CSCI 410-A
The Green Team
Architecture Assignment Document

Gabriel Windham
Danny Pham
Joey Green
Alex Thompson

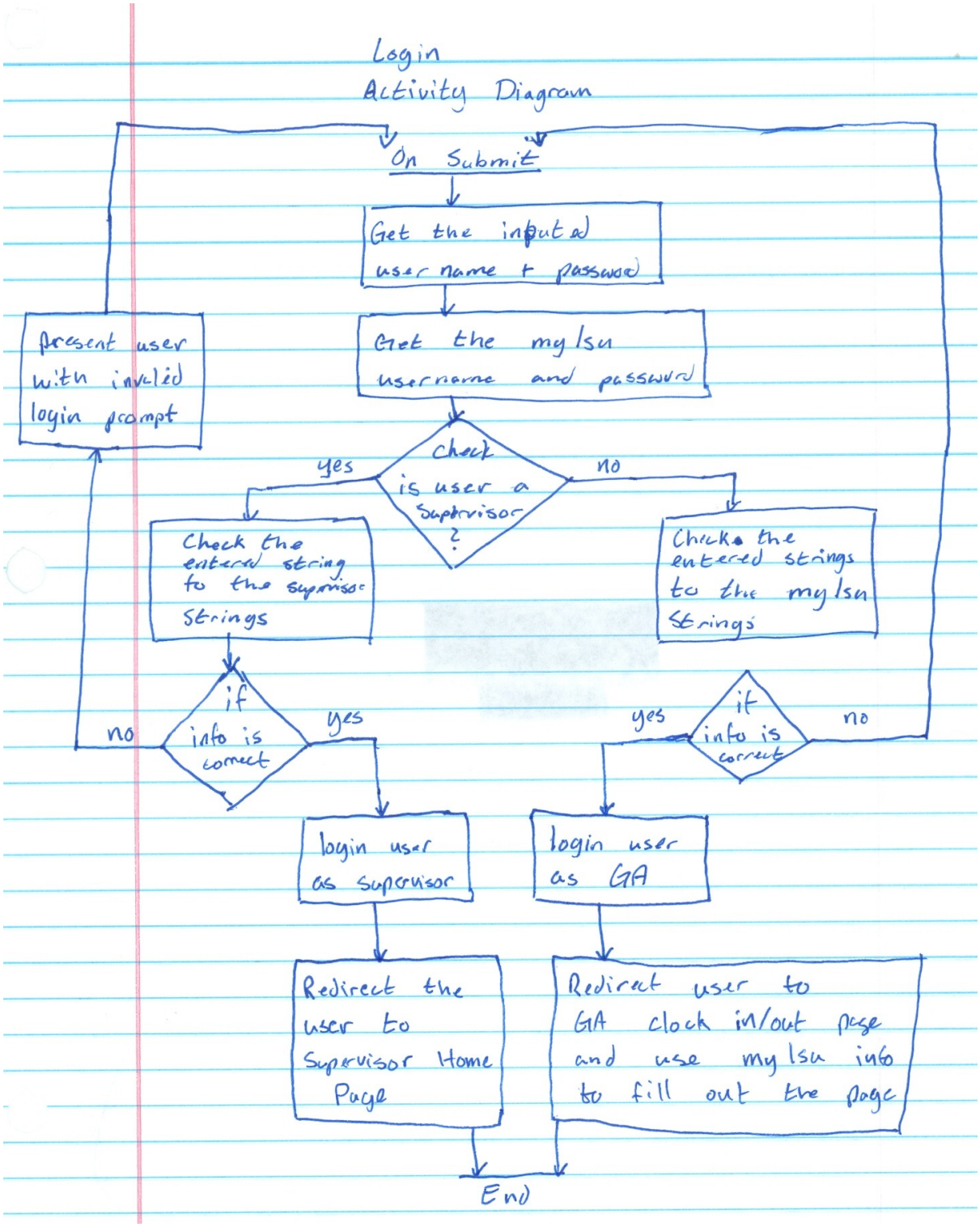
McNeese State University

November 30th,2016

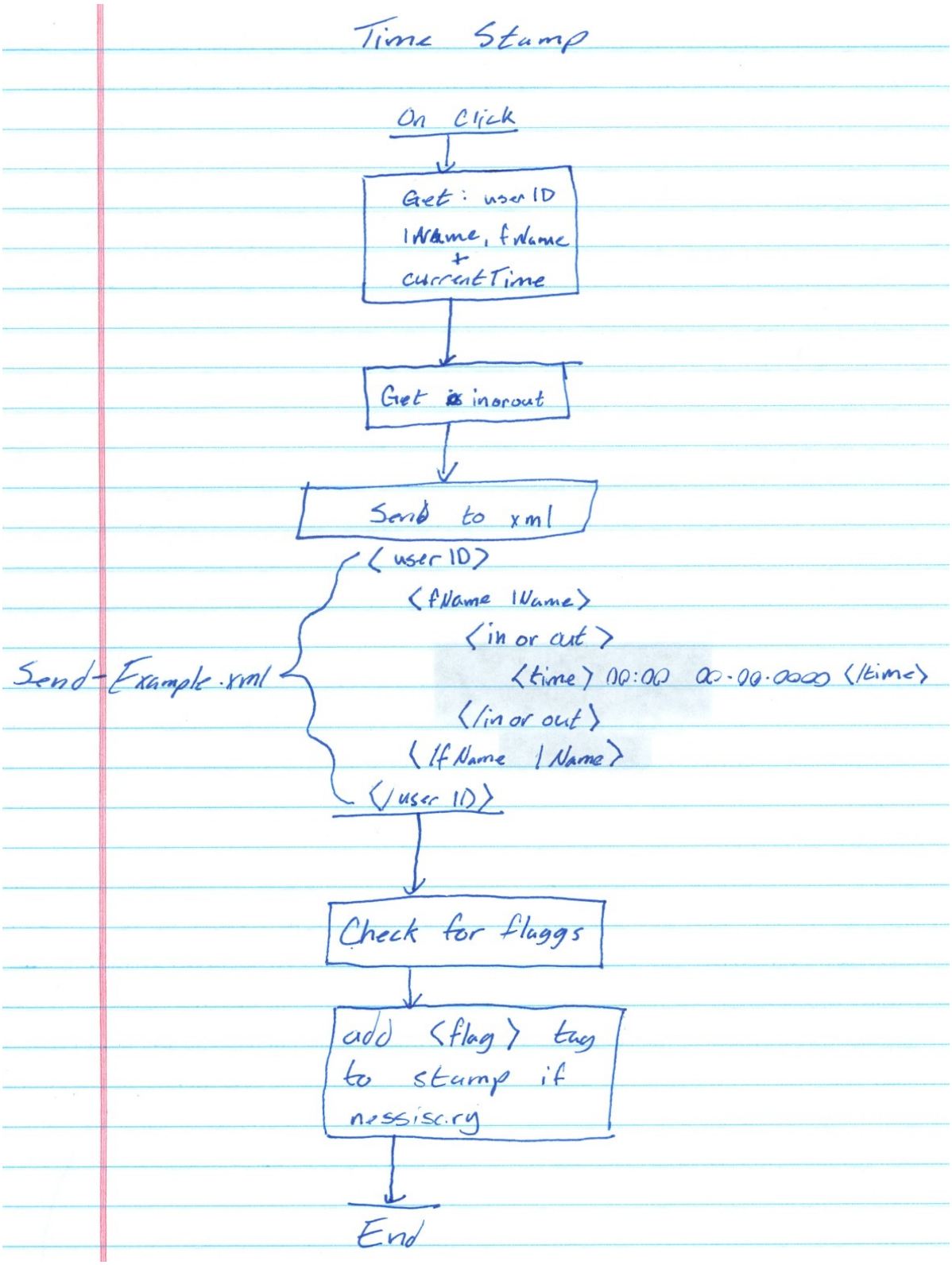
Contents

UML Class Diagrams

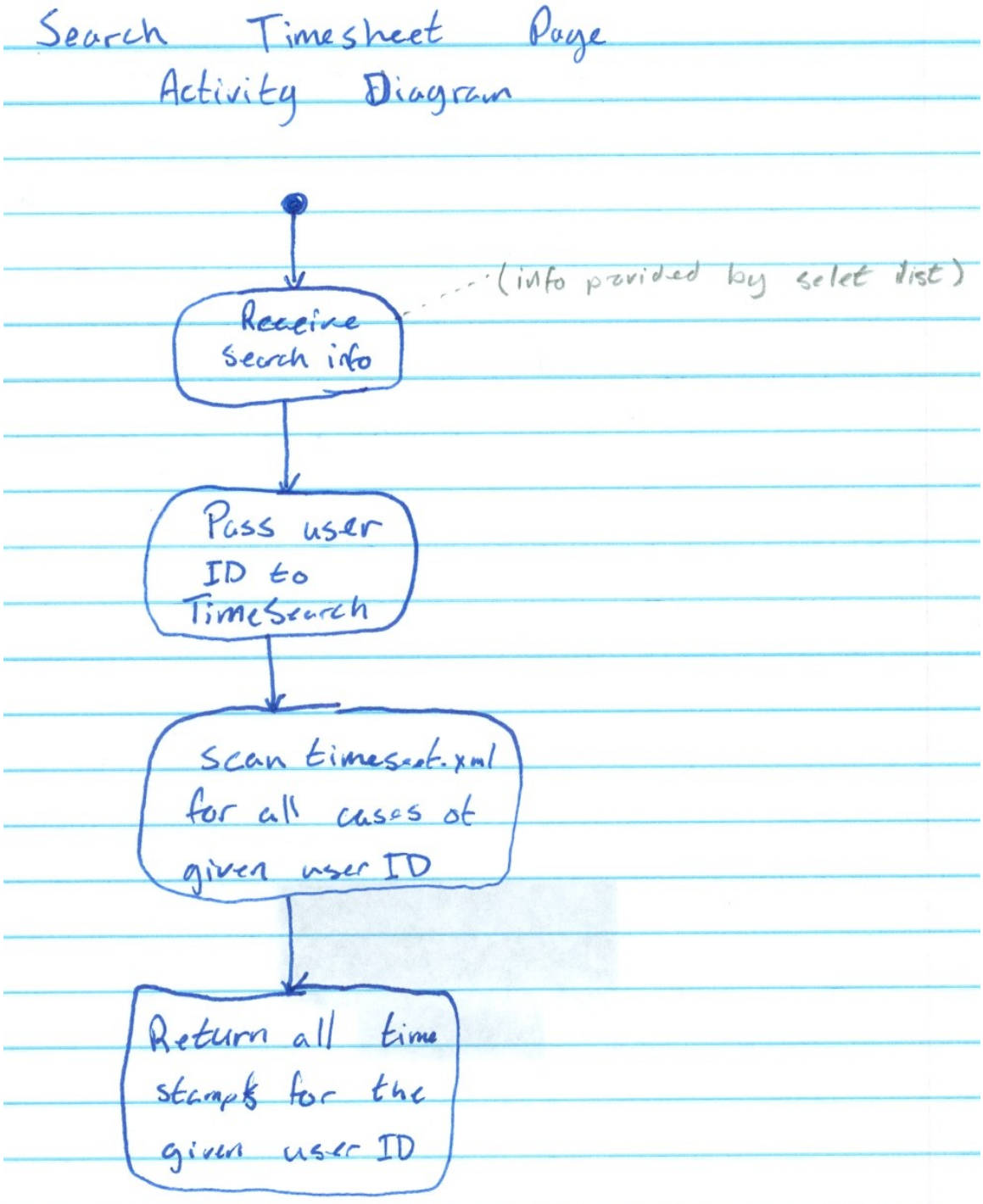
Login UML Diagram



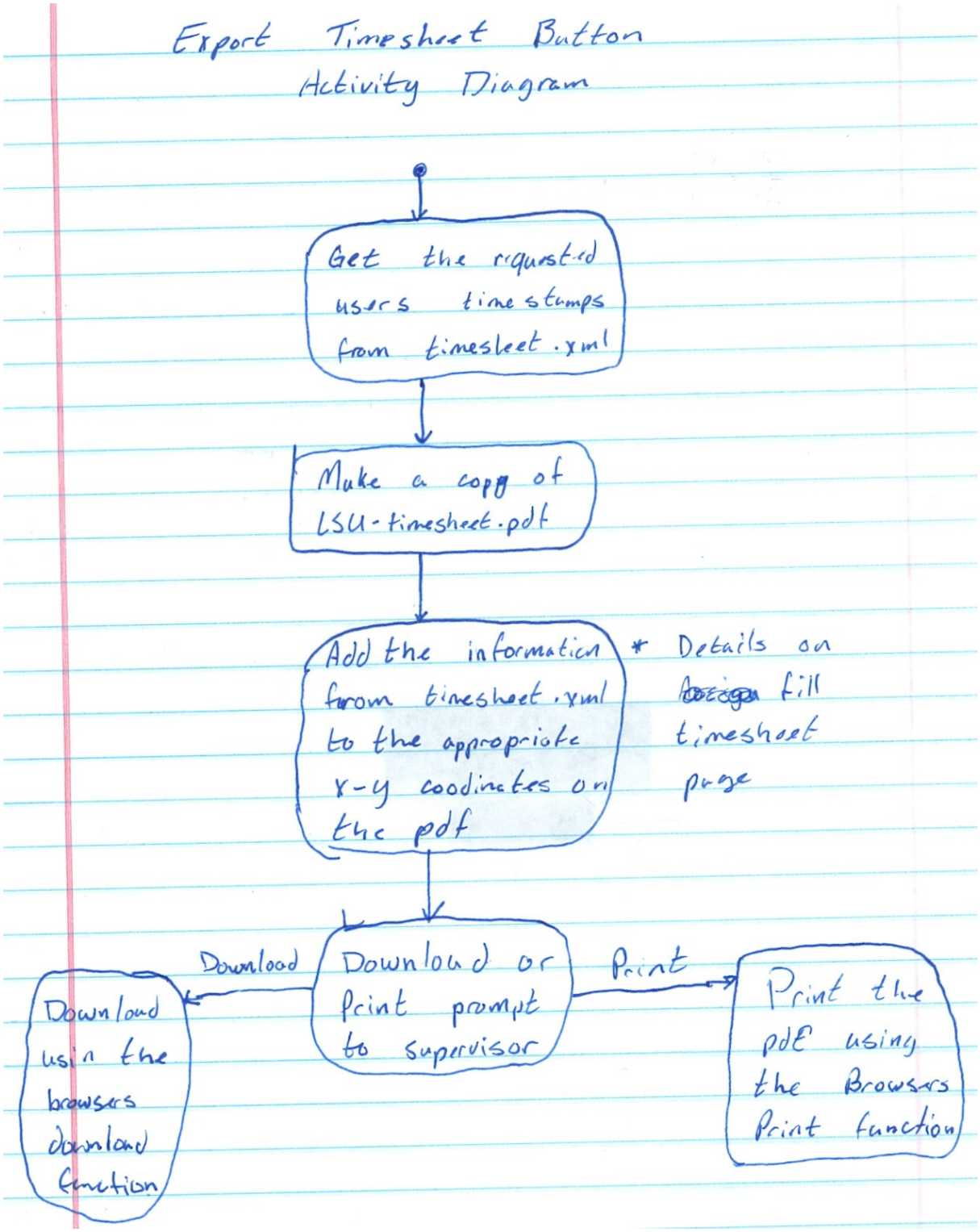
Time Stamp UML Diagram



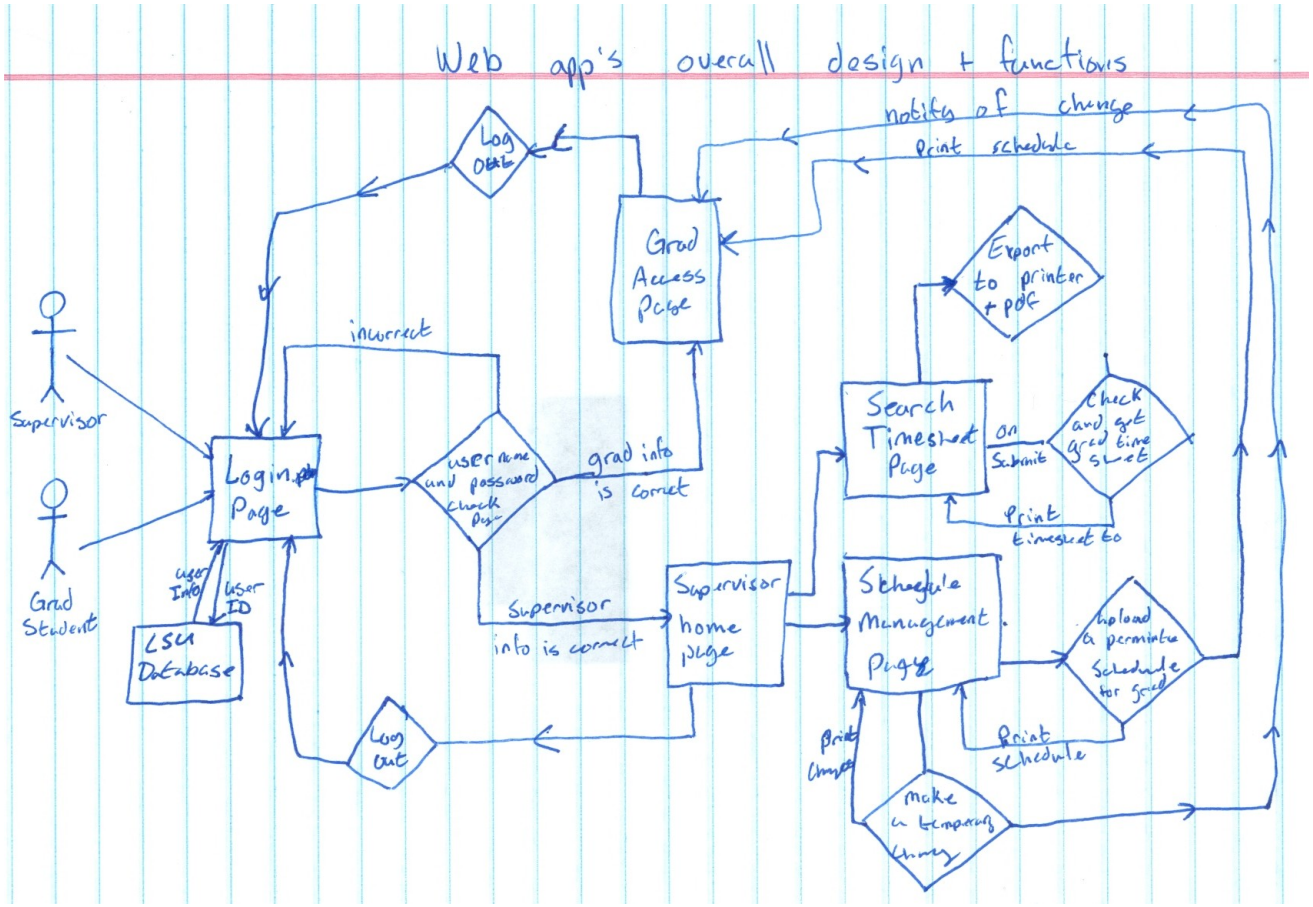
Search Time Sheets UML Diagram



Export Time Sheet UML Activity Diagram



WebFlow UML Diagram



The Four Process Views

The Logical View

This process view provides us with the ability to figure out the functional requirements of our system. For example, our requirement regarding the ability for users to enter their login information and then handle that data can be visualized through a logical process view. The logical view allows us to visualize our components into manageable objects. For instance, we can use a logical view to show the relationship between our requirement state above with our requirement to make the system be able to compute if the user is a graduate student or if they are the supervisor. Logical views can also organize theses objects into object classes to improve efficiency. Diving deeper into the logical view process model showed us a way to diagram our system into object interactions very similar to the relationship between the data entered by the users and who they are according to our system.

The Process View

This process view helps us visualize the entire system into a single visualization. Our single visualization for our project shows how the system works from the login page to when the user logs out. The visualization can single out subsystems and refine each one that is needed. This process can show us what would happen to the whole system if changes are made to a component in the system. If we wanted to add another component to add another layer of security we would move that subsystem before our graduate access page. Process views can also utilize UML diagrams (like activity diagrams) to show the static and dynamic aspects of the system. For example our time stamp cannot be changed by the graduate students, which would mean that the information is static to them. Although that information is dynamic for the supervisor if he or she needs to edit time stamps. Both of these examples can be used with the process view by using a UML for each scenario.

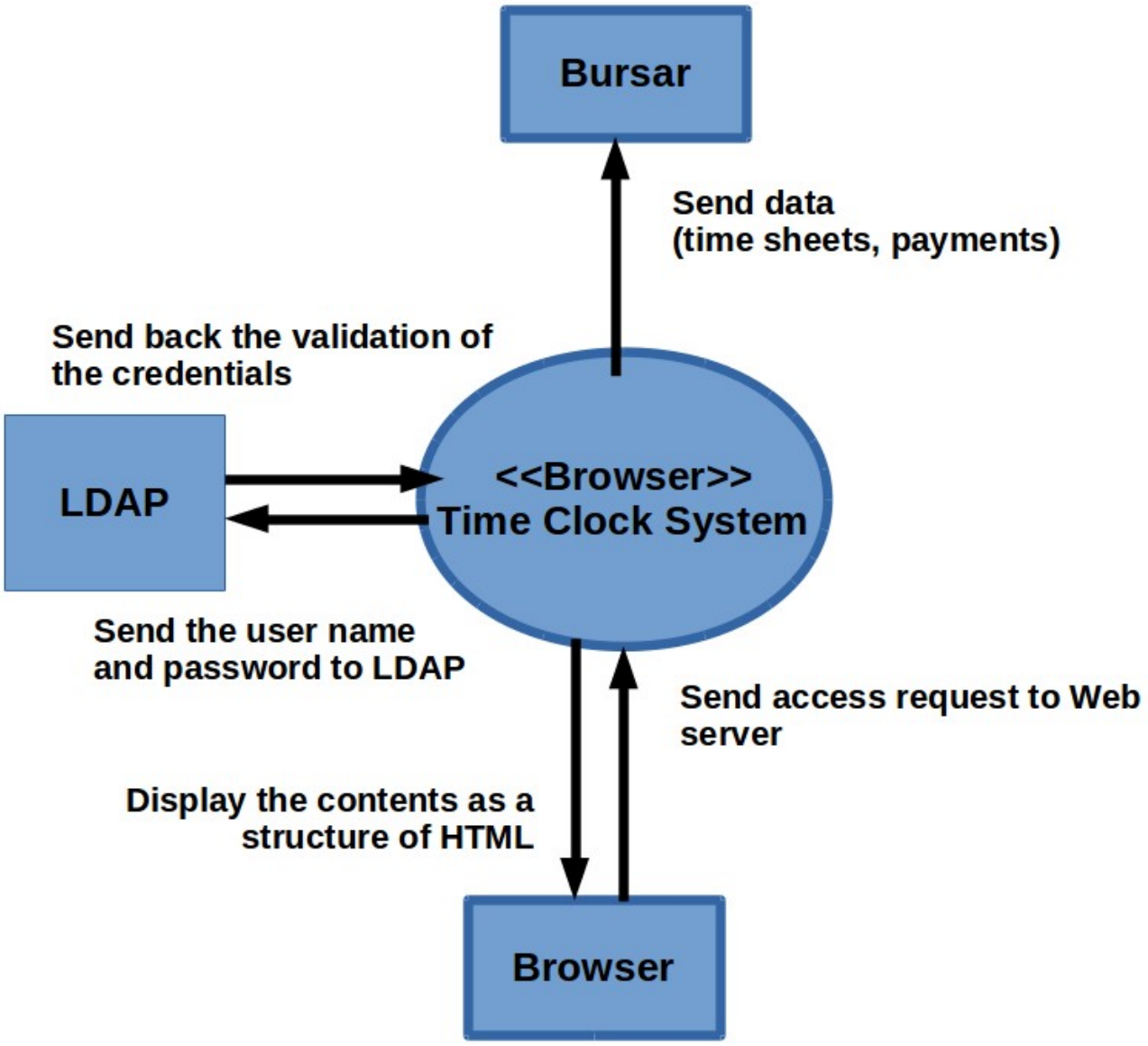
The Development View

This process view can help us manage our software in a software engineer's perspective. We can decompose the system to fit the environment. For example if this system is going to move to another university we would decompose it and find the components needed to be edited to fit that university's standards and technologies. Using this process view, we can make it easier for other software developers that will work on the system. If done correctly, every developer working on the system can better understand the dependencies and relationships between the system modules. If we did move this system but we ourselves did not become part of the new team, a development view will help the new developers.

The Physical View

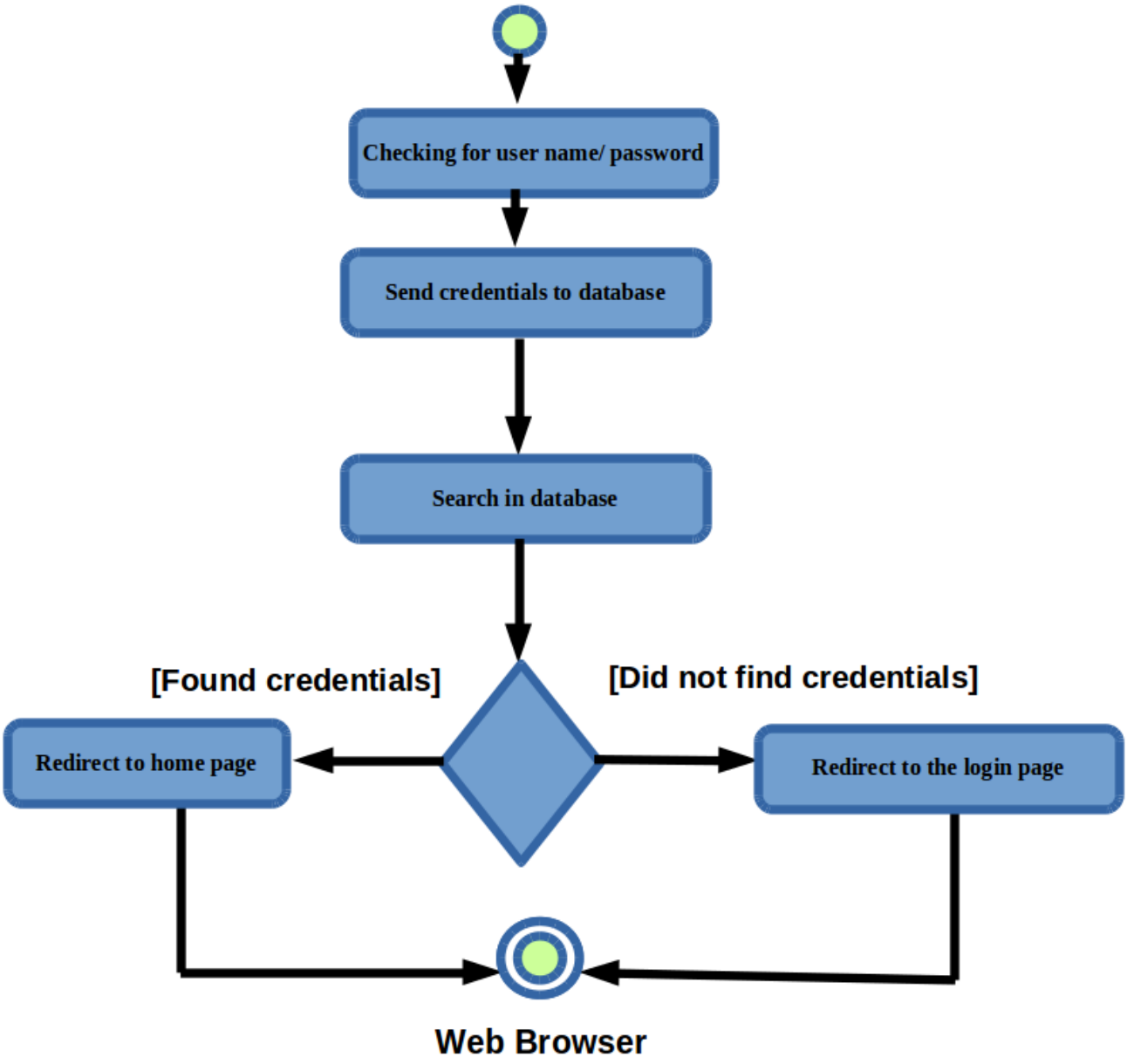
This process view will show how our system is organized in a physical point of view. For example, the user interaction in our system involves him/her to type on the keyboard and enter their information to login. With the physical process view it will create a visualization of how information is passed from the keyboard to each of the components in the system. Administrators and system designers that take care of the system will be able to understand: the physical locations of the software, the physical connections between nodes, and how to deployment and installation the system.

Context Diagram

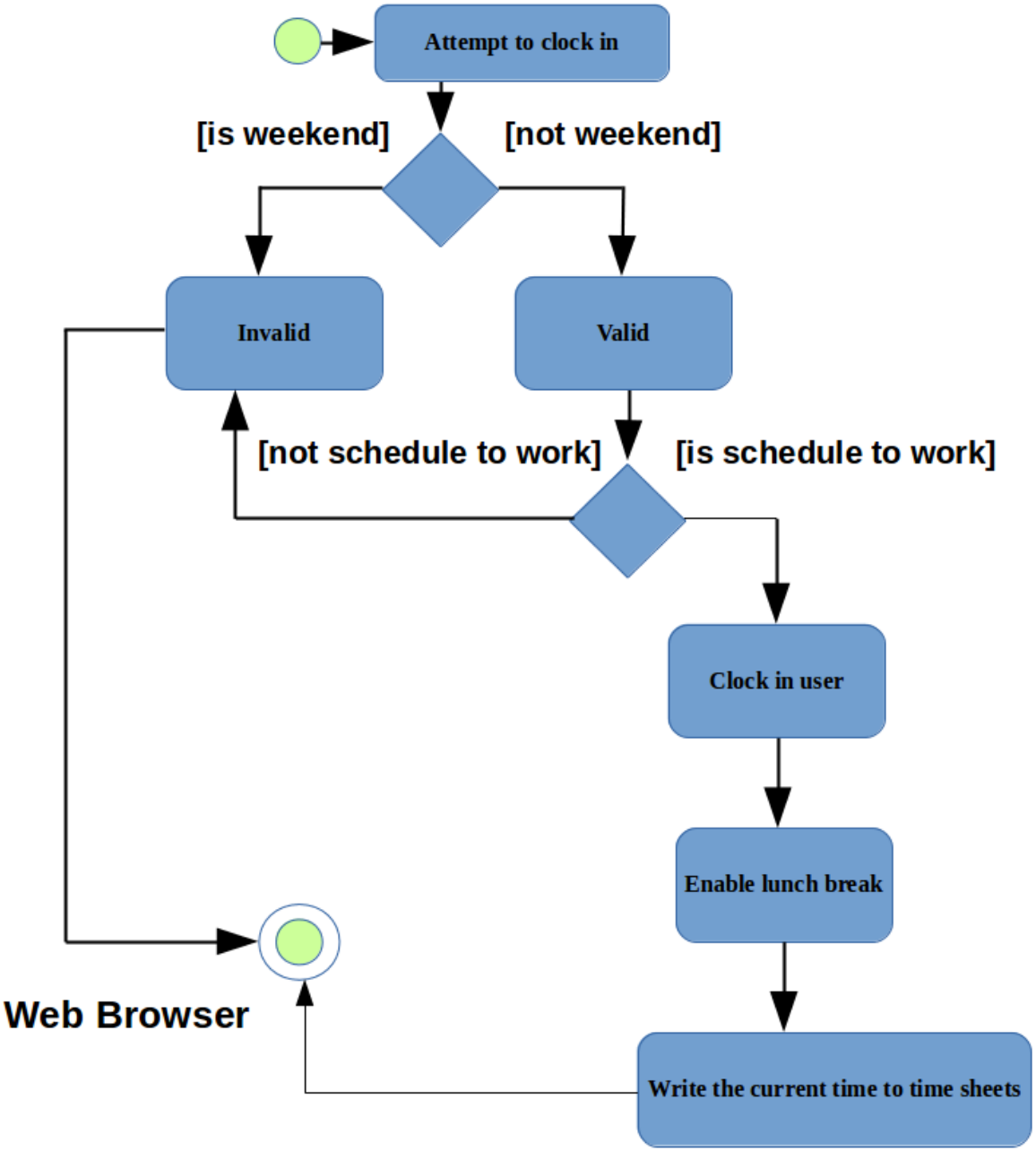


Process Models

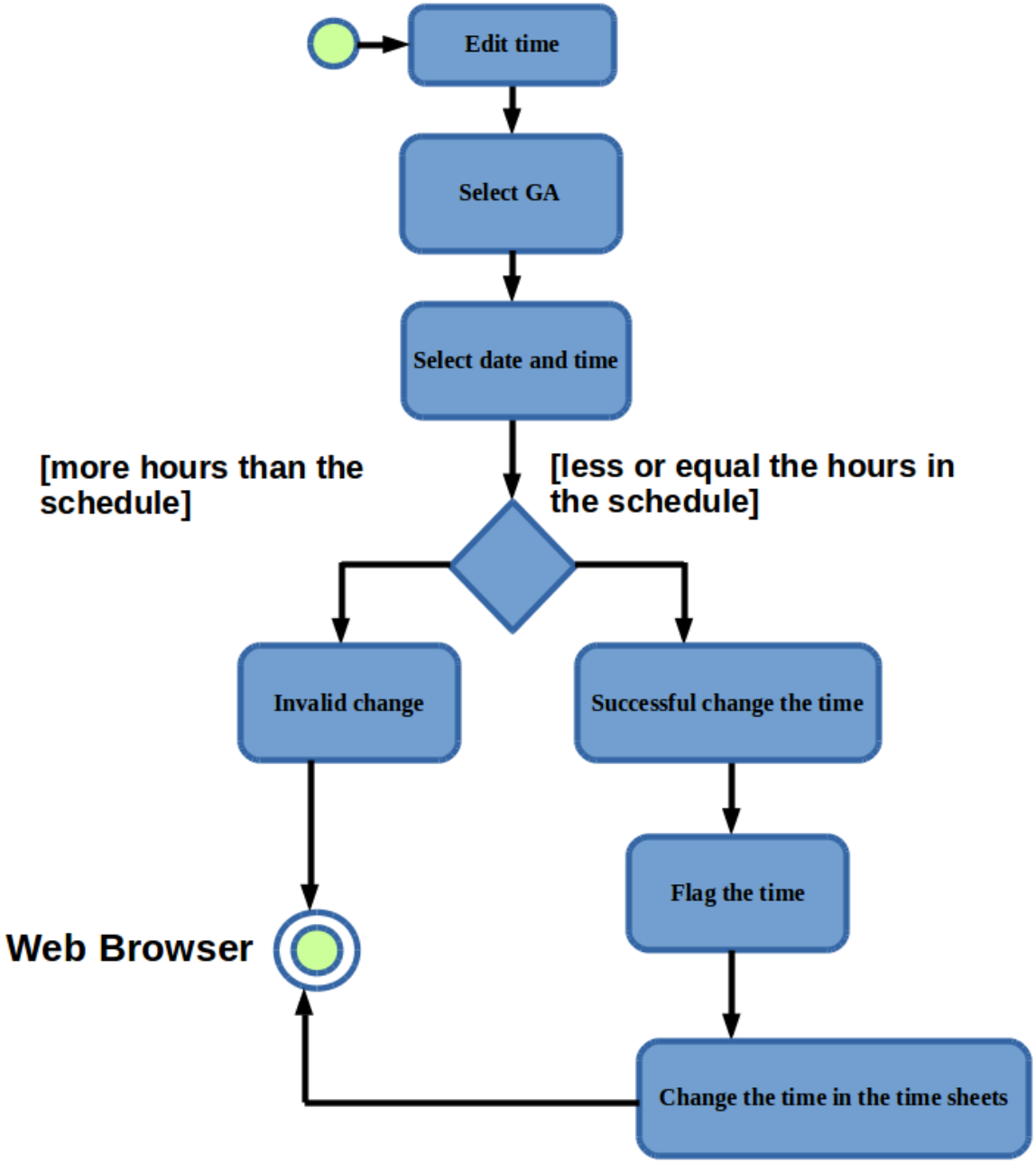
Process model for checking for user name/password



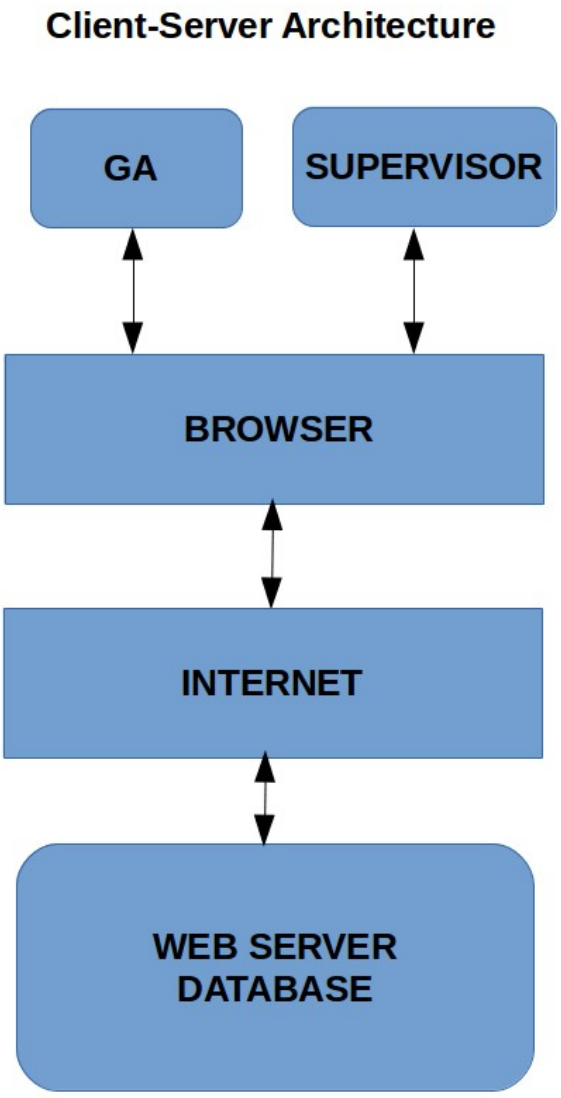
Process model of an attempt to clock in



Process model of a supervisor editing time sheets



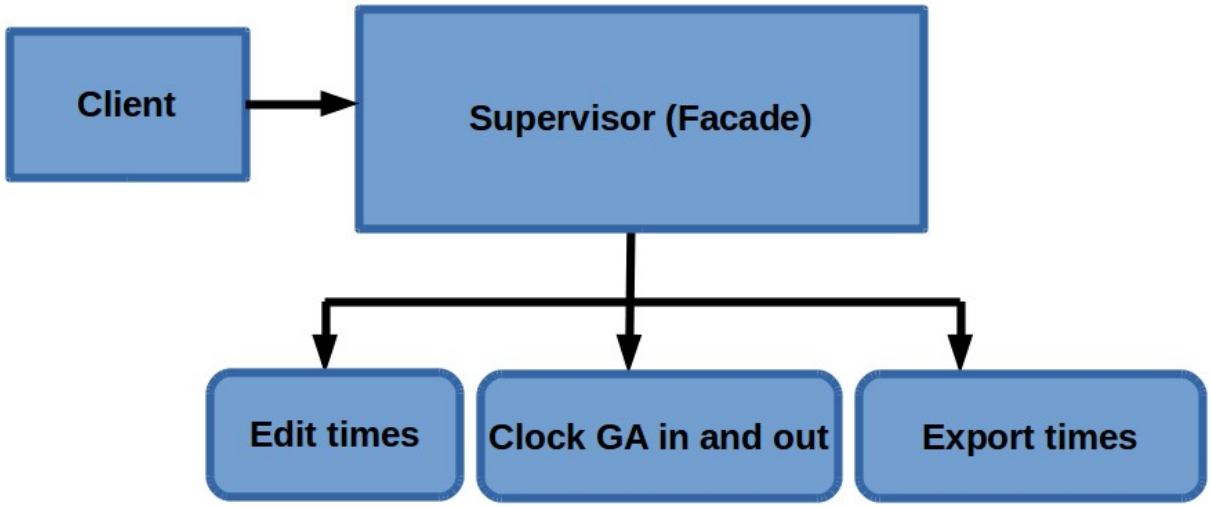
Architecture Pattern



Design Patterns

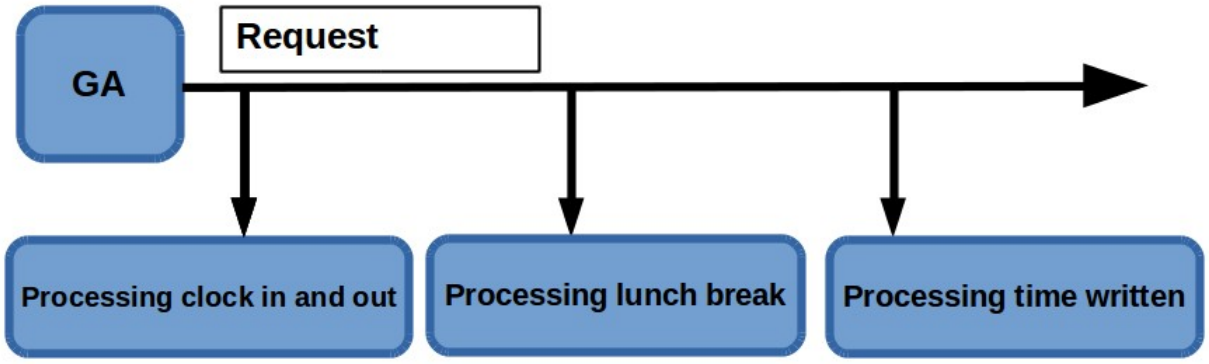
Facade

The facade design pattern provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. We choose this pattern because its good to describe how the system can give the supervisor authority over the graduate students. The subsystems that used this design pattern are the following: the system to let the supervisor edit time sheets, the clock in and out system for the graduate students, and our messaging system for the supervisor to send messages to the graduate students.

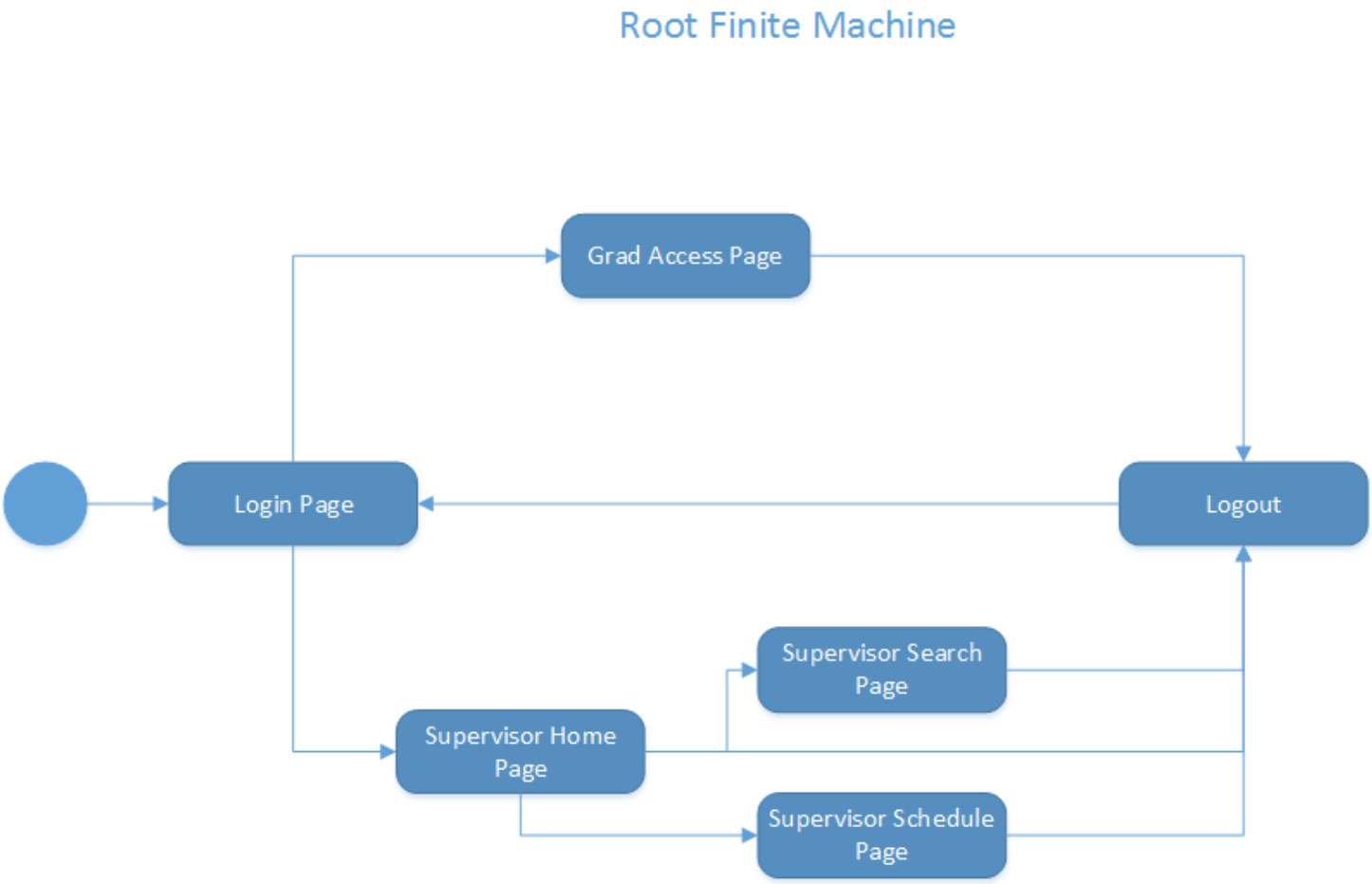


Chain of Responsibility

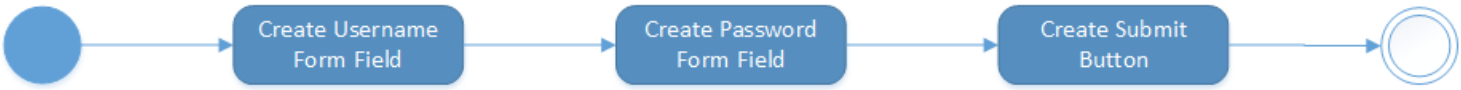
The Chain of Responsibility allows us to avoid coupling the sender a request to its receiver by giving more than one object a chance to handle the request. we choose this pattern because it describes the way how the request can be send and can be handle by the system. The subsystems that used this design pattern are the following: the clock in and out system for the graduate students, the button allowing the workers to clock in their lunch breaks, and the system that calculates the time worked considering the workers lunch breaks and other variables..



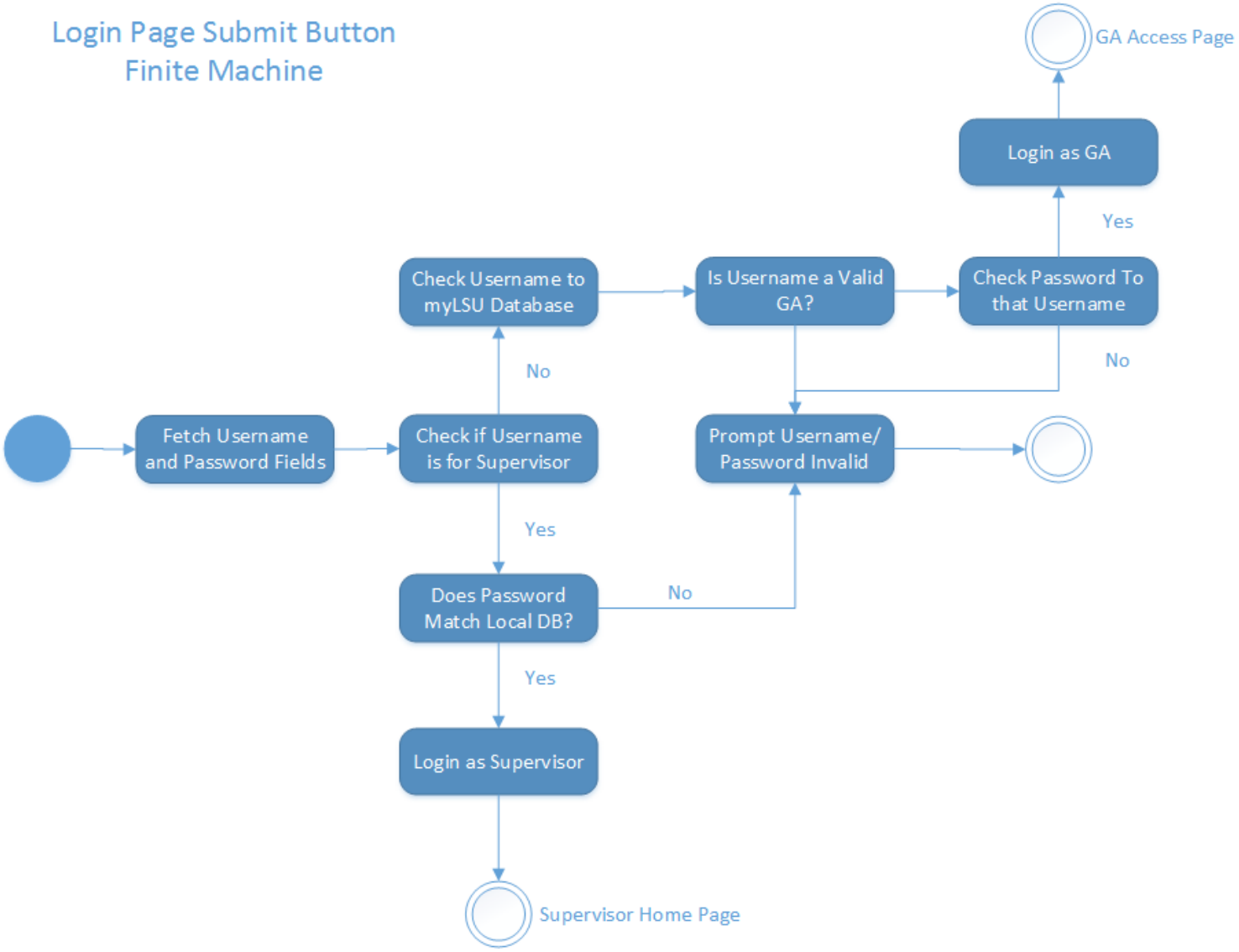
Finite State Machine



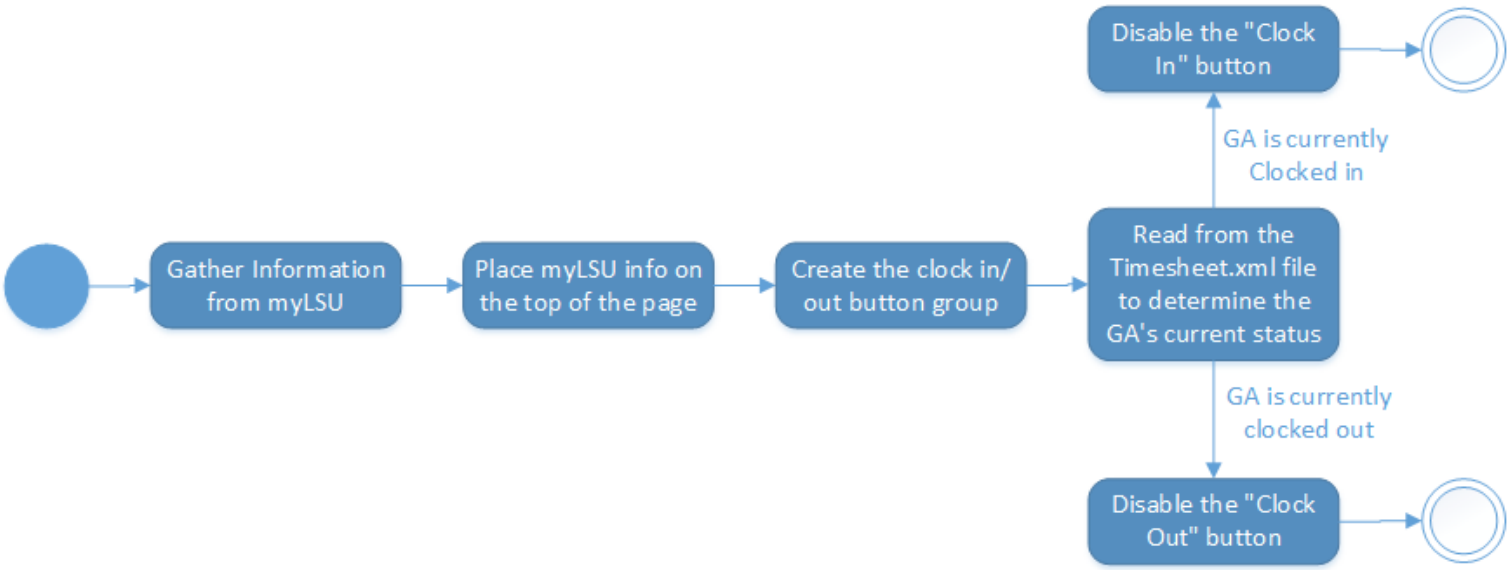
Login Page



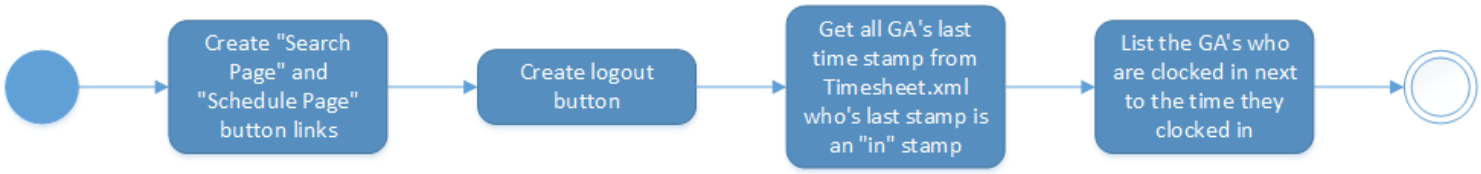
Login Page Submit Button Finite Machine



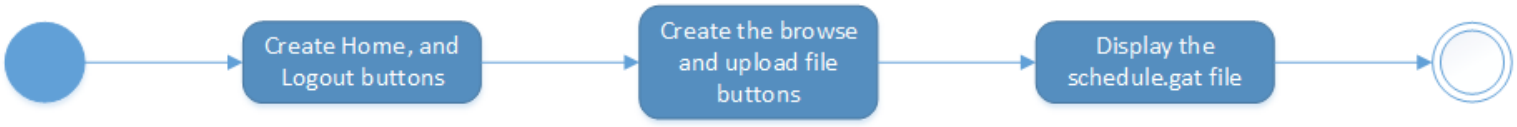
Grad Student Access Page



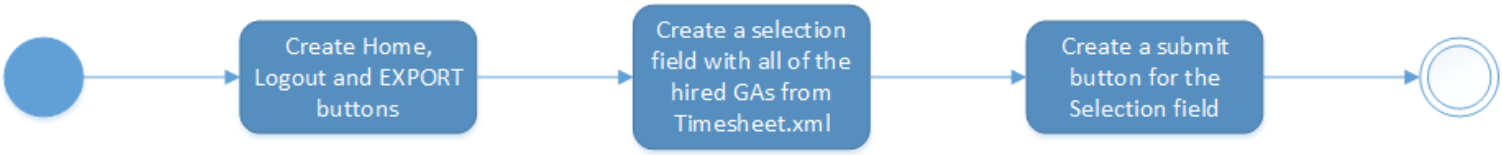
Supervisor Home Page



Supervisor Schedule Page



Supervisor Search Page



Short Answer

What you learned during this process?
We learned that online collaboration is very beneficial to software engineering. The fact that we don't have to be together in person makes things much easier considering our class schedules and responsibilities. GitHub specifically gave use a technology that is catered to software developers was much appreciated.

What software did you use to do the models in?
We used Microsoft Visio and Star UML to design the models.

Was the software difficult to learn?
The software was actually very easy to use. We just created objects gave them names and drew lines connecting to each other. The only difficult part was figuring out how how model should flow before we used the software.

How did you like the software? Was it easy to use, helpful, stable, etc.?
Like stated above, it was pretty easy to use. Specifically the feature to export our diagram into a .PNG file made it easy to put the diagrams into this document.

Which models were the most difficult to create for this project?
The process model of an attempt to clock in was the only difficult model to create since it involves the most steps to design it.

Which models were the easiest?
Our process model for checking user name and password was the easiest. All that it involved was one instance of checking the input of the client and the rest was passing that data to the other components.

Do you feel these models would help you actually implement a system like this?
They're somewhat helpful for our system but like most amateur developers, we would design the system the way that we are used to. Which is to just dive in head first and code the entire system.

What models do you feel would be the most helpful during implementation?
We feel that a sequence model would be the most beneficial to our system. Since we decided on client to server architecture, a sequence model can easily show us how our objects will interact with each other. Also the fact that our system is very linear, a sequential model would be the most appropriate of the design models that we learned.

What models would be the least useful during implementation?
State models would be one of the least useful models to use for our system. The state model would over complicate our system since all of our objects are simple. The extra detail in state models would be unnecessary. Another design model that would not be beneficial to our system would be the aggregation design model. They reason why it wouldn't be useful is that our system is too small to utilize aggregation. Aggregation would be useless for a system that is so small. Merging our components would be nearly impossible.

How hard was it to use Git?
It was actually pretty difficult using Git. Some of us didn't really see the real benefit using Git when all what we had to do was upload our work by dragging and dropping it into the GitHub repository on our web browsers. Some of us had a better experience because they are used to working in a Linux environment were they use terminal quite often.

Would you use Git again? Would you use some other form of version control?
We wouldn't really need to use Git again because dragging and dropping our files and manually editing the files inside the repository in browser is less strenuous.

Did your team see the benefit in using version control or did it just seem to get in the way?
We did find it beneficial to use version control. Some of us added what version of a file we uploaded by adding a “v” and a number representing what version it was to the file name before uploading. We have to know who did what and what version is the most recent. Without version control we would be all over the place and out of order.