**Software Design Specification**

**1. Introduction**

It is a document which will provide an overview of the software designs being implemented in the project including use case models, sequence diagrams, and other supporting requirement information.

**1.1 Purpose of this document**

This document will provide a detailed description of various UML design components like use-case diagram, state diagram and sequence diagram instilled in the project. The various interactions between the components are outlined at the end of this document.

**1.2 Scope of the development project**

Project aims to provide web application to manage the safe and secure wireless connection provided by the organization.

BENEFITS:

* Secure login.
* Easy management of devices accessing the wireless connection.

**1.3 Definitions, acronyms, and abbreviations**

* IEEE: Institute of Electrical and Electronics Engineers
* SDS: Software Design Specification
* UML: Unified Modelling Language

**1.4 References**

* R. S. Pressman, Software Engineering: A Practioner’s Approach, 5th Ed, McGraw-Hill, 2001
* Software Engineering: Ian Sommerville, 9th Ed
* IEEE SDS template

**1.5 Overview of document**

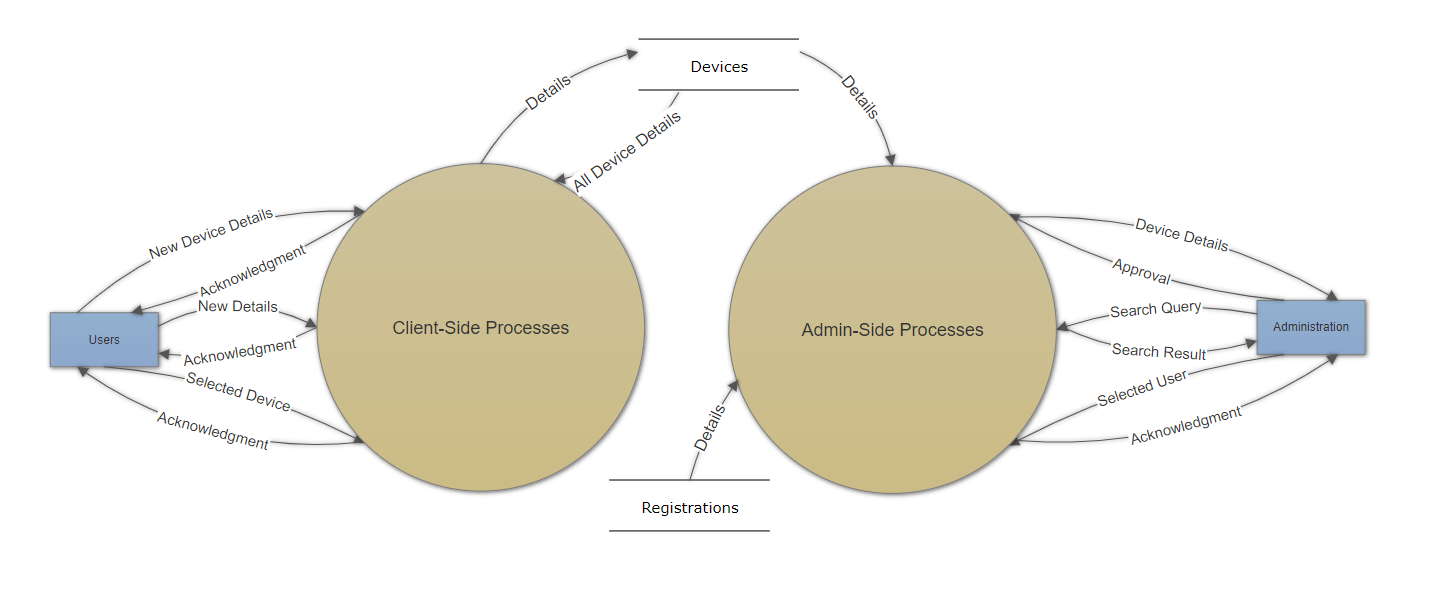
This SDS document is consist of seven section with various sub-sections. The sections of Software Design Document are:

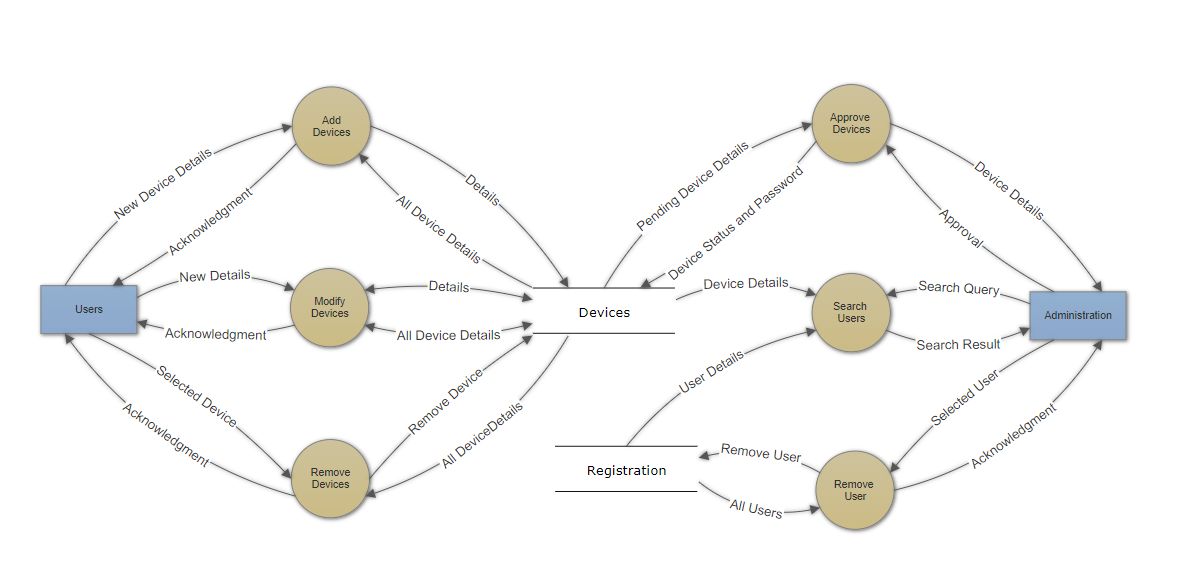
* **Introduction**- This section recount about the document, purpose of this document, scope of the development project, definitions, acronyms and abbreviations, references related to design issues are used in the document.
* **Conceptual Architecture/ Architecture Diagram**- Recounts about the context for system’s use, gives overview of modules/components, structure and relationships and user interface issues.
* **Logical Architecture**- Depicts about logical architecture(class diagram sequence diagram, etc) and it’s components.
* **Execution Architecture**- Define the runtime environment, processes, deployment view also tells about the reusability and relationships to other products.
* **Design Decisions and Trade-offs**- This section will help the reader understand the design that we are using. Also the reasons why few decisions were made over other alternatives.
* **Pseudocode for components**- As the name indicates, describes pseudocode.
* **Appendices**- Describes auxiliary matter.

**2. Logical Architecture (Data Flow Diagram, Sequence Diagram, State Diagram)**

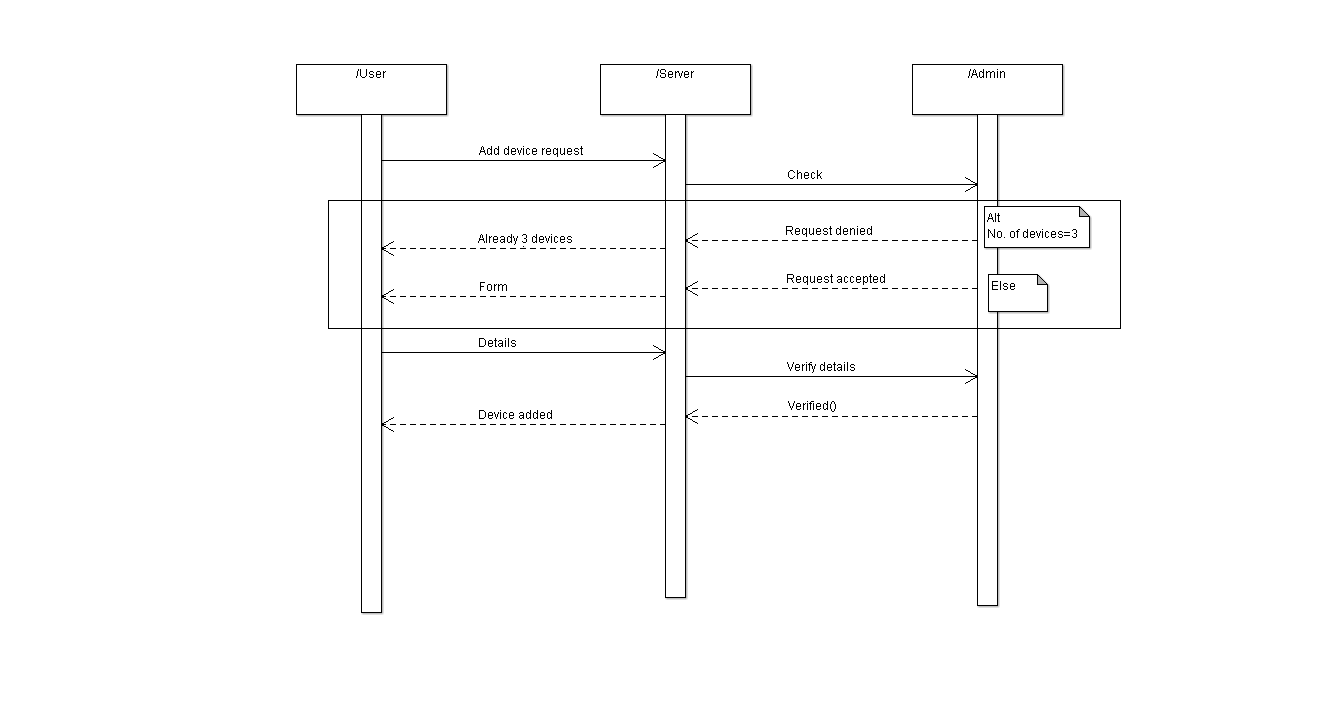
**Data Flow Diagrams**

Level – 0 Diagram

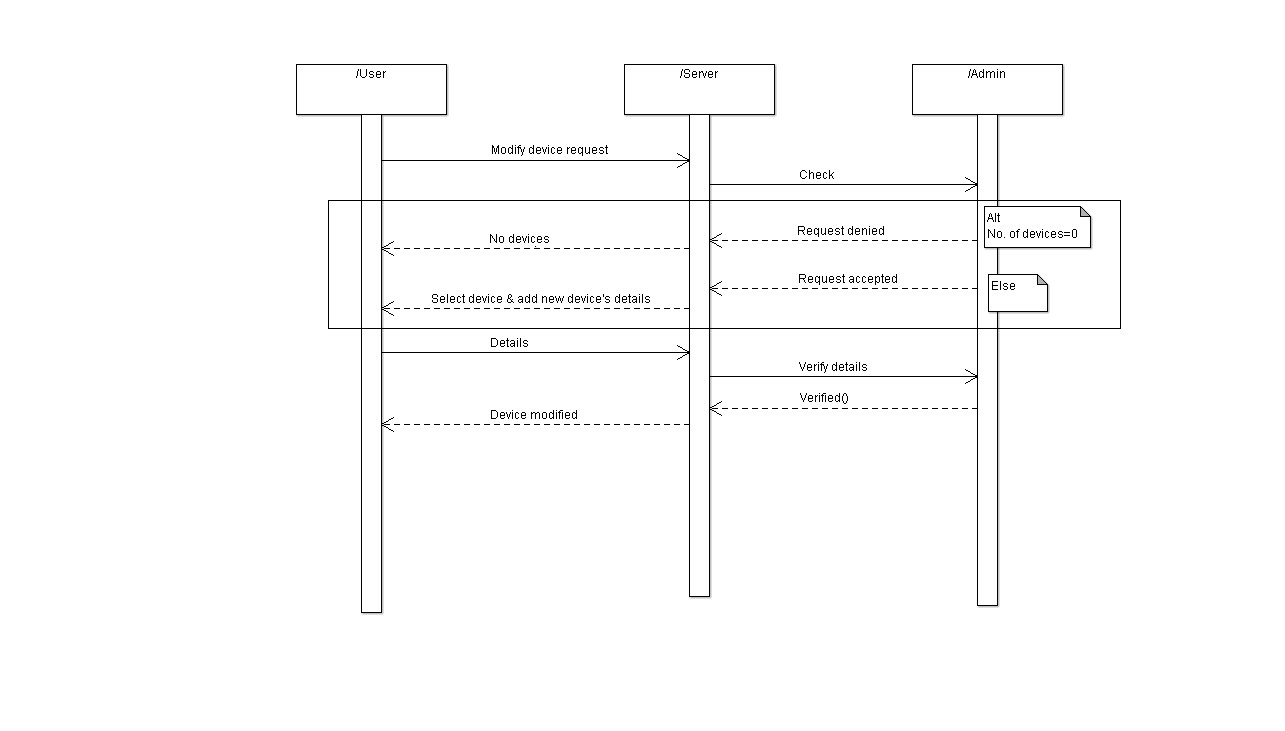
****

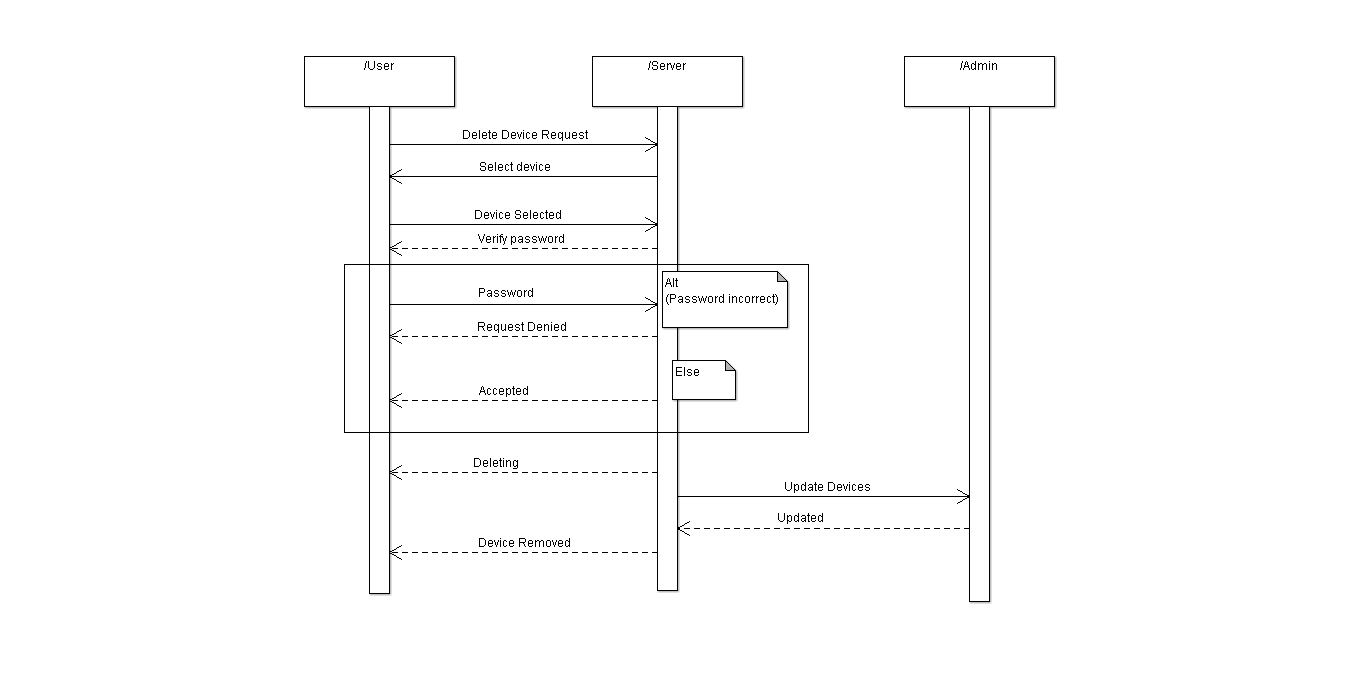
Level – 1 Diagram

**Sequence Diagrams**

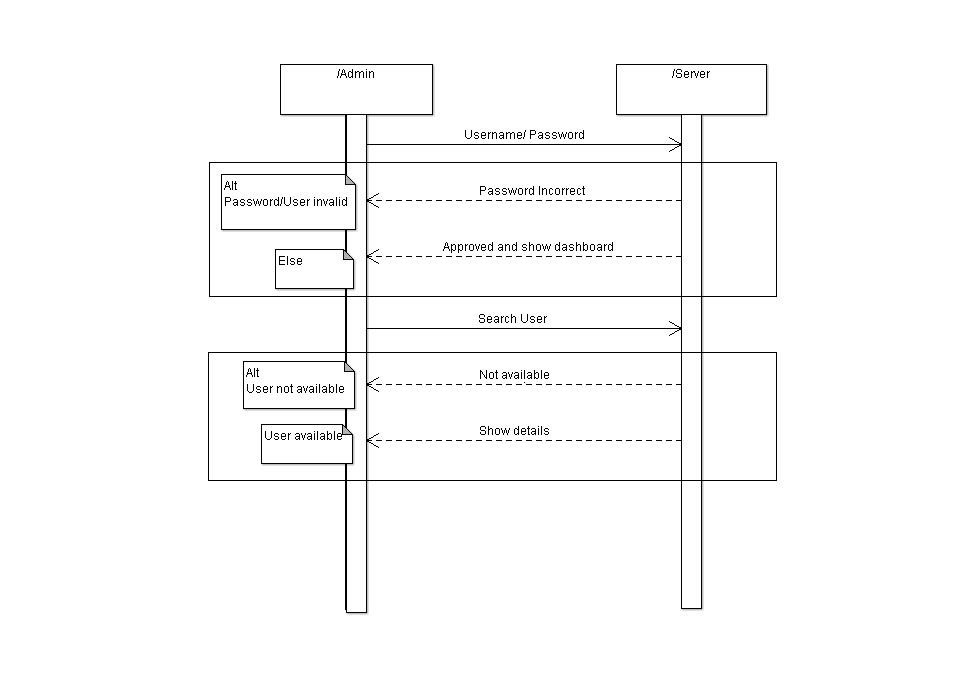
Sequence Diagram: Client Side- Add Device

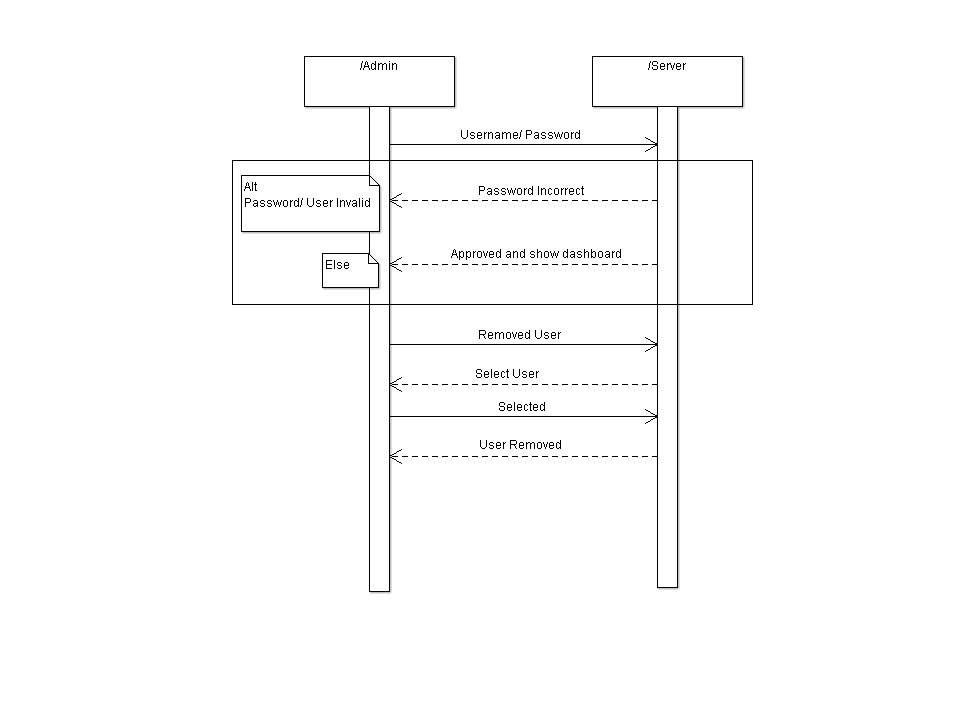
Sequence Diagram: Client Side- Modify Device

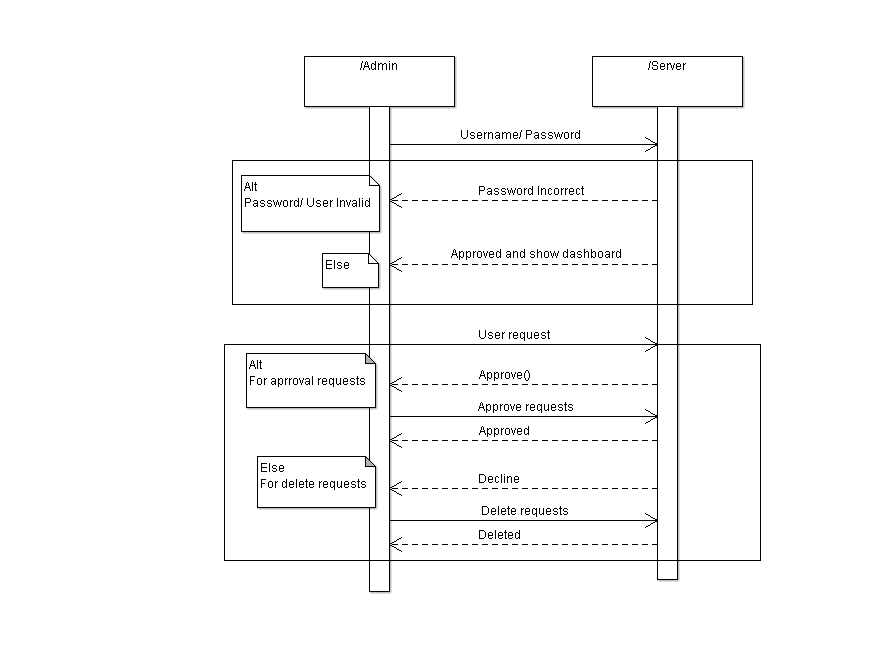


Sequence Diagram: Client Side- Remove Device

Sequence Diagram: Admin Side- Search User

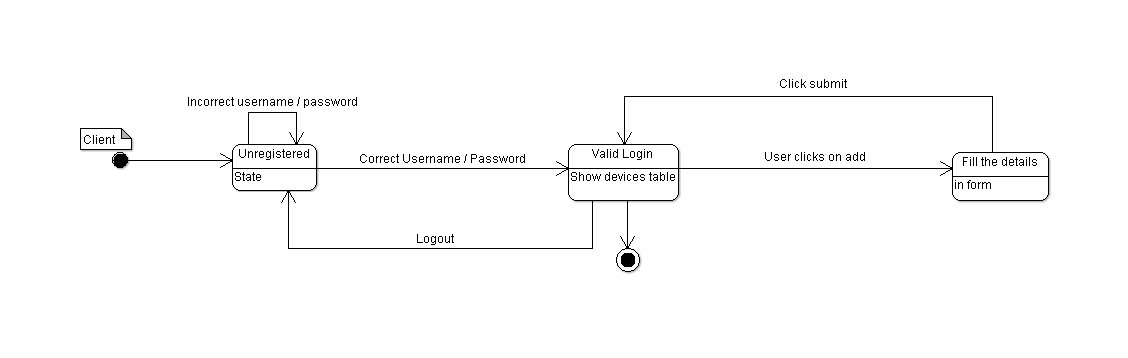


Sequence Diagram: Admin Side- Remove User

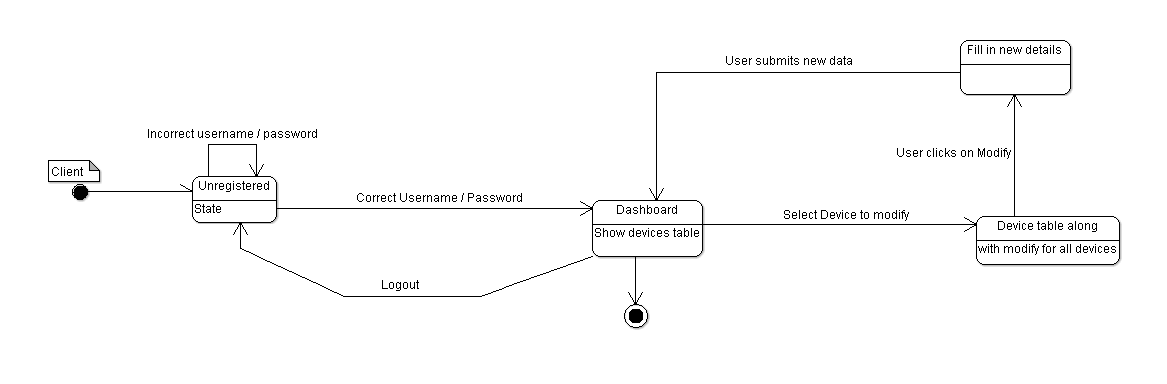
Sequence Diagram: Admin Side- Approve Requests

**State Diagram**

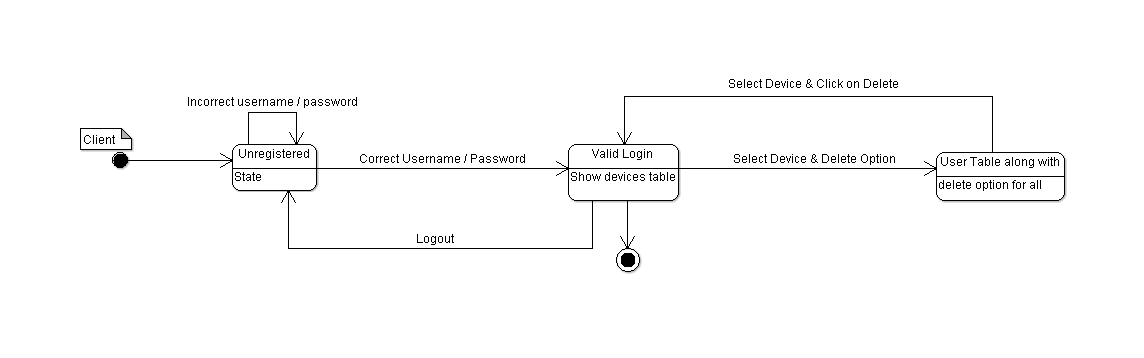
State Diagram: Client Side- Add Device



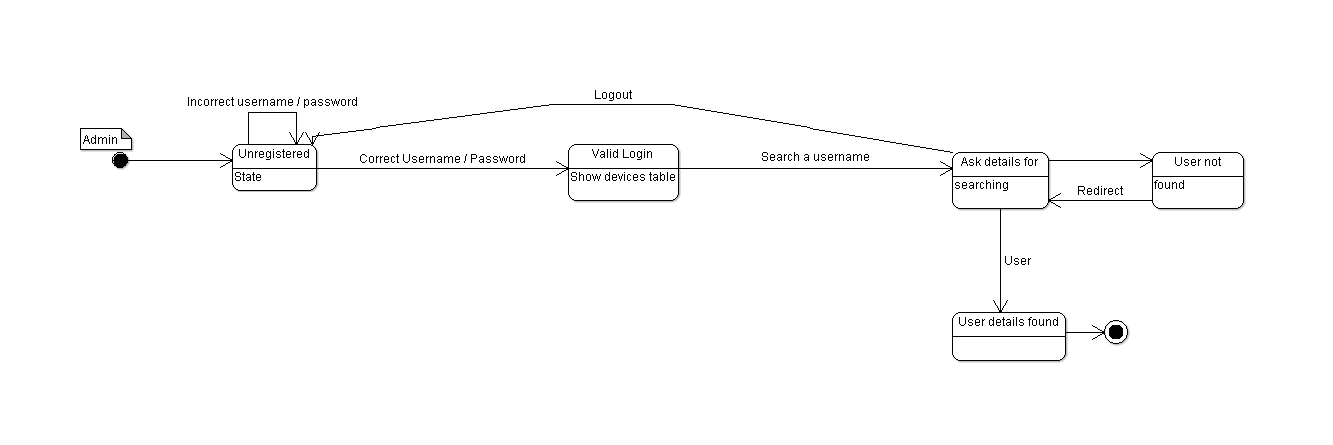
State Diagram: Client Side- Modify Device



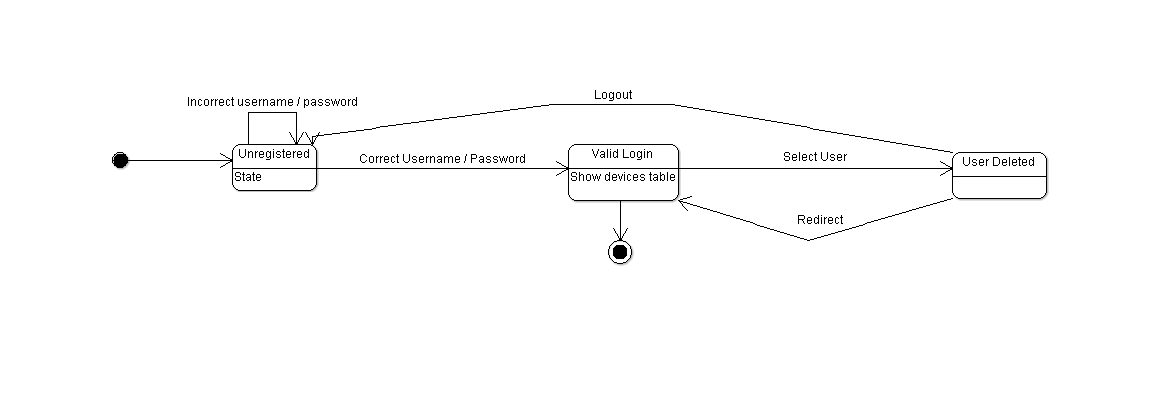
State Diagram: Client Side- Delete Device



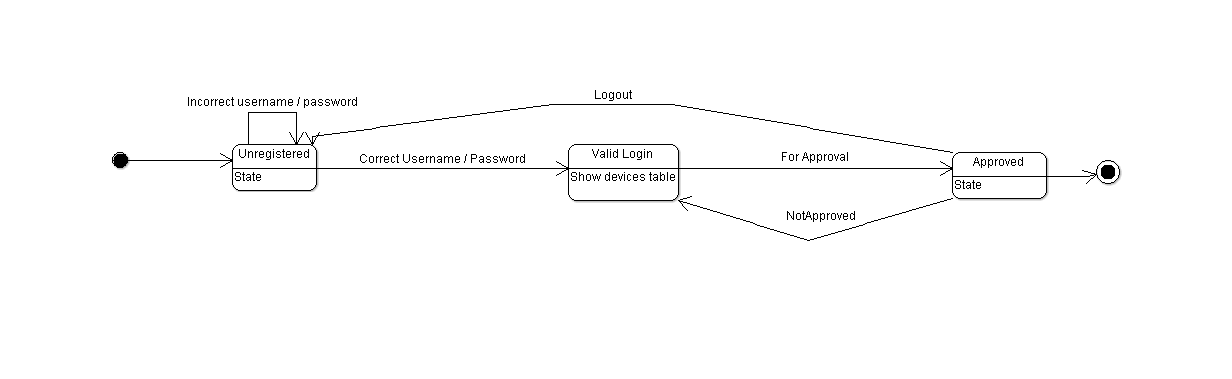
State Diagram: Admin Side- Search User



State Diagram: Admin Side- Remove User



State Diagram: Admin Side- Approve Requests



Lines and boxes are a software architect’s best friend. UML can help with that, if used appropriately. LA includes “high-level design”, “detailed design” and for some components, may extend even to the code.

**2.1 Logical Architecture Description**

Discuss some details(generic) of Logical Architecture

**2.2 X Component (or Class or Function ...)**

Use exactly the template you define in 3.2. If a part of the template is not applicable, then mark it N/A rather than omitting it.

**2.3 Y Component (or Class or Function ...)**

...

**2.n Z Component (or Class or Function ...)**

**3.0 Execution Architecture**

Define the runtime environment, processes, deployment view.

**5.0 Reuse and relationships to other products**

For teams doing enhancement work, reuse is an important issue. Most enhancement work should focus on extending, rather than replacing, the design and product development from earlier semesters. For teams doing new development, reuse can also be an important strategy. In some cases, there is freeware that could be incorporated. In other cases, there are existing modules or classes that could be adapted. Another possibility is the use of special tools that produce open source results and thus permissible under the terms of this course.

This section should include the following subsections as appropriate:

* how reuse is playing a role in your product design
* how reuse is playing a role in your product implementation (and the motivation for changes)
* if you are not reusing material that is available, then give motivation for why it is being thrown out.

**6.0 Design decisions and tradeoffs**

Use this section to motivate any decisions that will help the reader understand

the design that your team is using. This section can also capture good ideas

that were abandoned and the reasons for leaving them out of the design.

**7.0 Appendices (if any)**

**SDS component template**

The template given below suggests a reasonable structure for giving a thorough

description of each component described in Part 3 of the SDS. The specific

information depends in part on the design approach. Your team must adapt this

template to your needs and describe it in section 3.1 of your SDS.

|  |  |
| --- | --- |
| Identification | The unique name for the component and the location of the  component in the system. |
| Type | A module, a subprogram, a data file, a control procedure, a class, etc |
| Purpose | Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS, but are implied or adjunct to formally stated SDS requirements. |
| Function | What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified. |
| Subordinates | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| Dependencies | How the component's function and performance relate to other  components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components. |
| Interfaces | Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here. |
| Resources | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services. |
| Processing | The full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| Data | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary. |