceptable.

4.3 Testing

Testing should not be an afterthought. You solution should be tested throughout its development, and testing should be automated where possible. It is worth devoting significant effort to this.

4.4 Building & Deploying

You should automate as much of the build and deployment processes as possible. Ant, Maven and Gradle may be viable options for this. Other tools may be more appropriate, depending on the nature of your system and your technology choices.

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 the chosen the latter option) would be a good way of arranging things.

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

5.1 Overview

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.

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 two or three 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 to the PO. All members of a team are expected to attend this meeting.

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

5.2 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.

5.3 Team Members

Team members should attend all meetings if possible and provide apologies for any absences.
Team members should use the project's issue tracker (see Section 5.2) to record information about the tasks they are carrying out. Other project documentation should be stored in the wiki (see Section 5.3).

Team members should use Git version control for all programming activities (see Section 5.1) 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, for instance. This is particularly recommended for complex or critical parts of the system.

6 Project Tools

Project management should be done using a remotely-hosted instance of GitLab, at https://gitlab.com. You should create a project area for your work. You must add the module staff, to your project with full access, so that we can review your work when required.

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

6.1 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:
HTTPS https://gitlab.com/xxx/team.git
SSH git@gitlab.com:xxx/team.git
Substitute your team's name (in lowercase) for team above.

By default, any clone, fetch or push operation involving the remote repository will require authentication. This can be done by supplying your GitLab username and password when prompted. It is possible to avoid manual authentication on every clone, fetch or push by using the SSH protocol and uploading an SSH public key to your account. For further information on this, see https://docs.gitlab.com/ee/ssh/index

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. Please refer to the lecture materials for detailed discussion of this. By default, all team members have been granted Master permission in GitLab—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.

6.2 Issue Tracker

Each team's project area should include an issue tracker. There is documentation at https://docs.gitlab.com/ee/user/project/issues/

Your team should agree on some suitable issue labels and set them up before recording any issues in the tracker. GitLab 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 GitLab'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.gitlab.com/ee/user/markdown

GitLab also provides an issue 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.gitlab.com/ee/user/project/issue_board

6.3 Wiki

Each team's project area should include a simple wiki, which can be accessed at
https://gitlab.com/xxx/team/wikis/home
where team is your team name in lowercase.

The wiki is your project's web site and the home for all project-related materials that are not stored in the repository. Further details of what we expect to see in the wiki are given in Section 6.

https://docs.gitlab.com/ee/user/project/wiki/

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