

**Table of Contents**

1 Process Model…………………………………………………………………....3

* 1. General…………………………………………………………………..3
  2. Specific……………………………………………………………….....4

2 Context Model…………………………………………………………………...5

3 Finite State Machine……………………………………………………………..6

4 UML Diagram…………………………………………………………………...7

5 Case View Model………………………………………………………………..8

5.1 Development View……………………………………………………...8

5.2 Logical View…………………………………………………………....8

5.3 Physical View…………………………………………………………...8

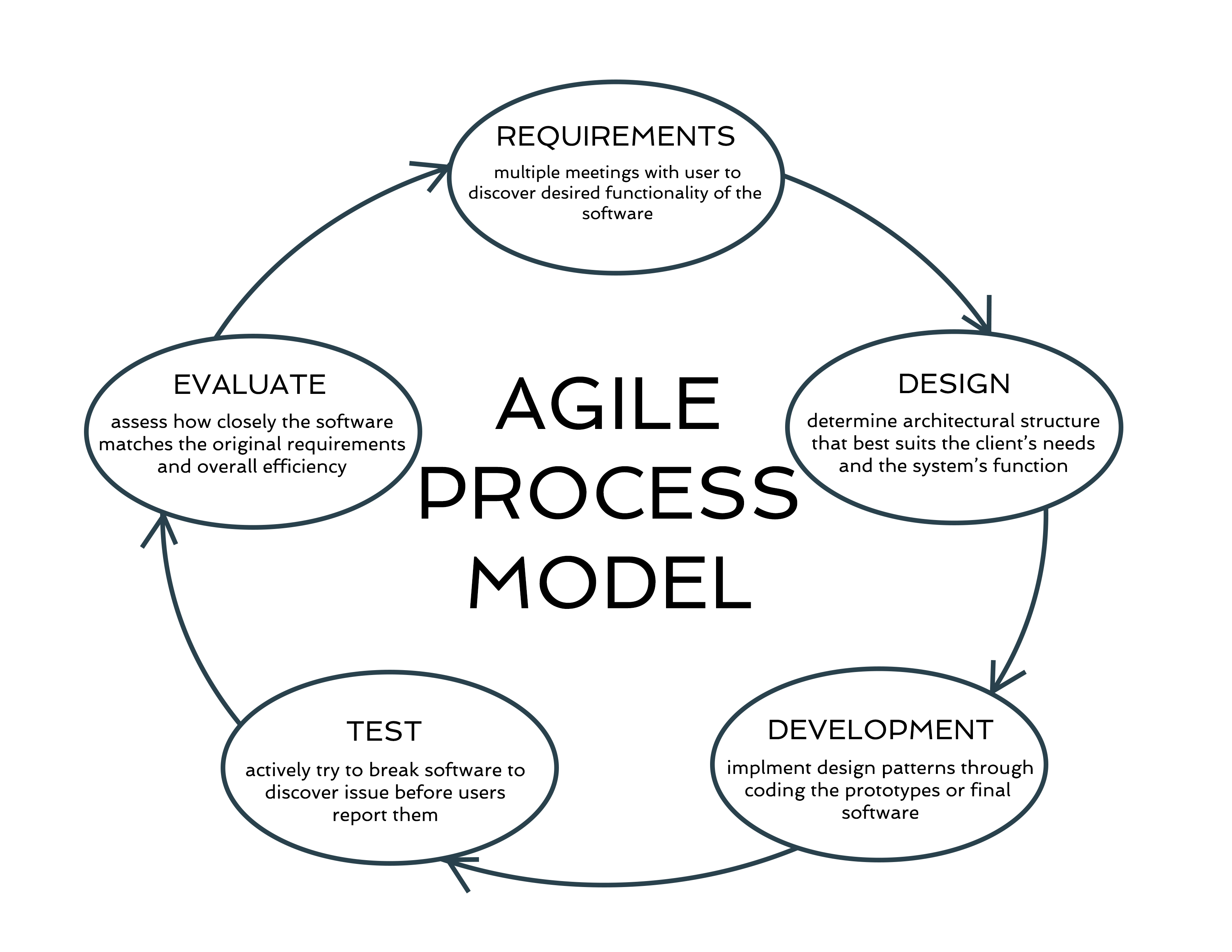
5.4 Process View……………………………………………………………8

6 Client-Server Pattern…………………………………………………………….9

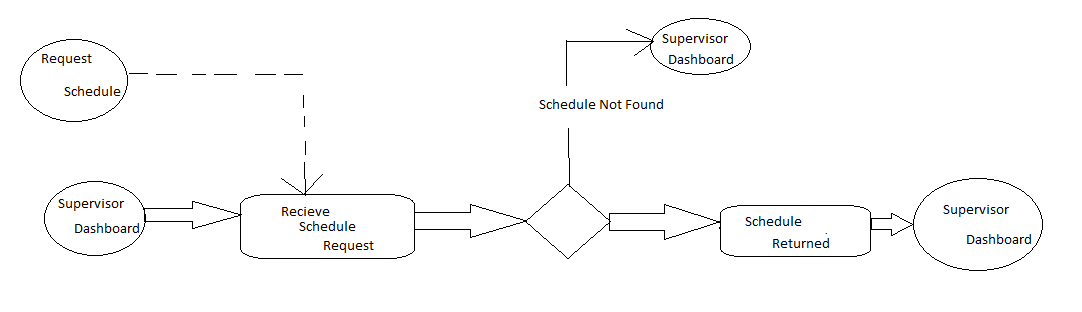
7 Afterthoughts…………………………………………………………………...10

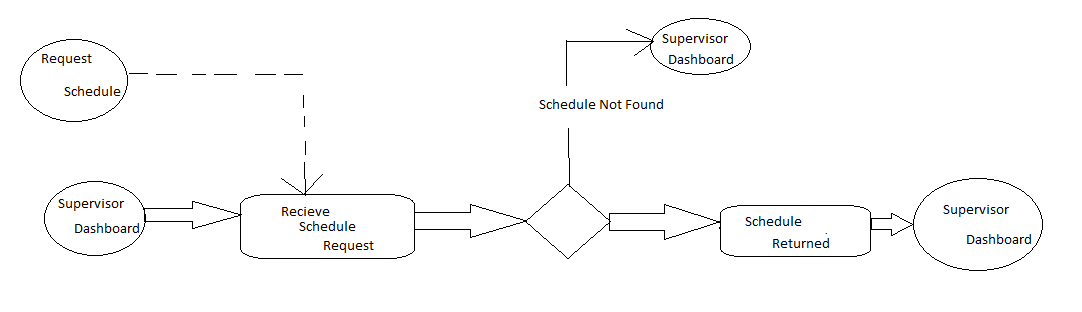
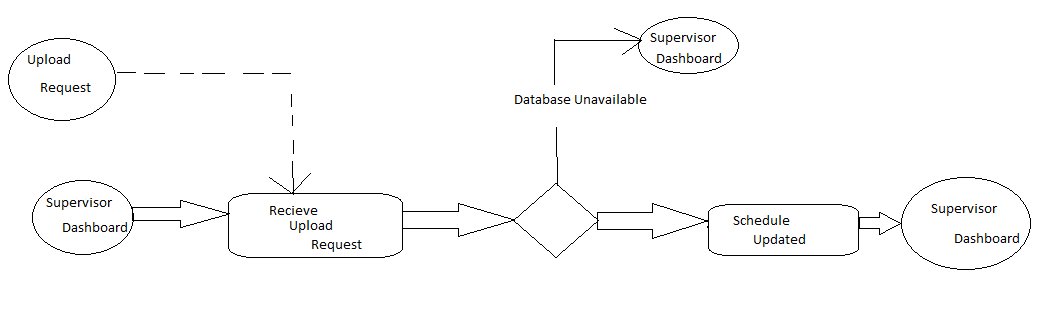
1. Process Model

1.1 General

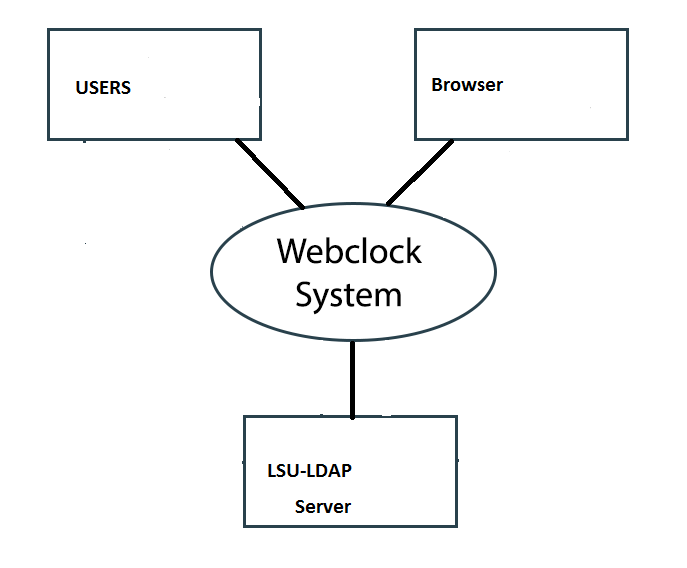


1.2 Specific

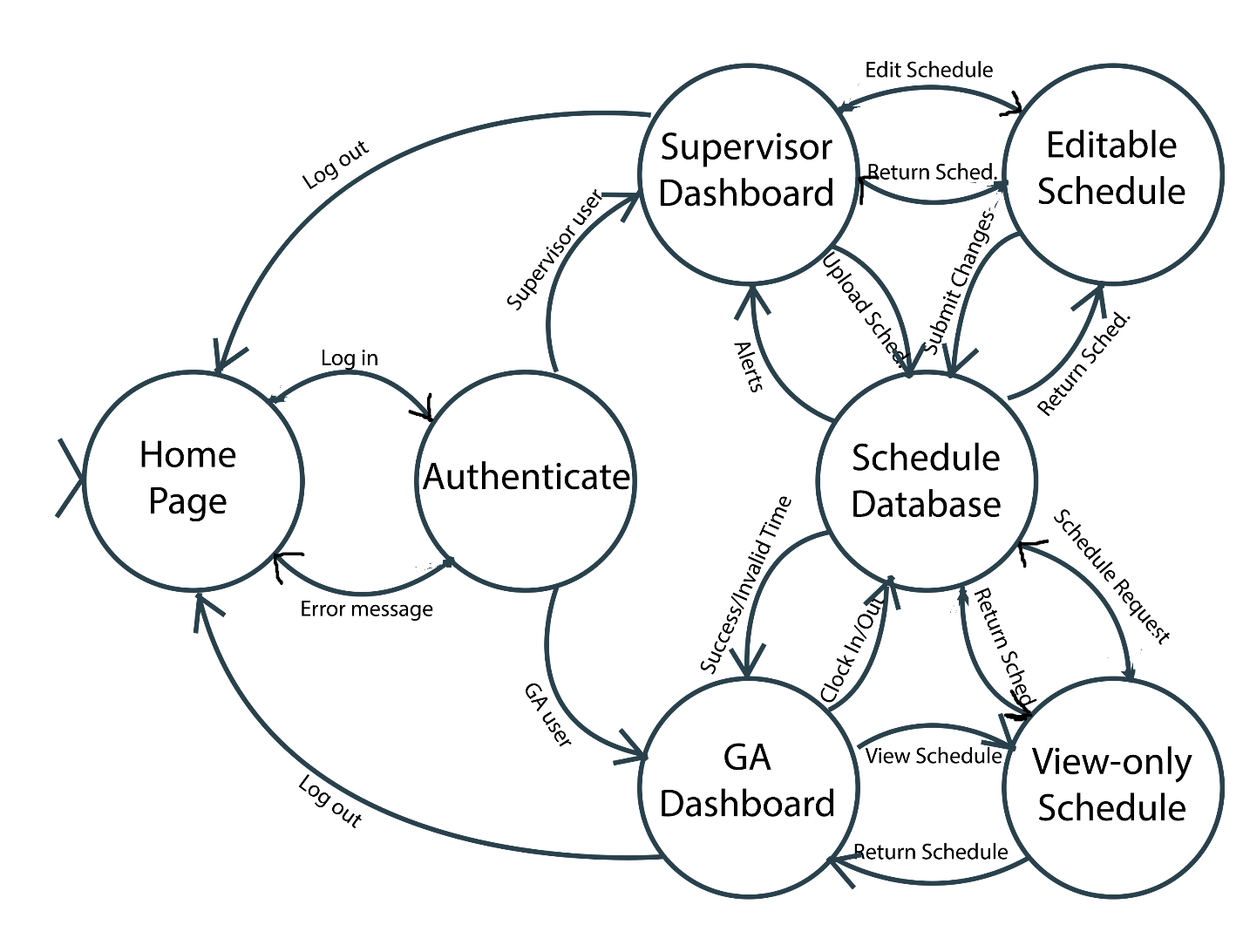




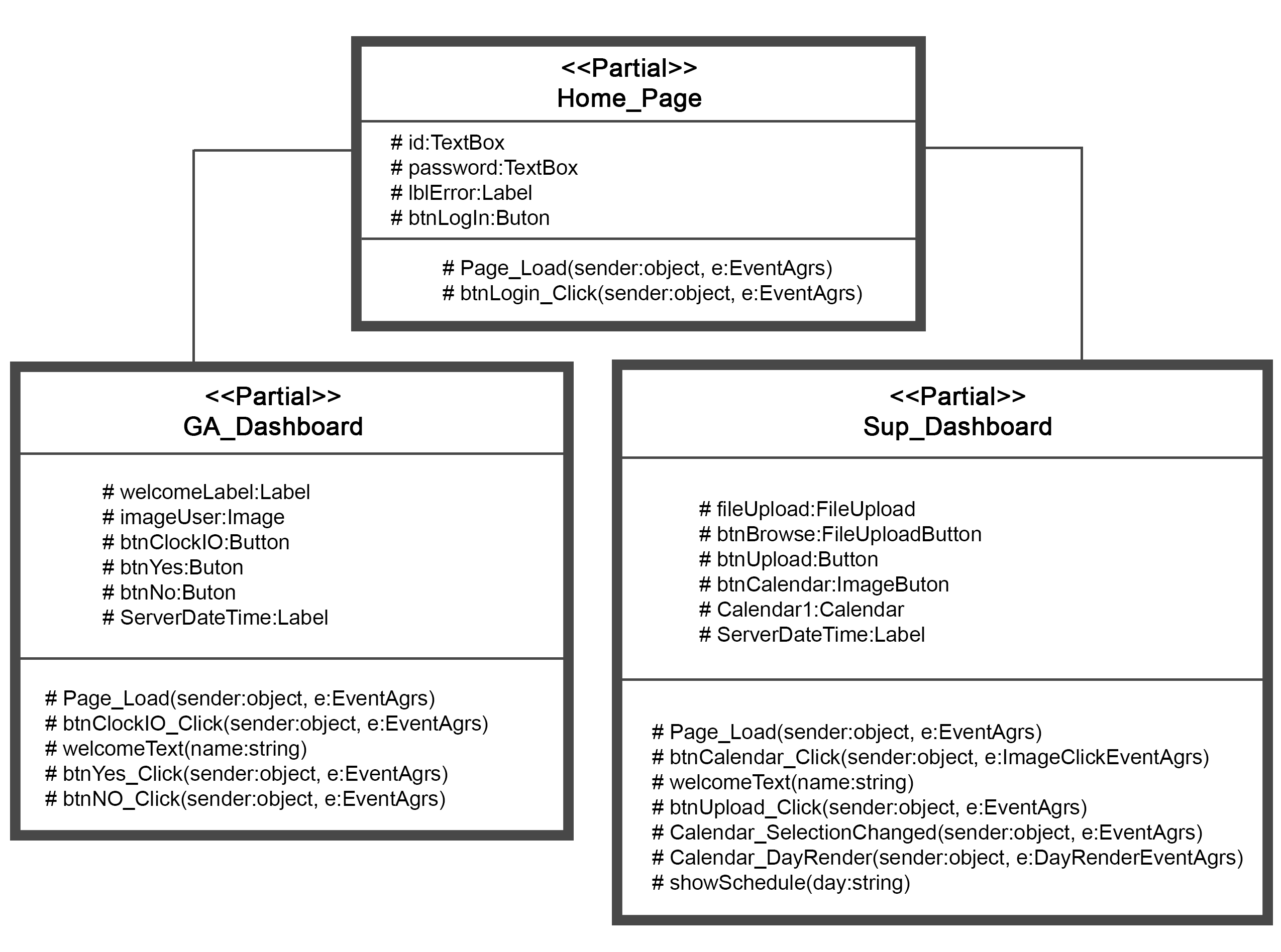
2. Context Model



3. Finite State Machine



4. UML Diagram



5. Case View Model

**5.1 Development View**

The development view can show us, the developers, or help us in the planning and design of the system. Looking at coding structure and what dependencies the system may have. The configuration of deliverables and what constraints the system as a whole would have. For our system we could look at browsers and OS the system would run on.

**5.2 Logical View**

In this view we are looking at the functional requirements of the system. It shows the interactions of collaborating classes that are key in the system. This view ultimately shows us end-user functionality of the system. For instance, in our system the logical view would be in say the ability of the user to log meal times.

**5.3 Physical View**

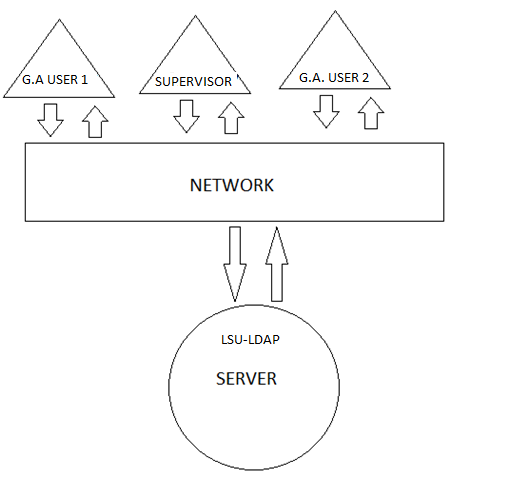
This view addresses how the hardware and software are working together. In other words how the data is stored. This would be low level where logical is high level. For instance when we log a user into the system the log, user name and time, is saved to a location which can be referenced later.

**5.4 Process View**

This view is looking at the interactions of systems or interaction of processes. It is looking at the run-time behavior. For our system this would be the checking of known usernames with the inputted ones and also the associated password of the entered username.

6. Client-Sever Pattern

Client server would work well for this system since we have multiple people accessing a database over the network.



Each Client being a lab G.A. and they are accessing the LSU server to verify their credentials

7. Afterthoughts

This assignment exposed us to versatile tools for working in a team of developers and modeling techniques for specification of viable software. The models were made in Adobe Photoshop and Windows Paint. Windows Paint is something that everyone is very familiar with due to its prevalence in Microsoft operating systems, and Photoshop tutorials are readily available on the internet. Photoshop being a very powerful image editing tool did have a steep learning curve, but this was overcome through diligent study and practice.

The UML diagrams were tedious in the amount of content required due to the many methods and variables inherent to this project, but overall they were less intellectually intensive than creating the finite state machine. The finite state machine posed the most difficulty because, while they simply followed the logic of the software, fitting this all into a single diagram was problematic. The easiest model to create was the context model. The context model was easy because it only involved four agents of which had to be addressed; the browser, the server, the client, and the webclock itself.

Obviously, these models, by design, help with the planning of the software. They are very useful tools to get an overview of the intricacies of the software and keep every team member on the same page. We feel that proper planning through the creation of such models ultimately results in a better final product.

The most helpful models were also the ones we found most difficult to create. The UML provides a microcosm of the software by detailing individual classes while the finite state machine demonstrates how the software will interact with the end-user and the software’s internal logic. We found the least helpful model to be the context model. In this situation, the context of the system is something that is so apparent it is almost trivial. From our combined experiences in developing web applications the information presented by this model was very redundant. However, given a larger system that is interworking with many other software or databases, we see how such a model could be useful in the design and implementation of some other software.

Git is a very useful tool for a team of software developers, but is also presents some issues. The repositories can be complicated to utilize due to the large amount of commands required. While these commands provided by the system are designed for the advantage of the users, to us, as new comers, the library was daunting. Given this, once one becomes comfortable with the interface and commands Git is a highly useful tool. If the program were to be corrupted in some way, Git provides a saved version of the program that can be utilized as a backup to restore your progress. The ability to comment on every uploaded version through Git’s messaging system allows the team members to work in tandem while being aware of others changes to the software. We would unanimously say that we would use some type version in the future given that the situation called for it.