Project v1.2

CS2901 Software Engineering I

After collecting and specifying requirements, it is time to conceptualize the v2.0.

1 Summary of Milestones

Milestone	**Minimum acceptable performance to consider as 'reached'**
Requirements documented	a draft of v2.0 requirements
A draft of User Guide	Follow Section 3.2.1
A draft of Developer Guide	Follow Section 3.2.2
Feature releases planned	a rough feature release plan
Components Diagrams	diagram of components to be implemented

2 Summary of Deadlines

2.1 Mock-up Presentation: Wednesday, 2nd October

You should consider into your presentation the following items:

- List of refined requirements based on feedback.
- Updated Mock-up according to refined requirements.
- Feature Release Plan (Section 3.1).

2.2 Documentation v1.2: Wednesday, 2nd October

Based on the feedback of the client, you should update your current documentation in your project repo. You should update in your repo all items described in Section 1.

2.3 Discussion Session: Thursday and Friday, 3rd-4th October

During our scheduled classes, we will discuss wich each team all items described in Section 3.2.2. We will announce the timetable in the coming days.

3 v2.0 Conceptualized

Given that requirements have been specified and we have validated them against stakeholders, you are ready to conceptualize your product in its last version, named as v2.0.

3.1 Feature Release Plan

You should elaborate a Feature Release Plan where the following points are considered:

- After the v2.0 is conceptualized, decide which features each member will do by v1.4. It is better to start with some plan rather than no plan at all. If in doubt, choose to do less than more; we don't expect you to deliver a lot of big features.
- Divided each of those features into three increments, to be released at v1.2, v1.3, v1.4 and v2.0. Each increment should deliver a end-user visible enhancement.

3.2 Documentation

Recommended procedure for updating docs:

- Divide among yourselves who will update which parts of the document(s).
- Update the team repo.
- Use markdown format for UG and DG documents.

3.2.1 User Guide (UG)

Start moving your requirements specification into the User Guide page in your repository. You should follow the below structure:

1. **Introduction.** Describe briefly your project.
2. **Requirements.** Enlist your functional and nonfuntional requirements.
   (a) Functional
   (b) Non-Functional
3. **Features.** Enlist your features.
   (a) Viewing help: `help`
   (b) Adding a person: `add`
   (c) etc ...
4. **User Stories** Include your use stories. Do not forget to tag them as *must-have* and *nice-to-have*
5. **Use Case** Include your use cases diagrams using UML.
6. **Glossary**

3.2.2 Developer Guide (DG)

Similar to the User Guide, we start working on the Developer Guide in your team repository.

The main objective of DG is to explain the implementation to a future developer, but a hidden objective is to show evidence that you can document deeply-technical content using prose, examples, diagrams, code snippets, etc. appropriately. To that end, you may also describe features that you plan to implement in the future, even beyond v2.0 (hypothetically).

You should folllow the next structure:

1. **Introduction.** Similar to the User Guide, describe briefly your project.
2. **Features.** Enlist your features. For DG, the description can contain things such as:
   - How the feature is implemented.
   - Why it is implemented that way.
   - Alternatives considered.
3. **Design.**
   (a) **Architecture**
   - **Class Diagrams** describe the structure (but not the behavior) of an OOP solution. These are possibly the most often used diagrams in the industry and an indispensable tool for an OO programmer. Figure 1 shows an example of a class diagram.

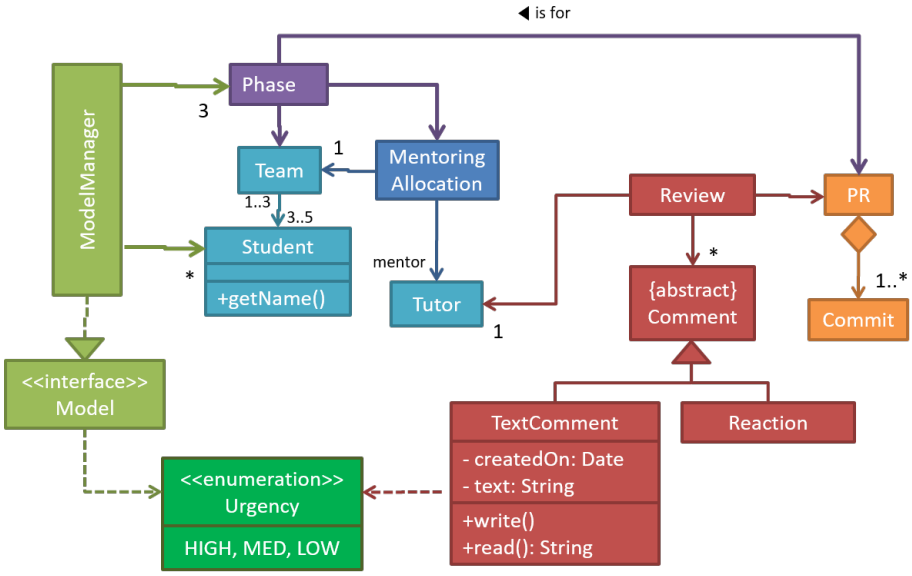

Fig. 1: An example of a class diagram

- **Architecture Diagram.**This diagram explains the high-level design of the App. The Architecture Diagram is a quick overview

of each component. Figure 2 shows an example of an architecture diagram.

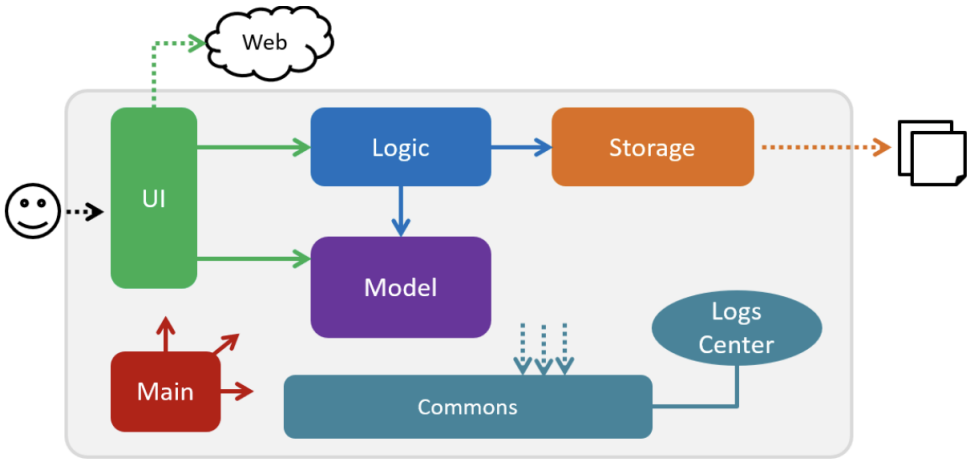

Fig. 2: An example of an architecture diagram

- **Component A.**
  i. Description of Component A.
  ii. A class diagram of Component A. Follow the example shown in Fig. 3.

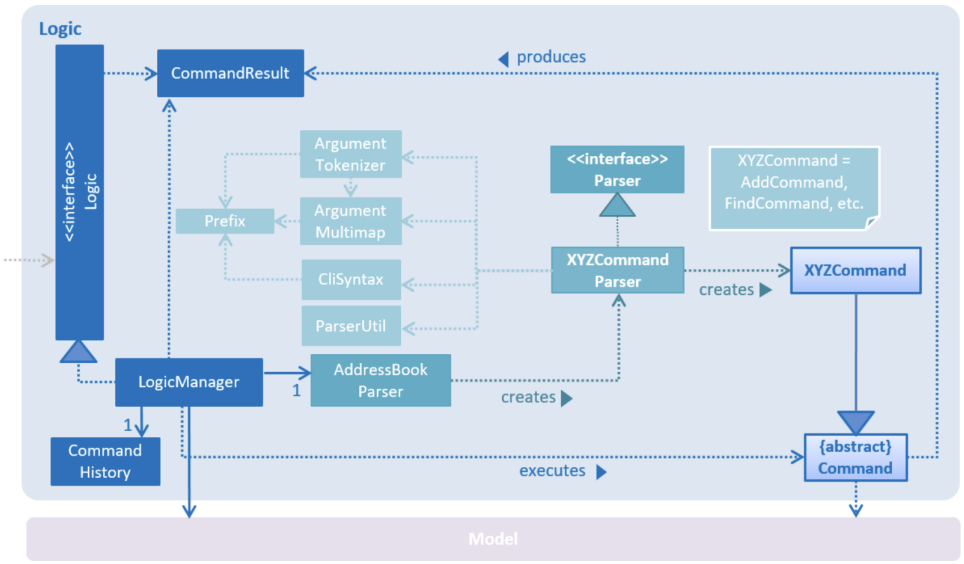

Fig. 3: An example of a class diagram of the Logic component

- **Interaction between Components.** In this section, you shoud add sequence diagrams that show how the components interact

with each other for different scenarios (minimum 3 scenarios) of your project.

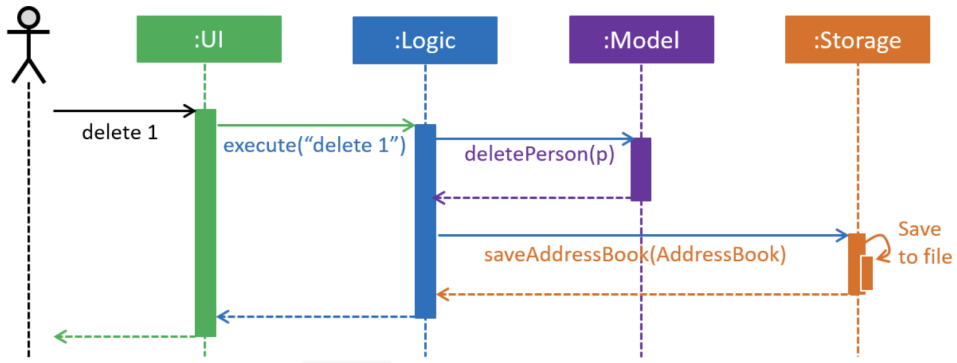

Fig. 4: An example of a sequence diagram for a particular command

4. **Glossary**