Chosen option: open-source software (Section 3.1).

Merge: A Dating Website for Software Developers

1. Introduction

MMG (Manchester Match Group) are an British internet company that owns and operates several online dating web sites. They have asked MMU corp to design and build a dating website aimed at software developers.

This document provides a report on the progress of the project in each of the first five phases of the software lifecycle (Ghahrai, 2018):

- Requirements gathering (Section 2),
- Design (Section 3),
- Implementation (Section 4),
- Testing (Section 4.4),
- Deployment (Section 4.6),

and discusses the various approaches and technologies used in each phase.

Maintenance of the software could be provided by MMU Corp if we put in place a maintenance contract with MMG, but they may decide to go elsewhere for this.

Section 5 provides an evaluation of the work done so far. A link to the source code and instructions for building and running the application can be found in Appendix 2.

2. Requirements Gathering

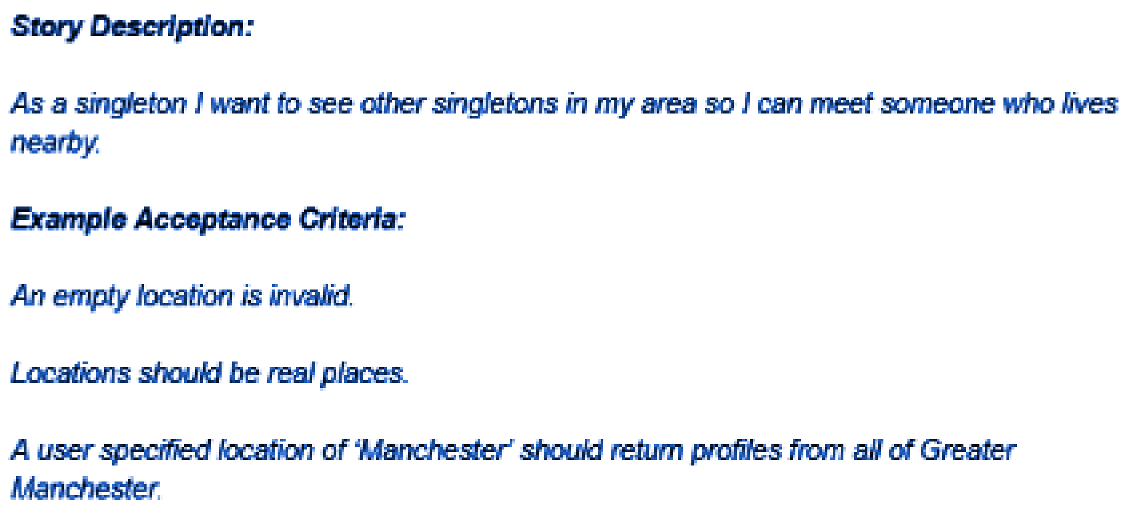
Requirements gathering in software engineering involves defining a business problem so that business needs can be translated into a software solution. Requirements can represent high level needs, or can include lower level technical details. They can express the needs of different types of user, or users with different 'roles' (e.g. unregistered user can create a dating profile, registered users can view other users profiles), or they can express system-wide non-functional requirements (e.g. the application must be built in Spring Boot).

We will not create a traditional system requirements specification (SRS) document (often associated with a 'waterfall' development model) for this project. As we aim to practise an 'agile' project management methodology (Section 4.2), requirements with be gathered dynamically, and refined continuously throughout the development lifecycle. This mitigates the risk of changing requirements. However, it also requires requirements to be gathered in a way that minimises disruption to the user. Some techniques that are commonly used in requirements gathering include end user interviews, questionnaires, observations and workshops.

Given our ‘end user’ is the public, individual interviews and observations will be difficult to carry out, and the closed ended nature of questionnaires mean they are not suitable for this initial requirements gathering phase. As the project matures, questionnaires could be used to determine user priorities, and observations may help to refine the requirements of an application prototype.

Given that MMG operate several other dating websites, they can be considered domain experts, and can serve as a proxy for the website users. Requirements will be gathered from MMG in a series of workshops with developers, a tester and a BA from MMU Corp. The aim will be to ‘brainstorm’ requirements and develop ‘paper prototypes’ representing the main features of the application, and user stories which aim to add further detail to the user required functionality in a verifiable way, using written descriptions, details of stakeholder conversations, and definitions of ‘acceptance criteria’ (Figure 1). Requirements can also be represented as UML ‘use case’ diagrams (Figure 2).

One of the advantages of writing user stories is that the process it highlights issues that might otherwise be missed. The process of writing user stories is the foundation for the ‘Behaviour Driven Development (BDD) philosophy, which uses stories to build scenarios which can then be used to develop unit tests to support test driven development (TDD).



Story Description:

As a singleton I want to see other singletons in my area so I can meet someone who lives nearby.

Example Acceptance Criteria:

An empty location is invalid.

Locations should be real places.

A user specified location of ‘Manchester’ should return profiles from all of Greater Manchester.

Figure 1: An example of a ‘user story’ developed during an initial workshop with MMG.

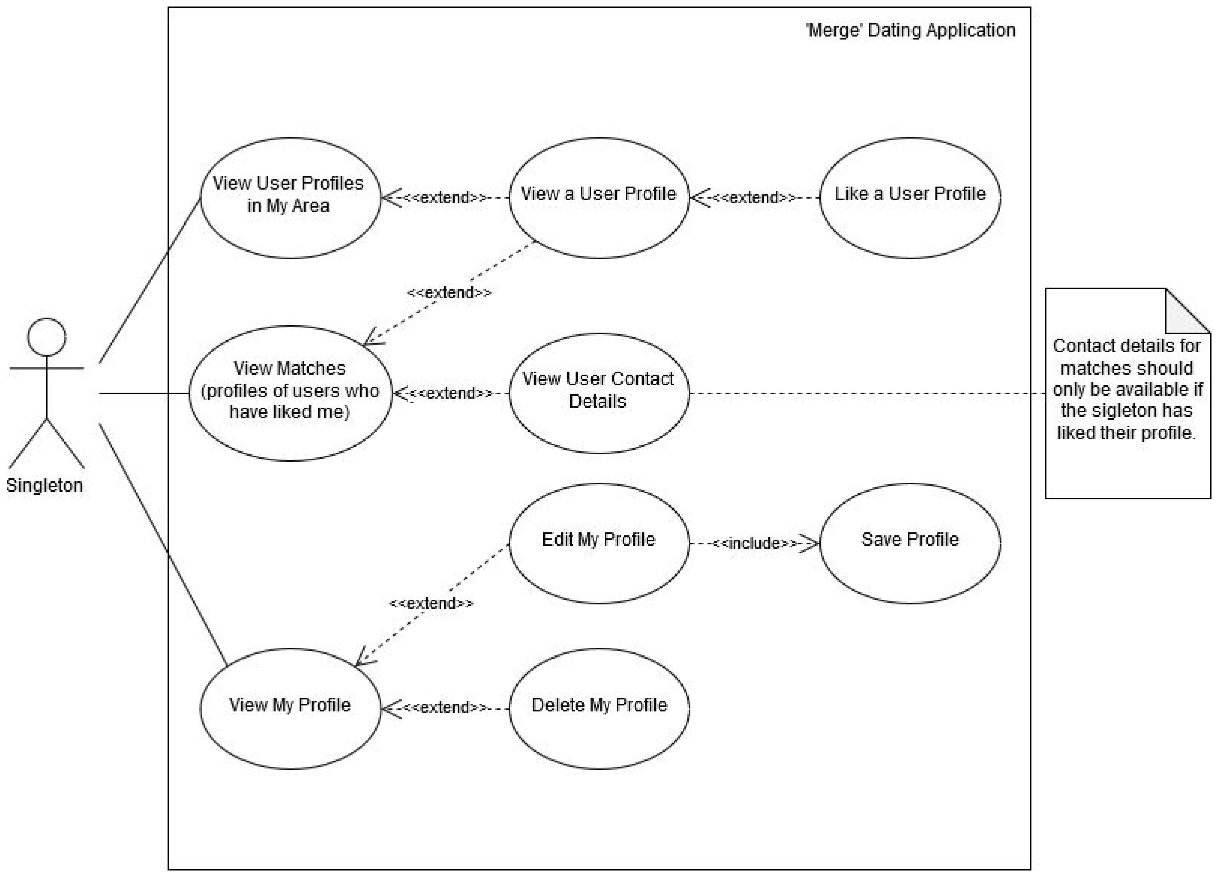

'Merge' Dating Application
View User Profiles in My Area
<<extend>>
View a User Profile
<<extend>>
Like a User Profile
<<extend>>
View Matches (profiles of users who have liked me)
<<extend>>
View User Contact Details
Contact details for matches should only be available if the sigleton has liked their profile.
Singleton
Edit My Profile
<<include>>
Save Profile
<<extend>>
View My Profile
<<extend>>
Delete My Profile

Figure 2: A UML 'use case' diagram representing requirements gathered during an initial workshop with MMG.

3. Design

3.1. Open source Software

This project makes heavy use of Open-Source Software (OSS) including:

- Spring Boot - a Java-based development framework which we've used to create our application.
- Bootstrap - a CSS library used in creating the front-end of the website.
- Thymeleaf - a templating engine (Section 3.2).
- Apache Maven - as our build automation tool (Section 4.5).
- Junit - as our testing framework (Section 4.4).
- Mockito - as our mocking framework (Section 4.4).

OSS is software whose source code has been made available for anyone to view, distribute, and independently develop, subject to the appropriate licence (e.g. 'copyleft' licenses require derivative work to use the same license as the source, whereas permissive licenses only require attribution). Much OSS software is also free of charge. OSS is often developed 'in public', opening it up to a large number of users, who can then scrutinise it, and contribute bug fixes and new features. This leads to products that evolve quickly and (it could be argued) to higher quality code, and the software that users ‘want’. For a business using OSS, it creates the additional advantages of lower overheads, and the opportunity to fix any urgent bugs themselves.

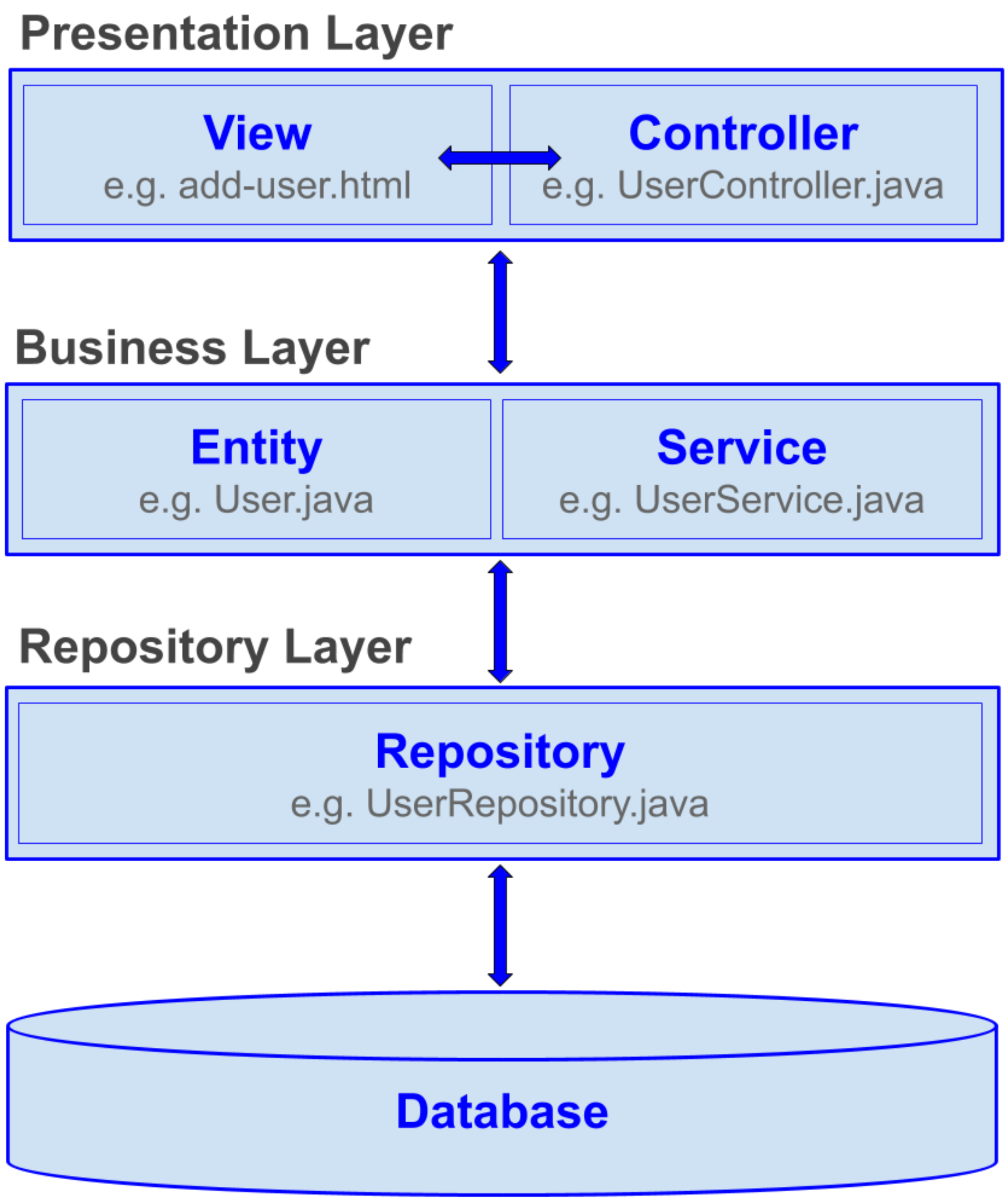
There are, however, some risks associated with using OSS. For example, there is no contracted support (and therefore no obligation for bugs to be fixed, and no accountability if the software doesn’t work as it should). In the event of the discontinuation of an open source project, there would also be no option to pay for long-term support, and it would be unfeasible to take on the future development of such a project ourselves. In addition, if an OSS project is not adequately maintained (specifically by people with the necessary expertise) it may pose a security risk. These are factors that we need to consider when carrying out our risk analyses.

3.2. System architecture

As is common in many business applications, this project uses a layered architecture (Figure 3) with each layer addressing a separate concern. Layered architectures thus embody the computer science principle of the Separation of Concerns (SoC).The aim of SoC is to create code that is easier to scale, reuse and maintain. It allows changes to be made to one component (e.g. a front-end web page, or a database) without other components being affected. Ultimately these advantages translate into cost savings for the business. SoC in a layered architecture does however involve abstraction through the addition of code interfaces. This could result in a larger codebase, and therefore a slower program. In pursuing SoC, and a complete decoupling between layers, there may also be a danger of creating unnecessary complexity in the code.

This project was created with Spring Boot (a Spring-based framework that allows for rapid development of production-ready applications via autoconfiguration). Spring Boot provides default configurations for Spring MVC - a presentation framework which deals with controllers and views. The presentation layer in our Spring Boot project therefore only includes the controller and the view, but as it also deals with a representation of the model, it could be described as using a model-view-controller (MVC) architectural pattern. One advantage of separating the ‘front-end’ and back-end’ code by way of a presentation layer is the separation of development responsibilities, which could improve developer efficiency.

In a Spring Boot project, the controller populates the model, and then maps it to a view name. A templating engine is then responsible for integrating the model object with the correct view (in this project we are using Thymeleaf as our templating engine). The job of the view is then to generate the customer ‘face’ of the application. The HTML files which comprise the ‘views’ used in this project can be found in the ‘templates’ folder in the source code.



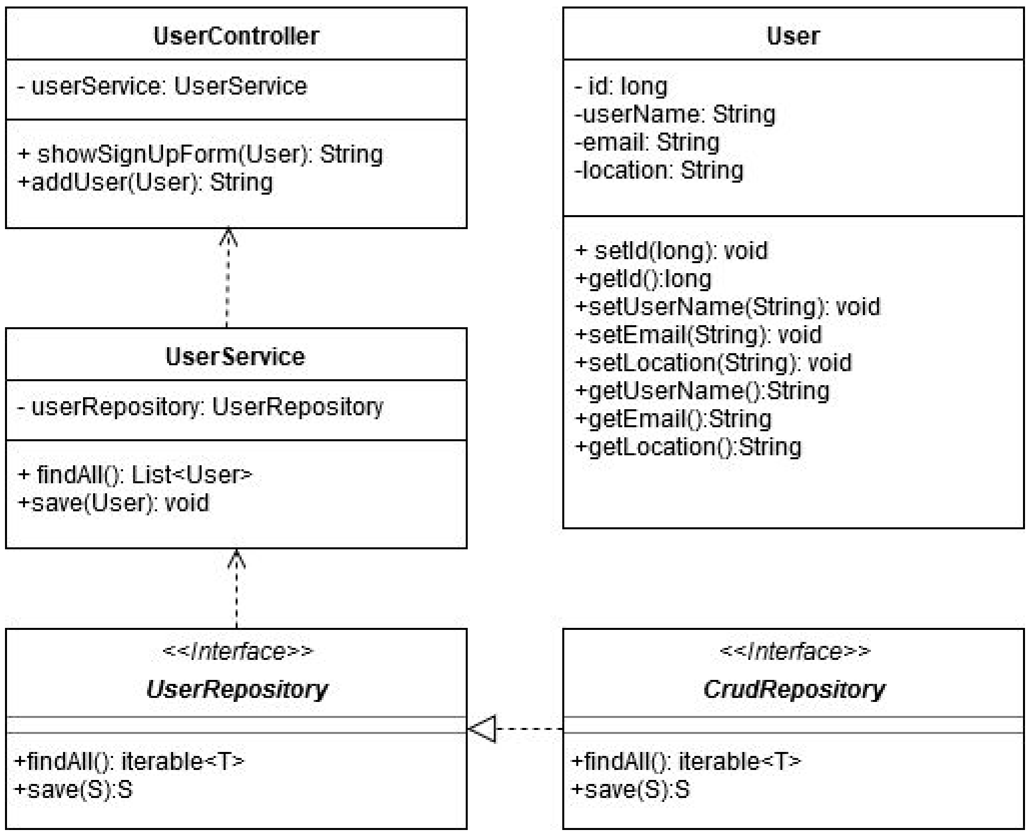
.Figure 3: The ‘four-layer’ architecture used for the ‘Merge’ dating application.

3.3. The Strategy Pattern and Dependency Injection

The ‘strategy’ pattern is a ‘Gang of Four’ behavioral design pattern (Gamma et al., 1995) which defers algorithm selection until runtime. The goal of this is to increase code flexibility and reusability, and is achieved via a context class which holds a reference to the required strategy.

Due to its layered architecture, this project utilises the strategy pattern, with each layer holding references to the classes in the preceding layer. To preserve SoC a ‘Repository’ layer is included in the architecture, which decouples the Business Layer from the Data Layer (Figure 3). Communications between the Business, Data and Repository layers are implemented using the interface classes in the Repository layer (Figure 4).

Making use of ‘Dependency Injection’ (a variety of ‘Inversion of Control’, which delegates the responsibility of setting object dependencies) the repository instance can be injected into a service object in the Business Layer when required. In the Spring framework, this behaviour is configured in the IoC container (i.e. the ApplicationContext).



.Figure 4: A UML ‘class diagram’ for the ‘Merge’ dating application.

4. Implementation

4.1. The Lifecycle Model

There are two main options in our approach to delivering the phases of the software development lifecycle:

- A waterfall approach, where we would finish each phase before moving on to the next.

- An ‘agile’ approach where a large project is split into small features, which are developed iteratively.

In modern software development it is generally preferred to use an agile approach, as it easier and cheaper to make design changes, allows early identification of any issues, is more in line with how people naturally solve problems, and begins to deliver value to the stakeholders almost immediately. Since this project involves delivering a niche project to a specific client, MMU Corp has no previous experience of developing dating applications, and there is no specific need to follow a waterfall approach (e.g. use of limited hardware) it makes sense for us to also follow an agile approach here.

The word ‘agile’ has come to mean many things, but encompasses several techniques which are considered ‘good practise’ in software development, such as ‘test-driven’ development (TDD), continuous delivery (CD) and project management methodologies such as Scrum and Kanban.

4.2. Project Management

Scrum (Scrum.org, 2019) is an agile project management approach often used in software engineering, which emphasises the delivery of partial software products in specified time frames (or ‘sprints’)

Our scrum team, consisting of the ‘product owner’ (MMG), ‘scrum master’ (the project manager - me) and development (our 3 developers) will carry out work in one month sprints. At the start of each month we will hold a ‘sprint planning’ meeting. Work will be selected from the ‘product backlog’ (i.e. the list of requirements defined by MMG - Appendix 1) and added to our ‘sprint backlog’ which contains the work we expect to be able to complete by the end of a particular sprint.

To enable estimation of the work that can be achieved in each sprint, each user story (Appendix 1) will be assigned a priority (based on business need and technical risk) and a number of ‘story points’ (equivalent to expected required development time). At MMU Corp, one story point equal to one day of development time. Stories will be dealt with in priority order.

A typical working month (one Sprint) for 3 developers (not taking into account sickness or leave) equates to 3 x 5 days x 4 weeks = 60 story points. This figure is known as the sprint ‘velocity’. It will be reevaluated at the end of each Sprint to improve our future estimates.

Progress will be visualised using a Kanban style board. Kanban is a visual system that was originally designed for the manufacturing industry. it encourages continuous delivery to the stakeholder, so it can be considered an agile methodology, but it lacks the clearly defined timescales and responsibilities of Scrum. A Kanban board consists of columns representing each development stage. Cards representing each requirement, are moved across the board as they move through each stage of development. A maximum number of cards is defined for each column to prevent bottlenecks. We could use also use a project management tool such as Jira to monitor our velocity over time and/or a ‘burndown’ chart to visualise the number of remaining story points.

Daily activities will be coordinated in a 15 minute ‘daily scrum’. At the end of each month I will lead a meeting with all stakeholders (the ‘sprint review’) to assess the current state of the project (in Scrum terms this referred to as the ‘Increment’) and deliver the latest version of the product to MMG. Following this, the scrum team will meet for a ‘sprint retrospective’ to evaluate performance in the last sprint, and highlight any lessons that can be fed into future sprint planning.

4.3. Source Control

In this project we will use Git, which is the most frequently used tool for source control in software development. Source control (or version control) in the context of software development, is a method of managing and monitoring changes to a code base. Source control tools store the change history of the codebase, allowing developers to identify the source of any issues, and revert to a previous version of the code if required. They also enable the separation of development (unstable) and production (stable) code by allowing the creation of multiple ‘codelines’ or ‘branches’. This also makes it easier for multiple developers to work on the same codebase simultaneously. Some of the advantages of Git include the ease with which code branches can be created and merged, and the existence of several cloud-based Git repository hosting services (e.g. GitHub, GitLab) which simplify the use of Git and facilitate collaboration.

To ensure code stability and minimise conflicts between multiple codelines we will use a popular code branching model called ‘Gitflow’ (Atlassian, no date). This model defines a number of branches with specific roles, and stipulates how they should be merged together. In particular, developers must create a new branch for each new feature, but they branch from, and merge back into a ‘develop’’ branch, rather than the ‘master’ code branch. To ensure the early identification of any issues, developers should integrate their code changes into the develop branch frequently. The process of frequent integration into a shared branch is known as ‘continuous integration (CI).

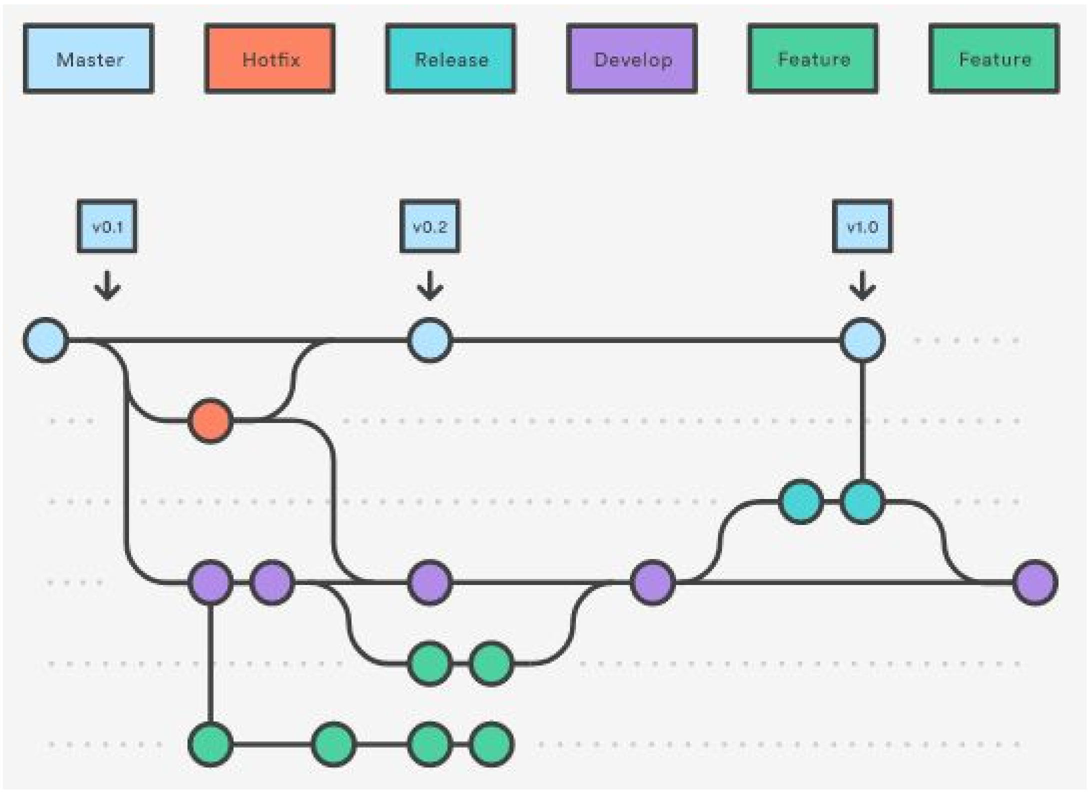


Figure 5: An example of a ‘Gitflow’ branch model in action.

4.4. Test Driven Development

To ensure our code is self-documenting and of high quality, we will utilise a technique called ‘Test Driven Development’ (TDD). This is a central feature of the modern agile development methodology, and involves writing tests first, then writing code to ensure these tests are ‘passed’.

Maven provides built in support for automated ‘unit’ testing via the JUnit plugin, ‘Unit tests’ are those which test individual code ‘units’ such as a class or a method. When the skeleton source code of our project was created using ‘Spring Initializr’ (Spring, no date) it automatically created a ‘test’ directory containing the application ‘MergeApplicationTests.java’. This includes a simple test to verify the successful load of the application context. Additional unit tests were then written for the “User” and “UserController” classes.

To test classes and their methods in isolation, while also testing that dependendencies are called, we can create dummy classes called ‘mocks’. In this project, mocks are created for us using the ‘Mockito’ plugin.

4.5. Build Automation

Build Automation tools are used to automatically compile, verify and package software development projects, and to simplify the management of project dependencies. They remove the repetition and risk of error associated with performing these tasks manually, and speed up the build process (saving time and therefore money).

This project is written in Java, so we are using a Java-based build automation tool called Maven. In Maven, the project description, its dependencies, and the order of the build is described in the 'Project Object Model' (POM) XML file. Maven automatically downloads the required dependencies from a central repository, which means developers do not have to store these locally. The POM also serves as a documentation of the project dependencies.

Maven also allows you to build executable JAR files, which contain the source code and all required resources and dependencies for your projects, making it easier to deliver to stakeholders, and allowing them to be deployed across different environments. The attached source code for this project has been packaged as a JAR using Maven.

4.6. Continuous Delivery

Continuous Delivery (CD) extends the concept of CI, and includes building, verifying and deploying your codebase on a regular basis (often each time a change is 'committed' to develop). The aim is to create confidence in successful code builds, and to allow the early detection of problems. It also supports the 'Agile Manifesto' principle of customer satisfaction through 'early and continuous delivery of valuable software' by ensuring that code is ready for release into production at all times (Beck et al., 2001).

To ensure CD to MMG, and to speed up the deployment process (saving us resources and money) we will use a use the CI tool Jenkins (Jenkins, no date) with Maven and configure it to automatically build, test and deploy our code each time a developer commits their code to the develop branch (as per the Gitflow branching model). The developer will then receive a notification (via Jenkins) if the subsequent build is unsuccessful, so they can prioritise a fix. We will use the reporting tools in Jenkins to provide regular updates to MMG.

5. Evaluation

Most of the technology used in this project had not been used by MMU Corp previously (in particular, the Java programming language and the Spring Boot framework). Despite this, we have been able to create a functioning web application while incorporating many of the principles of 'good' software design and agile development.

However, the functionality of our site in its current state is basic, and given the amount of time that was required to become familiar with the new technologies, we were not able to develop all of the features requested in the time allocated for this initial development phase. The learning process also required writing and running code first, to getter a better understanding of how the technology works. Thus we did not carry out genuine TDD. The tests in the current source code package were written later, and time constraints meant we were unable to go back and add a full suite of tests. Therefore, it may have been better, for this particular project, to make use of technologies that we were already familiar with it. However, I do feel that taking the extra time to learn Java and Spring Boot has greatly helped in furthering our understanding of SOLID design principles and object oriented programming.

Developing the code first also meant that assumptions were made about required functionality before the requirements gathering and design processes were complete. For example, it was assumed that the 'singletons in your area' search would be based on a location input by the user in their profile, and the implementation was started based on that assumption. However when going back to complete user stories with MMG it became apparent that this approach could cause a number of complications, and that there may be alternative solutions that would simplify the code, and better meet the business needs (e.g. obtaining a location from a user's device). Had we gone on to fully develop the search feature before consulting the stakeholder, significant resources could have been wasted. This issue reinforces the importance of seeking to understand the needs of the business when gathering requirements, and of practising agile development processes to successfully manage changing requirements. However, now that we have a basic functioning product in place, we are in a better position to move forward with the project, while making full use of the agile development methodology.

Given the limited functionality of the project in its current state it could also be argued that the addition of a separate business layer to house business logic (services) and business entities (models) is excessive, and creates unnecessary complexity in the code. However, as the application grows, the need for the Business Layer is likely to become more apparent in ensuring the security of business data, and loose coupling between the business logic and data access methods.

References

Atlassian. (no date) *Gitflow Workflow.* [Online] [Accessed on 4th December 2019] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001) *Manifesto for Agile Software Development.* [Online] [Accessed 29 Apr. 2015] http://www.agilemanifesto.org/

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley.

Ghahrai, A. (2018) *Software Development Life Cycle – SDLC Phases.* Testing Excellence. [Online] [Accessed on 9th December 2019] https://www.testingexcellence.com/software-development-life-cycle-sdlc-phases/

Jenkins. (no date) *Jenkins. Build great things at any scale.* [Online] [Accessed on 25th October 2019] https://jenkins.io/

Scrum.org. (2019) *Welcome to the Home of Scrum!* [Online] [Accessed on 6th December 2019] https://www.scrum.org/

Spring. (no date) *Spring Initializr. Bootstrap your application.* [Online] [Accessed on 29th September 2019] https://start.spring.io/

Appendix 1: User Stories

Story Description 1

As an unregistered user I want to create a profile so I can connect with other single software developers.

Example Acceptance Criteria

When created, user profiles should be displayed to other users in the same area.

Story Description 2

As a singleton I want to let others know that I like them so that I increase my chance of connecting with them.

Example Acceptance Criteria

Users should be notified when their profile is liked.

Story Description 3

As a singleton, I want to know when someone likes me so that I know who I have the best chance of connecting with.

Example Acceptance Criteria

Users should be notified when someone likes their profile.

Story Description 4

As a singleton I want to view the contact details of people I like who also like me back so that we can connect.

Example Acceptance Criteria

Contact details (e.g. email address) should be valid

Story Description 5

As a singleton I want to be able to update my profile so that I always have the chance of finding the best possible match.

Example Acceptance Criteria

Updated details should be visible to other users immediately.

Story Description 6

As a singleton I want to be able to delete my profile so that my details are no longer visible should I decide I no longer want to use the website.

Example Acceptance Criteria

A deleted profile should no longer be visible to other users.

Appendix 2 : Building and Running the Source Code

The attached zip file (alternative Google Drive link here) contains the 'merge' directory which is a Maven project packaged as an executable JAR. It should therefore contain all of the resources associated with the project.

The process of building and running the source code may depend on the IDE. In IntelliJ:

- Go to File-->New-->Project from Existing Sources.
- Import as a Maven project.
- Navigate to the 'merge' directory.
- In the project file structure, navigate to the main class 'MergeApplication'.
- Right click and select 'Run'. Open a browser and navigate to http://localhost:8080 to view the site.