2DV609

PriTask

Design Document

Author1: Armend Azemi

Author2: Adam Nyman

Author3: Chadi Kawaf

Author4: Donald Nti Appiah-Kubi

Author5: Oliver Rimmi

1. Purpose

This document describes the goals and philosophy, assumptions and dependencies,decisions, constraints, justifications, significant requirements, architecture mechanism and any other overall aspects of the system that shape the design and implementation of PriTask.

PriTask is a form of task planner with its main purpose being that the user may add their tasks and have them organized in such a way that the user knows what tasks are the most important due to the priority sorting algorithm.

2. Architectural goals and philosophy

The main points for our product regarding the architecture is to make an application that will have low coupling and components that are easy to reuse for future projects. Considering that our application is going to be monolithic, thus making it lightweight and simple, we are not expecting any issues with the deployment process and neither do we expect any performance issues therefore the goals regarding these two issues have not been addressed. Long term maintenance is not a major issue as we see it since the application is going to be structured in a way that allows for low coupling thus making the application flexible to future change.

The goals we have for the application are directly related to the previously mentioned facts about the application and the goals are also directly related to the issues we have considered. Some of the issues are to avoid a high system load, hard navigation using the UI, complicated and high coupling source code and conforming to a wide set of operating systems.

3. Assumptions and dependencies

Assumptions:

- The application will be monolithic.
- The application will be deployed locally.
- The application will be created using Java. Java is currently one of the most popular and most widely used programming languages . Hence, we believe that the use of Java will allow the developers to access a greater amount of documentation and support compared to what they would by using another framework. Additionally, Java supports a wide variety of frameworks and has a lot of third party libraries that could turn out to be very useful during the development of this application.
- Some third party libraries that we will use are Gson and Maven.

Dependencies:

- Developers knowledge and experience. All of our developers have experience with Java. Thus, we believe that the development process will not be slowed down due to the lack of knowledge and experience.

4. Architecturally significant requirements

Some of the most important architectural requirements are found in the requirements document. During the development of this document several requirements were thought of and considered for this section of the document, they are:

- UI1: Simple user interface
- SYS7: The application should be lightweight
- SYS8: OS Compatible

Taking into consideration that this service will be offered as a desktop solution and that the content found inside the service is not sensitive we did not consider that security was of that high importance.

We also took into consideration some of the functional requirements as well, they are the following:

- SYS1: Creating a list
- SYS2: Removing a list
- SYS3: Creating a task
- SYS4: Edit a task
- SYS5: Delete a task
- SYS6: Priority.
- DB1: Persistence.

Although these requirements are of very high importance they do not, in our opinion, have a significant impact on the architecture of the service that we intend to provide. The main thing that we will focus on regarding our architecture is that we have a reliable and stable foundation to add our desired functionality, thus making the functional requirements of less importance regarding the overall architecture structure.

The table below is intended to show what requirements are of the utmost importance for our product.

Requirement	Priority
OS compatible	Very High
Simple user interface	High
The application should be lightweight	Moderate

When examining the table we can see that the priority of the non functional requirements differ. OS compatibility has the highest priority since we can’t anticipate the end user's OS, thus making it available to a wide range of operating systems we ensure that the user is able to run the application. A simple user interface is also important since we want the user's experience to be smooth and make sure that the UI is intuitive, but considering the previously mentioned requirement if the user can’t run the application due to the application not being compatible with a certain OS we deem this requirement to be of a slightly lower priority. And the requirement that has the lowest priority is that the application should be lightweight, please refer back to the Requirements Document for a specific definition of this requirement, this is due to the fact that the expected size of the application is going to be relatively small thus making it a bit less important.

5. Decisions, constraints and justifications

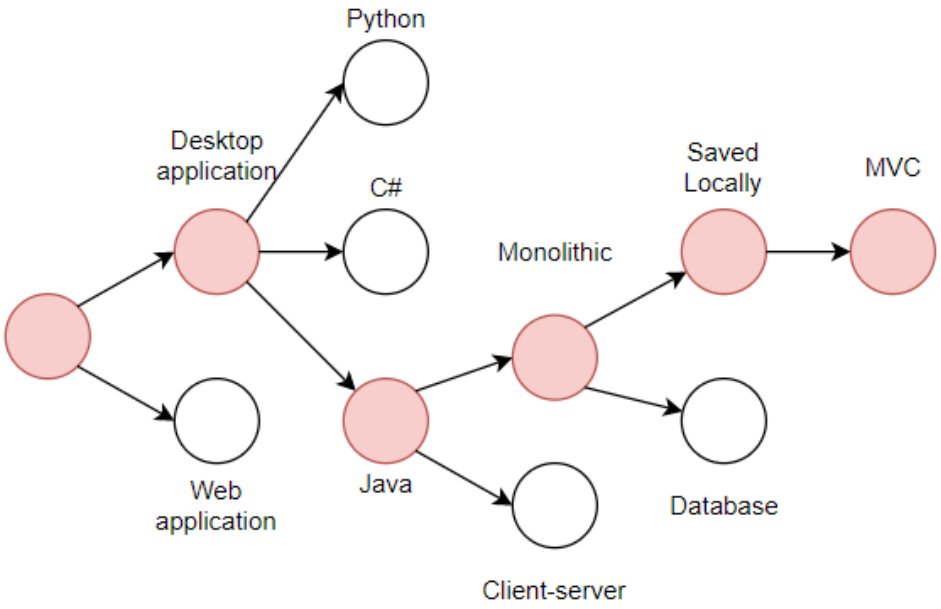

Figure 1. *The figure shows the decisions we made during this project.*

- The architecture should allow for high reusability. We used the MVC architecture to make it easy to create new UIs while keeping the main functionality unchanged.
- The MVC architecture also allows for lower coupling. This means that the impact of changing the main functionality is minimal. E.g., changing the algorithm for sorting the tasks by priority (which is placed in the model) should not incur any required changes in either the controller, or the view.
- We decided to implement a desktop application in Java instead of an Angular application due to our previous experience with Java.
- We decided to make the application monolithic because the application is not intended to contain information with a high importance for the end user. This means that potential data loss due to, e.g., hardware failures, will not leave the end user devastated. Therefore it would be unnecessary to implement a database.
- The performance should be efficient due to the small size of the application.
- We decided to store the data from the application locally, in the same folder as the application project, by parsing the application data to JSON format and storing it in a simple text file using Gson. We chose this approach because of previous experience using JSON and the ease of use.

6. Architectural Mechanisms

6.1. Architectural Mechanism 1

The main architecture pattern to be used is the Model-View-Controller pattern as inspiration for the design. The alternative to this was to use the Observer Pattern for interactions between the view and the rest of the system which also seemed to be a common pattern for this sort of system, but as a majority of the group members has more experience working with the MVC-pattern therefore this was the favourable option to use. This design mechanism will let the system have an overall low coupling, high reusability and maintainability.

7. Key abstractions

Since what makes this system unique is the fact that it should perform priority sorting on different tasks. Therefore the most important key abstractions of the system will surround this functionality. Therefore the following abstractions are probably the most essential abstractions for the system:

- **Task:** The idea of the task object is that it should contain some of the key information about the task that the user has provided. This includes a completion date, a priority value as well as a brief description of the task itself.
- **Tasklist:** This is another object that is directly managed by the user. It is a list of tasks that the user wants to order after priority. The user could for example have a tasklist for school assignments,and another for houseshores.
- **PriTaskManager:** This is a placeholder class for the PriList objects and works somewhat like a facade in the sense of hiding some of the complexity within the model.

8. Architectural views

[Describe the architectural views that you will use to **describe the software architecture**. This illustrates the **different perspectives** that you will make available to review and to **document architectural decisions**.]

8.1. Recommended views

- **Logical:** Since the application should follow an MVC architecture pattern we have a few packages given. Namely, the model package, the view package and the controller package. The relationship between these packages is also given in the mvc pattern. Model should not be dependent on the view, any changes to the view that will affect the model package should go through the controller. The application will also have persistence built in. Therefore there also exists a package called persistence within the model package.
- **Operational:** The system is planned to be a single process without any additional threads.
- **Use case:** The only requirement that makes any major impact on the architecture is the fact that the data will be saved in a database.

To further illustrate our application we've created five UML diagrams. The figures below show different type topology of the system along with a brief description.

Deployment Diagram

The diagram below shows how the application will be deployed. However, since the PriTask application is a monolithic system everything will be deployed on a single computer. The user will deploy the application, with all its components, and the files created by the application will be saved locally at the same computer. Therefore no web servers, database nor any other form of likewise services will be needed to be deployed.

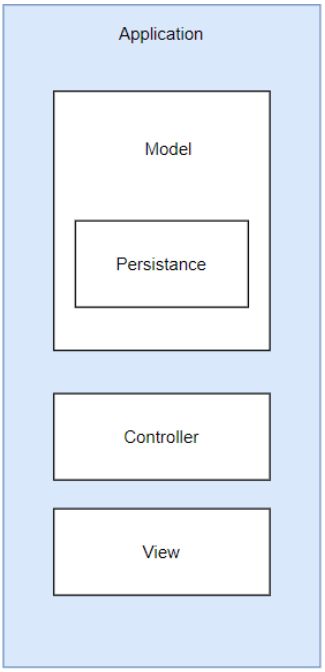

Figure 2: *This deployment diagram represents an overview of the deployment of the PriTask application. Since the application is monolithic everything is deployed on the users computer and no deployment of for instance web servers or databases are needed.*

Class diagram

The class diagram below shows the overall architecture of the application. It includes all of PriTasks classes as well as these classes' methods and functions.

Figure 3: *A class diagram of the application as a whole. Since the application is created using a MVC separation, we can see that it is divided into different packages representing the different parts.*

Sequence diagram

The sequence diagram below illustrates the sequences of which different actions occur within the system when the use case “SYS1: Creating a list” is executed by an user. As shown in the diagram, the user must press two different buttons for the entire process to execute. The application then updates the UI, model and saved file according to the document below.

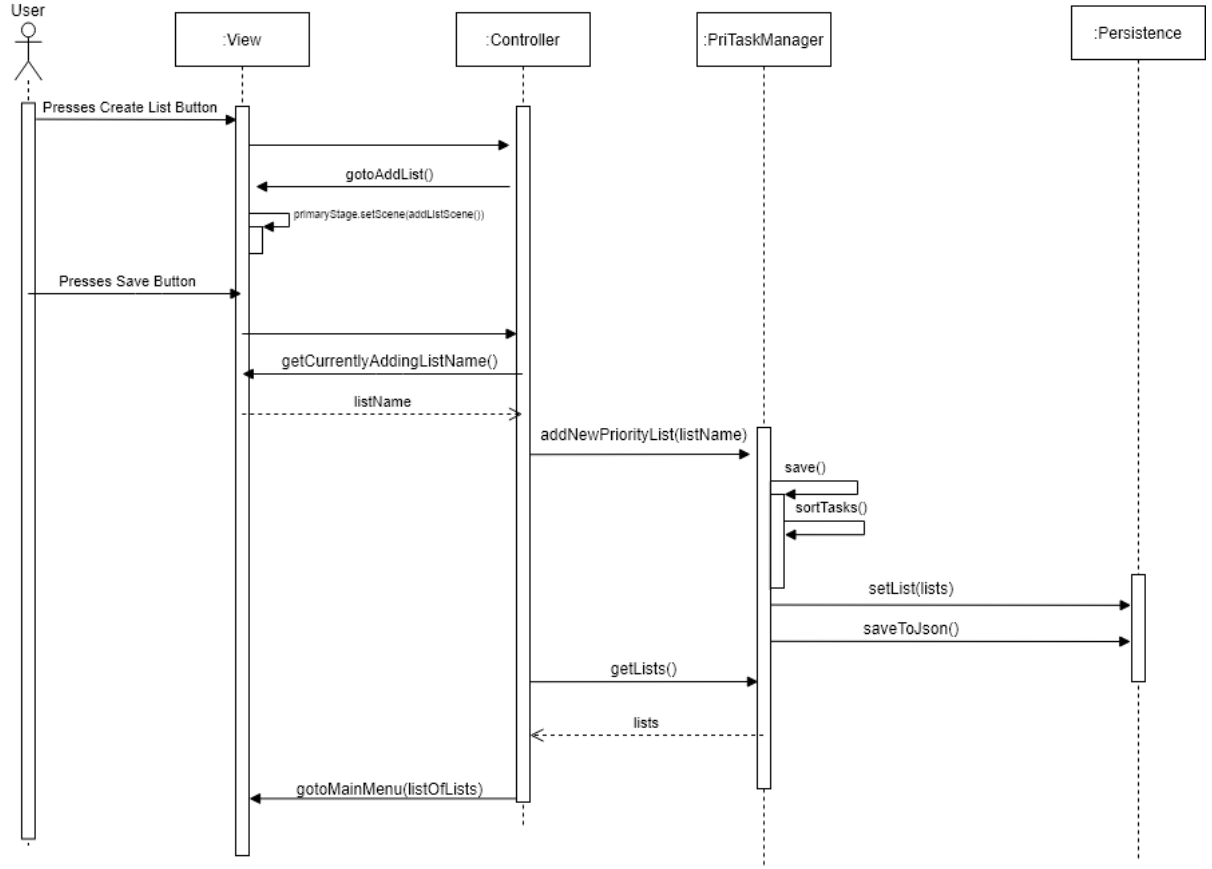

Figure 4: *A sequence diagram over the use case “SYS1: Creating a list”. It presents the sequences of actions that are taken when a user creates a new list using two button presses which then generates a new list of tasks within the application.*

Activity diagram

The activity diagram below shows the actions that the system takes when a user creates a new list following the use case “SYS1: Creating a list”. When the user inserts the input the system creates a new PriList object. The object is then saved locally as a JSon file. The system then presents the new information of the PriList object alongside any other already existing lists and tasks in the system.

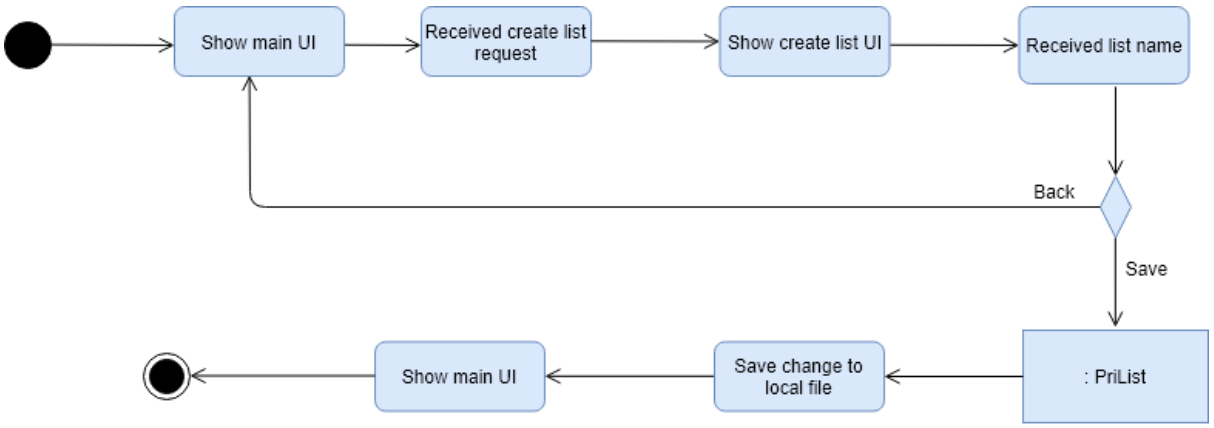

Figure 5: *Activity diagram showing the actions taken by the system when executing the use case “SYS1: Creating a list”.*