Table of Contents

• 1. Introduction
  • 1.1 The Purpose of This Document
  • 1.2 Document Features
• 2. System Overview
• 3. Database Design
  • 3.1 Entity Tables
    • 3.1.1 User
    • 3.1.2 Group
    • 3.1.3 Category
    • 3.1.4 Skill
    • 3.1.5 Query
  • 3.2 Reference Tables
    • 3.2.1 Ref_Class_Level
    • 3.2.2 Ref_User_Type
    • 3.2.3 Ref_Status
  • 3.3 Relation Tables
    • 3.3.1 user_belongs_to_group
    • 3.3.2 user_skill
  • 3.4 UML Model
• 4. Web Based Interface
  • 4.1 CSS Functionality
    • 4.1.1 Screen Layout
    • 4.1.2 Color Scheme
    • 4.1.3 Page Display
  • 4.2 PHP/JavaScript Functionality
    • 4.2.1 Login
    • 4.2.2 Security
    • 4.2.3 Menu Options
    • 4.2.4 Messages
    • 4.2.5 Confirmation
    • 4.2.6 Inbox
    • 4.2.7 Skills
    • 4.2.8 Groups
• 4.2.9 Queries
• 4.2.10 User Management
• 4.2.11 User Information
• 4.2.12 Archiving
• 4.2.13 Syncing
• 4.3 Diagrams
  • 4.3.1 Group Management Flow
  • 4.3.2 User Management Flow
  • 4.3.3 Skill Management Flow
  • 4.3.4 Query Building
• 5. PDA Application
  • 5.1 PDA Application Installation
  • 5.2 PDA System Design
    • 5.2.1 Main Program
    • 5.2.2 UserData Class
    • 5.2.3 Category Class
    • 5.2.4 Skill Class
    • 5.2.5 LoginScreen Class
    • 5.2.6 CategoryScreen Class
    • 5.2.7 SkillScreen Class
    • 5.2.8 SubmitScreen Class
    • 5.2.9 SyncScreen Class
    • 5.2.10 CreateUserScreen Class
    • 5.2.11 Request Class
    • 5.2.12 RequestList Class
  • 5.3 PDA Diagrams
    • 5.3.1 UML Class Diagram
    • 5.3.2 PDA Story Board Diagram
• 6. Communication Protocol
  • 6.1 Messages
    • 6.1.1 Message Format
    • 6.1.2 SkillRequest
    • 6.1.3 SkillSheet
  • 6.2 Version
  • 6.3 Swimlane Diagram
• 7. Encryption
  • 7.1 3DES String Encryption
  • 7.2 Encryption on the PDA
    • 7.2.1 3DES.NET
    • 7.2.2 Encryption Module
  • 7.3 Encryption in PHP
    • 7.3.1 The mcrypt Encryption Functions

1. Introduction

1.1 The Purpose of This Document
This document defines the architecture for SITS, describes the overall design of the system, and specifies component modules in terms of classes, attributes, methods, input/output specifications, and interaction with other modules. The details of this document will be used by the development team to implement all features described in the Requirements Document. This document also discusses the rationale behind many design
decisions. Any member of the development team should be able to easily reference this document in order to implement some part of the system and any outsider should be able to use this document to understand the design which went into the final product for system maintenance or modification purposes.

Any changes to the design of the system which are made during implementation need to be reflected by making changes to this document.

1.2 Document Features
This document is organized according to modules which satisfy both functional and non-functional requirements for the system. The document begins with an overview of the entire system, including a system architecture diagram. The design for each module of the system is then listed and grouped according to modules which closely interact with each other. Each design section includes diagrams and tables describing how that module works, data flow within that module or between that module and others it interacts with, and/or the architectural structure of that module in the form of class diagrams. Sections of code are included in some design sections to clarify descriptions and help define implementation concepts more clearly; these code sections should be treated as guidelines and might not be used verbatim in implementation.

2. System Overview

The SITS system consists of four top-level modules: a web server, a database, a PDA application, and a synchronization application. The web server and database both reside on a CNHP server, and the web server is the only module that directly accesses the database. End users communicate with the web server via a personal computer to log into the SITS website or a PDA to sign off on skills. The website is accessed through the user's web browser. The PDA communicates with the web server using an intermediary layer: the synchronization application. This application uses HTTP POST messages to send skill requests to and retrieve skill sheets from the server. The following diagram shows the relationships between each module:
3. Database Design

In order to track the information necessary for SITS to operate correctly, a database will be set up and maintained. The database holds data on all users of the system, groups created by system users, skill statuses for users, and the skills and categories which are listed on the skill sheet. The database also defines how the different data is related. The database consists of three types of tables:

- **Entity Tables** contain all attributes of given entities represented in the database
- **Reference Tables** match ID numbers for certain attributes to the full string values of that data
- **Relation Tables** match ID attributes for different objects stored as entities to each other

For all tables, primary keys are indicated by the text (PK) next to the appropriate attribute and foreign keys are indicated by (FK).

3.1 Entity Tables

These tables hold data for given entities. Each entity table contains a list of attributes for that entity. Some attributes within entity tables are listed by a unique ID number; the associated names referenced by these IDs can be looked up in reference tables. The relationships between objects in different entity tables can be found through looking up the IDs of their key attributes in the appropriate relation tables.
### 3.1.1 User

The User table contains data for users of the system. A User is any CNHP student, clinical instructor, or SITS administrator. A row in this table consists of the attributes `user_id`, `fname`, `lname`, `username`, `password`, `photo`, `level_id`, `type_id`, `archived`, and `active`:

<table>
<thead>
<tr>
<th>user_id</th>
<th>fname</th>
<th>lname</th>
<th>username</th>
<th>password</th>
<th>email</th>
<th>level_id</th>
<th>type_id</th>
<th>archived</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Fran</td>
<td>Cornelius</td>
<td>fc27</td>
<td>1c240f795219b9abd8c56b59b</td>
<td><a href="mailto:fc27@drexel.edu">fc27@drexel.edu</a></td>
<td>NULL</td>
<td>003</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>002</td>
<td>Jeremy</td>
<td>Johnson</td>
<td>jjohnson</td>
<td>deced981d11f99c231a3e6cf8</td>
<td><a href="mailto:jjohnson@drexel.edu">jjohnson@drexel.edu</a></td>
<td>NULL</td>
<td>002</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>003</td>
<td>Jeff</td>
<td>Popyack</td>
<td>jpopack</td>
<td>d6e1820332ab50d2821695b7d</td>
<td><a href="mailto:jpop@drexel.edu">jpop@drexel.edu</a></td>
<td>NULL</td>
<td>002</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>004</td>
<td>David</td>
<td>Oakes</td>
<td>dao23</td>
<td>c775f39b52546558b72e3139</td>
<td><a href="mailto:dao23@drexel.edu">dao23@drexel.edu</a></td>
<td>005</td>
<td>001</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>005</td>
<td>Joseph</td>
<td>Pace</td>
<td>jp322</td>
<td>aef05cd30f902524410cde67a0</td>
<td><a href="mailto:jp322@drexel.edu">jp322@drexel.edu</a></td>
<td>002</td>
<td>001</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>006</td>
<td>Jeffrey</td>
<td>Hamel</td>
<td>jrh48</td>
<td>69ae7ff98495b3f815fd4ef0f0</td>
<td><a href="mailto:jrh48@gmail.com">jrh48@gmail.com</a></td>
<td>004</td>
<td>001</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 3.1.1 User Table*

#### 3.1.1.1 `user_id`(PK)

A unique number which is automatically given to each User when they are added to the database. When a new User is added, the database's "auto increment" feature is used to assign them a user_id as the next number in the sequence of all created user_ids by default.

#### 3.1.1.2 `fname`

The first name of the User. The value of this attribute is taken from the file entered into the system when an administrator adds the user to the system.

#### 3.1.1.3 `lname`

The last name of the User. The value of this attribute is taken from the file entered into the system when an administrator adds the user to the system.

#### 3.1.1.4 `username`

The Drexel username of the User. The value of this attribute is taken from the file entered into the system when an administrator adds the user to the system.

#### 3.1.1.5 `password`

The MD5 hash of the the SITS password assigned to the User. When a new User is added, a password is randomly generated and then hashed - this hash is set to the password value by default. A user can set their password at any time using the User Information page.

#### 3.1.1.6 `email`

The email address of the User. The value of this attribute is taken from the file entered into the system when an administrator adds the user to the system. A user can set their email address at any time using the User
3.1.1.7 **level_id**(FK)
A number associated with a student's class level. If the user is a student, this ID matches a possible class level: Freshman, Sophomore, Junior, Senior, Graduated, or Dropped. Otherwise, this attribute is set to **NULL** by default. The value of this attribute is taken from the file entered into the system when an administrator adds the user to the system and remains **NULL** if this entry is left blank in the file. The reference table, *Ref_Class_Level (3.2.1)*, is used to match this number with the name of the student class level.

3.1.1.8 **type_id**(FK)
A number associated with the type of User. There are three possible type IDs, corresponding to Student, Instructor, or Administrator. The value of this attribute is set corresponding to the radio button selected when an administrator adds the user to the system. The reference table, *Ref_User_Type (3.2.2)*, is used to match this number with the name of the user type.

3.1.1.9 **archived**
A boolean variable denoting if a user has been archived or not. When an administrator adds a new user to the system using the User Management page, this attribute is set to "false" (0) by default. Students whose class level is set to "Graduated" can be archived using the Archive page; once a User is archived, this attribute is set to "true" (1) using the following SQL statement: *UPDATE User SET archived = 1 WHERE level_id = 5*. Any users who have this attribute set to "true" (1) can be safely deleted from the system.

3.1.1.10 **active**
A boolean variable denoting if a user is active or not. When an administrator adds a new user to the system using the User Management page, this attribute is set to "true" (1) by default. When an administrator sets a user as inactive, the active attribute for that user is set to "false" (0) using the following SQL statement: *UPDATE User SET active = 0 WHERE user_id = INT*. When an administrator sets a user to active, the active attribute for that user is set to "true" (1) using the following SQL statement: *UPDATE User SET active = 1 WHERE user_id = INT*.

**Adding/Removing Rows**

- When an administrator adds a new user to the system using the User Management page, a new row is added to this table using the following SQL statements, depending on User type:
  - **student**: *INSERT INTO user (fname, lname, username, password, email, level_id, type_id) VALUES (STRING, STRING, STRING, STRING, STRING, INT, INT)*
  - **instructor/administrator**: *INSERT INTO User (fname, lname, username, password, email, type_id) VALUES (STRING, STRING, STRING, STRING, INT, INT)*
- After an administrator has archived Users using the Archive page, they can then remove them from the system. When an administrator removes archived Users from the system, all rows which have "true" (1) set for the archive attribute are deleted from this table using the following SQL statement: *DELETE FROM user WHERE archived = 1*

**3.1.2 Groups**

The Groups table contains data for groups stored within the system. A Group is managed by an instructor or administrator and is populated by students. A row in this table consists of the attributes group_id, user_id, and
Figure 3.1.2 Groups Table

<table>
<thead>
<tr>
<th>group_id</th>
<th>user_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>002</td>
<td>Morning Group A</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>Morning Group B</td>
</tr>
<tr>
<td>003</td>
<td>002</td>
<td>Afternoon Group A</td>
</tr>
<tr>
<td>004</td>
<td>003</td>
<td>Morning Group A</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Afternoon Group A</td>
</tr>
</tbody>
</table>

3.1.2.1 group_id(PK)
A unique number which is automatically given to each Group when it is created. When a new Group is added, the database's "auto increment" feature is used to assign it a group_id as the next number in the sequence of all created group_ids by default.

3.1.2.2 user_id(FK)
The name of the user who created the group in the same row. The value of this attribute is supplied by checking the user_id of the instructor or administrator when they create a group using the Group Management page.

3.1.2.3 name
The name of the group. The value of this attribute is supplied by an instructor or administrator when they create a group using the Group Management page.

Adding/Removing Rows

- When an instructor or an administrator creates a group using the Group Management page, a new row is added to this table using the following SQL statement:

  ```sql
  INSERT INTO groups (user_id, name) Values (INT, STRING)
  ```

- When an instructor or administrator deletes a group using the Group Management page, the corresponding row is removed from this table using the following SQL statement:

  ```sql
  DELETE FROM groups WHERE group_id = INT
  ```

3.1.3 Category

The Category table contains data for categories listed on the skill sheet. Categories are listed on the skill sheet, which is displayed on the Student Page or the User Details page of any student. A Category may contain any number of Skills. A row in this table consists of the attributes category_id, name, and active:
### 3.1.3.1 category_id (PK)
A unique number which is automatically given to each category when it is created. When a new Category is added, the database's "auto increment" feature is used to assign it a category_id as the next number in the sequence of all created category_ids by default.

### 3.1.3.2 name
The name of the category. The value of this attribute is supplied by an administrator when they add a category to the skill sheet using the Skill Management page.

### 3.1.3.3 active
A boolean variable denoting if a category is active or not. When an administrator adds a new category to the skill sheet using the Skill Management page, this attribute is set to "true" (1) by default. When an administrator deletes a category, the active attribute for that category is set to "false" (0) using the following SQL statement: `UPDATE Category SET active = 0 WHERE category_id = INT`. All Skills which contain the same category_id as the given category also have their active attributes set to "false" (0) using the following SQL statement: `UPDATE Skill SET active = 0 WHERE category_id = INT`.

### Adding/Removing Rows

- When an administrator creates a new category using the Skill Management page, a new row is added to this table using the following SQL statement:
  ```sql
  INSERT INTO category (name) VALUES (STRING)
  ```
- Rows will not be removed from this table.

### 3.1.4 Skill

The Skill table contains data for skills listed on the skill sheet. Skills are grouped by category and listed on the skill sheet, which is displayed on the Student Page or the User Details page of any student. A Skill is contained by a Category. A row in this table consists of the attributes skill_id, name, and active:
<table>
<thead>
<tr>
<th>skill_id</th>
<th>category_id</th>
<th>name</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>001</td>
<td>Admission procedure</td>
<td>1</td>
</tr>
<tr>
<td>002</td>
<td>001</td>
<td>Discharge plan</td>
<td>1</td>
</tr>
<tr>
<td>003</td>
<td>001</td>
<td>Discharge procedure</td>
<td>1</td>
</tr>
<tr>
<td>004</td>
<td>001</td>
<td>Therapeutic communication</td>
<td>1</td>
</tr>
<tr>
<td>005</td>
<td>002</td>
<td>Handwashing</td>
<td>1</td>
</tr>
<tr>
<td>006</td>
<td>002</td>
<td>Surgical hand scrub</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 3.1.4 Skill Table*

3.1.4.1 **skill_id**(PK)
A unique number which is automatically given to each skill when it is created. When a new Skill is added, the database's "auto increment" feature is used to assign it a skill_id as the next number in the sequence of all created skill_ids by default.

3.1.4.2 **category_id**(FK)
The ID of the category which contains the skill corresponding to the skill_id in the same row. The value of this attribute is supplied by checking the category_id of the category which contains this skill when an administrator adds it to the skill sheet using the Skill Management page.

3.1.4.3 **name**
The name of the skill. The value of this attribute is supplied by an administrator when they add a skill to the skill sheet using the Skill Management page.

3.1.4.4 **active**
A boolean variable denoting if a skill is active or not. When an administrator adds a new skill to the skill sheet using the Skill Management page, this attribute is set to "true" (1) by default. When an administrator deletes a skill, the active attribute for that skill is set to "false" (0) by default. When an administrator deletes a category all Skills which contain the category_id of the deleted category also have their active attributes set to "false" (0) using the following SQL statement:

```
UPDATE Skill SET active = 0 WHERE category_id = INT
```

**Adding/Removing Rows**

- When an administrator creates a new skill using the Skill Management page, a new row is added to this table using the following SQL statement:

```sql
INSERT INTO skill (category_id, name) VALUES (INT, STRING)
```

- Rows will not be removed from this table.

3.1.5 **Query**
The Query table contains data for queries which are saved by administrators. When an administrator creates a new query using the Query page, they can choose to name that query and save it. When an administrator saves a query, data is entered into this table. A row in this table consists of the attributes query_id user_id, query, and name:

<table>
<thead>
<tr>
<th>query_id</th>
<th>user_id</th>
<th>query</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>001</td>
<td><em>SQL statement</em></td>
<td>Seniors with Fewer than 20 Skills Complete</td>
</tr>
<tr>
<td>002</td>
<td>001</td>
<td><em>SQL statement</em></td>
<td>Instructors with More than 10 Students</td>
</tr>
<tr>
<td>003</td>
<td>001</td>
<td><em>SQL statement</em></td>
<td>Students of Dr. Richard Pratt</td>
</tr>
</tbody>
</table>

*Figure 3.1.5 Query Table*

3.1.5.1 query_id(PK)
A unique number which is automatically given to each Query when it is saved. When a Query is saved, the database's "auto increment" feature is used to assign it a query_id as the next number in the sequence of all created query_ids by default.

3.1.5.2 user_id(FK)
The name of the user who saved the query in the same row. The value of this attribute is supplied by checking the user_id of the administrator when they save a query using the Query page.

3.1.5.3 query
The SQL statement for the saved query. The value of this attribute is supplied by the fields of the query builder on the Query page. This is the exact statement which will be run on the database to retrieve whatever information is requested by the query in the same row.

3.1.5.4 name
The name of the query. The value of this attribute is supplied by an administrator when they save a query using the Query page.

Adding/Removing Rows

- When an administrator saves a query using the Query page, a row will be added to this table for that query using the following SQL statement:
  ```sql
  INSERT INTO query (user_id, query, name) VALUES (INT, STRING, STRING)
  ```

- When an administrator deletes a query using the Query page, the corresponding row is removed from this table using the following SQL statement:
  ```sql
  DELETE FROM query WHERE query_id = INT
  ```

3.2 Reference Tables

These tables are used to match the values of entities’ ID numbers with the names they reference. Rows in these tables consist of the ID number and the corresponding string value for the referenced name.
3.2.1 Ref_Class_Level

This reference table is used to match level ID numbers to the names of student class levels. A row in this table consists of a level_id and a name:

<table>
<thead>
<tr>
<th>level_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Freshman</td>
</tr>
<tr>
<td>002</td>
<td>Sophomore</td>
</tr>
<tr>
<td>003</td>
<td>Junior</td>
</tr>
<tr>
<td>004</td>
<td>Senior</td>
</tr>
<tr>
<td>005</td>
<td>Graduated</td>
</tr>
<tr>
<td>006</td>
<td>Dropped</td>
</tr>
</tbody>
</table>

![Figure 3.2.1 Ref_Class_Level Table](image)

3.2.1.1 level_id(PK)
A unique number given to each class level. The given IDs are 1, 2, 3, 4, 5, and 6.

3.2.1.2 name
The name of the class level. The given names are “Freshman”, “Sophomore”, “Junior”, “Senior”, “Graduated”, and “Dropped”.

Adding/Removing Rows

- When the database is first created, rows will be added to this table for all current class levels using the following SQL statements:

  ```sql
  INSERT INTO Ref_Class_Level (level_id, name) VALUES (1, 'Freshman')
  INSERT INTO ref_class_level (level_id, name) VALUES (2, 'Sophomore')
  INSERT INTO ref_class_level (level_id, name) VALUES (3, 'Junior')
  INSERT INTO ref_class_level (level_id, name) VALUES (4, 'Senior')
  INSERT INTO ref_class_level (level_id, name) VALUES (5, 'Graduated')
  INSERT INTO ref_class_level (level_id, name) VALUES (6, 'Dropped')
  ```

- If a new class level is added to the program, the database administrator must add a new row to this table using the following SQL statement:

  ```sql
  INSERT INTO ref_class_level (level_id, name) VALUES (INT, STRING)
  ```
3.2.2 Ref_User_Type

This reference table is used to match type ID numbers to the names of user types. A row in this table consists of a type_id and a name:

<table>
<thead>
<tr>
<th>level_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Student</td>
</tr>
<tr>
<td>002</td>
<td>Instructor</td>
</tr>
<tr>
<td>003</td>
<td>Administrator</td>
</tr>
</tbody>
</table>

3.2.2.1 type_id(PK)
A unique number given to each user type. The given IDs are 1, 2, and 3.

3.2.2.2 name
The name of the user type. The given names are “Student”, “Instructor”, and “Administrator”.

Adding/Removing Rows

- When the database is first created, rows will be added to this table for the 3 user types using the following SQL statements:
  - `INSERT INTO ref_user_type (type_id, name) VALUES (1, 'Student')`
  - `INSERT INTO ref_user_type (level_id, name) VALUES (2, 'Instructor')`
  - `INSERT INTO ref_user_type (level_id, name) VALUES (3, 'Administrator')`

- Rows will not be removed from this table.

3.2.3 Ref_Status

This reference table is used to match status ID tags to the names of skill statuses. A row in this table consists of a status_id and a name:

<table>
<thead>
<tr>
<th>status_id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Awaiting Approval</td>
</tr>
<tr>
<td>002</td>
<td>Approved</td>
</tr>
</tbody>
</table>

Figure 3.2.3 Ref_Status Table
3.2.3.1 status_id(PK)
A unique number given to each status. The given IDs are 1 and 2.

3.2.3.2 name
The name of the status. The given names are “Awaiting Approval” and “Approved”.

Adding/Removing Rows

- When the database is first created, rows will be added to this table for the 4 statuses using the following SQL statements: 
  INSERT INTO ref_status (type_id, name) VALUES (1, 'Awaiting Approval')
  INSERT INTO ref_status (type_id, name) VALUES (2, 'Approved')

- Rows will not be removed from this table.

3.3 Relation Tables

These tables are used to relate objects in different entity tables to each other. Rows in these tables consist of the ID numbers for two related objects from two different entity tables.

3.3.1. user_belongs_to_group

This relation table is used to match Users with the Group(s) they are enrolled in. A row in this table consists of a group_id and a user_id:

<table>
<thead>
<tr>
<th>group_id</th>
<th>user_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>015</td>
</tr>
<tr>
<td>002</td>
<td>022</td>
</tr>
<tr>
<td>003</td>
<td>024</td>
</tr>
<tr>
<td>008</td>
<td>015</td>
</tr>
<tr>
<td>010</td>
<td>015</td>
</tr>
<tr>
<td>011</td>
<td>022</td>
</tr>
</tbody>
</table>

Figure 3.3.1 user_belongs_to_group Table

3.3.1.1 group_id(FK)
The ID of the Group in which the User identified by the user_id in the same row is enrolled.

3.3.1.2 user_id(FK)
The ID of the User who is enrolled in the Group identified by the group_id in the same row.

Adding/Removing Rows
• When an instructor or administrator adds a student to a group they manage using that group’s Group Page, a row is added to this table with that student’s user_id and the group_id of that group using the following SQL statement:

\[
\text{INSERT INTO user_belongs_to_group (user_id, group_id) VALUES (INT, INT)}
\]

• When an instructor or administrator removes a student from a group they manage using that group’s Group Page, the row which includes both that student’s user_id and that group’s group_id is removed from this table using the following SQL statement: DELETE FROM user_belongs_to_group WHERE user_id = INT AND group_id = INT

• When an instructor or administrator deletes a group they manage using the Group Management page, all rows including that group’s group_id are removed from this table using the following SQL statement:

\[
\text{DELETE FROM user_belongs_to_group WHERE group_id = INT}
\]

• When an administrator removes an archived student from the system using the Archive page, all rows including that user’s user_id are removed from this table using the following SQL statement:

\[
\text{DELETE FROM user_belongs_to_group WHERE user_id = INT}
\]

3.3.2 user_skill

This relation table is used to match Users with the status they currently have for each Skill on the skill sheet. A row in this table consists of a request_id, a student_id (a user_id), a skill_id, a status_id, an instructor_id (a user_id), a time_stamp, and a signature (a filename):

<table>
<thead>
<tr>
<th>request_id</th>
<th>student_id</th>
<th>skill_id</th>
<th>status_id</th>
<th>instructor_id</th>
<th>time_stamp</th>
<th>signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>015</td>
<td>002</td>
<td>002</td>
<td>002</td>
<td>2007-02-15 10:35:23</td>
<td><em>file location</em></td>
</tr>
<tr>
<td>002</td>
<td>015</td>
<td>002</td>
<td>001</td>
<td>002</td>
<td>2007-02-22 10:42:37</td>
<td><em>file location</em></td>
</tr>
<tr>
<td>003</td>
<td>015</td>
<td>012</td>
<td>001</td>
<td>003</td>
<td>2007-02-22 01:21:13</td>
<td><em>file location</em></td>
</tr>
<tr>
<td>004</td>
<td>022</td>
<td>002</td>
<td>002</td>
<td>002</td>
<td>2007-02-15 09:25:53</td>
<td><em>file location</em></td>
</tr>
<tr>
<td>005</td>
<td>022</td>
<td>012</td>
<td>001</td>
<td>003</td>
<td>2007-02-19 03:22:48</td>
<td><em>file location</em></td>
</tr>
</tbody>
</table>

Figure 3.3.2 user_skill Table

3.3.2.1 request_id(PK)
A unique number which is automatically given to each skill request when it is added to the database. When a new entry to this table is added, the database’s "auto increment" feature is used to assign it a request_id as the next number in the sequence of all created request_ids by default.

3.3.2.2 student_id(FK)
The ID of the User who has the status corresponding to the status_id for the Skill identified by the skill_id in the same row.
3.3.2.3 skill_id(FK)
The ID of the Skill which has a status corresponding to the status_id on the skill sheet of the User identified by the user_id in the same row.

3.3.2.4 status_id(FK)
A number associated with a status type for a Skill. A skill can have one of two statuses: 1 (Awaiting Approval), or 2 (Approved). The reference table, Ref_Status (3.2.3), is used to match this number with the name of the status. Since a row is only added to this table when a new skill request is made, the value of this attribute is set to 1 by default.

3.3.2.5 instructor_id(FK)
The ID of the User who signed off on the Skill corresponding to the skill_id in the same row.

3.3.2.6 time_stamp
The time that the User corresponding to the instructor_id signed off on the Skill corresponding to the skill_id all in the same row.

3.3.2.7 signature
The filename of the signature, stored as an image, made by the User corresponding to the instructor_id in the same row.

Adding/Removing Rows

- When a skill request is entered into the system, a row is added to this table using the following SQL statement:
  
  INSERT INTO user_skill (user_id, skill_id, instructor_id, time_stamp, signature) VALUES (INT, INT, INT, DATETIME, STRING)

- When a skill request for a student is approved by an instructor, the status_id for the row identified by the request_id is updated using the following SQL statement:
  
  UPDATE user_skill SET status = 2 WHERE request_id = INT

- When a skill request for a student is denied by an instructor, the row identified by the request_id is removed from this table using the following SQL statement:
  
  DELETE FROM user_skill WHERE request_id = INT

3.4 UML Model
4. Web Based Interface

The web based interface is the interface that system users (students, administrators, instructors) can access using any standard and sufficiently modern web browser. Specifically, the interface is designed to work with Safari (the Mac OS X default browser), Mozilla Firefox (the most popular cross platform browser), and Microsoft Internet Explorer 6 and 7 (the default Windows browsers). The site is designed with an optimal user screen resolution of 1024x768 pixels, but will be able to gracefully scale down to 800x600 pixels. This is intended to be a minimum as very few users use anything below 800x600 pixels.

The web based interface is designed using four programming languages acting in conjunction. HTML is used to set up the static elements of the page, as well as the actual content. CSS is used to apply colors and layout to the HTML. PHP is used to generate dynamic content from the database. PHP also parses input files and generates output files for exporting of query results, archiving of data, and importing of user lists. JavaScript is used to dynamically update the screen without reloading the current page, such as adding rows to a table or verifying the non-empty state of text boxes.

4.1 CSS Functionality

4.1.1 Screen Layout
This section satisfies the requirements in section 2.3.1.2 in the Requirements Document.

The layout for the system is done using percentages. This way the interface gracefully scales across multiple screen resolutions. There is a 2% border around the main screen area, except for the top which is 10%. The clear 10% at the top is used to display a banner with the name of the system and it's logo, this is the BANNER AREA.
The area in the middle of the screen is referred to as the ACTION AREA. The ACTION AREA is where all user interaction occurs and is split into data sub-sections. The first sub-section of the ACTION AREA is the MENU AREA, this contains the menu options for a user. These options change as defined in Section 4.2.3. This section has a height of 88% and a width of 15%. This tall height with a rather skinny width accommodates a thin but possibly long list of options. To the left of the menu area, the second subsection of the ACTION AREA is the DATA AREA. The DATA AREA is broken into three sub-sections. In total, the DATA AREA has a height of 88% and a width of 81%. The DATA AREA contains the INFORMATION AREA, ALERTS AREA, and DISPLAY AREA. The INFORMATION AREA is 2% tall and 81% wide. This area contains the welcome message that includes the user's name. Below this is a 2% tall and 81% wide area called the ALERTS AREA. In this area, messages to the user about the success or failure of their actions are displayed. If more information is required than can fit in this area, it is shown in the DISPLAY AREA. The display area is 73% tall and 80% wide. This area displays all of the MAIN PAGES, such as the query building tool, user tool, skills tool, groups tools, lengthy failure notices, etc. This area is able to scroll if the information inside of it is too large to fit in the allotted space.

Below the DISPLAY AREA is the INTERACTION AREA. This area is 5% tall and 80% wide. It displays buttons and other form objects that affect the DISPLAY AREA, but is on screen even if the DISPLAY AREA is scrolled. Below is a figure depicting the various areas and their positions/sizes.

4.1.2 Color Scheme
The general color scheme for the application is shades of blue and white. Dark blue is used for the ALERTS AREA, to highlight the selected menu option in the MENU AREA, and to outline the DISPLAY AREA. Light blue is used for rollovers in the MENU AREA, as well as highlighting of selected options. A mid-range blue is used for tables to help them stand out from the display area. White is used both in the DEADSPACE, the 10% buffer zone, as well as in the background of the display area. All text is black unless it is being displayed in an area that is dark blue, in this case it is white. The following are the colors being used as defined by their RGB hexadecimal value:
4.1.3 Page Display
All of the pages described in the Requirements Document are displayed using the layout parameters described above.

4.2 PHP/JavaScript Functionality
This section contains information on how PHP and JavaScript are used to enable the various web pages to communicate with each other. Pages are broken into four sections. First, the "Details" section describes basic information such as the file name and variables that are passed to the page. Next, the "Actions" section describes any actions that the page performs when it is loaded. These actions generally involve updating the database based on the POST or GET variables sent to the page. Third, the "Display" section describes what is displayed on the screen. Finally, the "Links" section lists all the connections between pages, including hyperlinks and buttons. The links section also discusses what POST and GET variables are set when these links are followed.

4.2.1 Login
This section satisfies the requirements listed in section 2.3.1.1 in the Requirements Document.

This page displays two text fields to the user and a "Login" button. The text entered into the "Username" field is stored as the login_id variable and the text entered in the "Password" field is hashed using JavaScript MD5 hashing and then stored as the login_pass_hash. When the "Login" button is pressed, the variables login_id and login_pass_hash are POSTed to the page. If these variables are empty, an error message displays. If the fields are both filled in, the following query is run on the database for the entered user name:

```
SELECT user_id, password, type_id, fname FROM user WHERE username= X;
```

The login_pass_hash is compared to the password returned by the query and, if they match, a session is created with PHP which stores the username, the user's database id, the type of the user, and the user's first name. The user is then redirected to the main page for their user type. The following are the main pages for each type:

<table>
<thead>
<tr>
<th>Page</th>
<th>User Type(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skills</td>
<td>Student</td>
</tr>
<tr>
<td>Inbox</td>
<td>Instructor, Administrator</td>
</tr>
</tbody>
</table>

4.2.2 Security
This sections satisfies the requirements listed in section 2.3.1.1.4 in the Requirements Document.
Every page in the application checks for an active PHP session. If an active session is not found, the user is re-directed to the login screen. Sessions automatically time out after 20 minutes of inactivity. If a user does not have an active session they are taken to the login page. Every page in the system also has a list of user types that can access it. If the user is not allowed to access the page, they are returned to their main page and an error message displays in the ALERT AREA notifying them of the restriction. Users are not able to access pages that they are not authorized for through the links and buttons in the system. However, as all pages are public, a user could attempt to circumvent the security and these measures will curtail this.

When a user selects the menu link "Log Off", their current session is invalidated and they are returned to the "Log In" screen; the security measures prevent the user from going back to a previous page they were accessing without logging back in.

4.2.3 Menu Options
This section satisfies the requirements in section 2.3.1.2.1 in the Requirements Document.

The menu on the left side of each page will be populated using a PHP function called create_menu(). This function takes the following input parameters:
- type: the user type of the user displaying the page
- page: the page that is being displayed

The function creates the menu area for the page, populating it with a list of links to each page accessible by the user. The page variable is used to display the menu item of the current page as highlighted and with no hyperlink.

The following table lists the possible menu options and what user type(s) they appear for:

<table>
<thead>
<tr>
<th>Option</th>
<th>User Type(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skills</td>
<td>Student</td>
</tr>
<tr>
<td>Inbox</td>
<td>Instructor, Administrator</td>
</tr>
<tr>
<td>Group Management</td>
<td>Instructor, Administrator</td>
</tr>
<tr>
<td>Skill Management</td>
<td>Administrator</td>
</tr>
<tr>
<td>User Management</td>
<td>Administrator</td>
</tr>
<tr>
<td>Queries</td>
<td>Administrator</td>
</tr>
<tr>
<td>Archiving</td>
<td>Administrator</td>
</tr>
<tr>
<td>User Information</td>
<td>Student, Instructor, Administrator</td>
</tr>
<tr>
<td>Log Off</td>
<td>Student, Instructor, Administrator</td>
</tr>
</tbody>
</table>
4.2.4 Messages
This section satisfies the requirement in section 2.3.1.2.2 in the Requirements Document.

The PHP function create_alert_area() checks the HTTP GET object for the `errortext` and `successtext` variables. If these variables exist, their values are printed inside of the ALERT AREA. The text of error messages is red and the text of success messages is green. All pages will call this function to create the ALERT AREA so that error and success messages can be easily passed between pages using GET. Error messages are also created by the PHP code in a page and then displayed when the page renders.

4.2.5 Confirmation
This section satisfies the requirements in section 2.3.1.3 in the Requirements Document.

4.2.5.1 Details:
Reached from: Inbox, Skills Management, User Management, Group Management

File Name: confirm.php

HTTP GET Variables:
- `errortext` - an error message to be displayed in the ALERT AREA
- `successtext` - a success message to be displayed in the ALERT AREA

HTTP POST Variables:
- `dest_page` - the page to return to after the user has confirmed the action
- `disp_message` - the message to display to the user
- any POST variables sent to this page are forwarded to `dest_page` when the user confirms the action

4.2.5.2 Actions:
This purpose of this page is to receive user confirmation before the system commits a requested action. Thus, this page does not perform any actions. Instead, it displays a message and returns the user to the calling page with their response.

4.2.5.3 Display:
This page displays the message passed in `disp_message` to the user in the DISPLAY AREA, and provides "Continue" and "Cancel" buttons in the INTERACTION AREA. All POST variables sent to the page are forwarded to `dest_page` when the user presses the Continue button. This is accomplished by creating a form with a hidden input for each POST variable that is sent to the confirmation page.

4.2.5.4 Links:
"Continue": This button POSTs all of the hidden fields to the page stored in the `dest_page` variable.

"Cancel": This button uses JavaScript to take the user back one page in the browser history.

4.2.6 Inbox
This section satisfies the requirements in section 2.3.3.1 in the Requirements Document.
4.2.6.1 Details:
Reached from: Main Menu "Inbox" link for instructors and administrators

File Name: inbox.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successmessage - a success message to be displayed in the ALERT AREA

HTTP POST Variables:
accept_requests - a | | separated list of request ids to be approved
reject_requests - a | | separated list of request ids to be denied

4.2.6.2 Actions:
When the Inbox is loaded, PHP is used to check for the accept_requests and reject_requests POST variables.
For each request id in accept_requests, the request is approved using the following SQL statement:
UPDATE user_skill SET status = 2 WHERE request_id = X
For each request id in reject_requests, the request is deleted using the following SQL statement:
DELETE FROM user_skill WHERE request_id = X

4.2.6.3 Display:
The inbox page uses PHP to retrieve a list of all skills awaiting approval for the instructor. The PHP executes
the following queries using the user_id from the session object to find active requests:
SELECT request_id, student_id, skill_id, time_stamp, signature FROM
user_skill WHERE instructor_id = X AND status_id = 1;

For each request, the following queries will be used to determine information related to the request:

SELECT photo, fname, lname, ref_class_level.name AS class FROM
(user INNER JOIN ref_class_level ON user.level_id =
ref_class_level.level_id) WHERE user_id = student_id
SELECT name FROM (groups INNER JOIN user_belongs_to_group ON
groups.group_id = user_belongs_to_group.group_id)
WHERE groups.user_id = instructor_id AND user_belongs_to_group.user_id =
student_id
SELECT name AS skill FROM skill WHERE skill_id = user_skill.skill_id

PHP is used to create a table containing in the DISPLAY_AREA with a row for each skill request. Each row's
check box is associated with a request_id from the database. The "Accept Selected Requests" and "Reject
Selected Requests" buttons are displayed in the INTERACTION AREA.

4.2.6.4 Links:
"Student Name": Clicking this link displays the Instructor View of the Skills Page (4.2.7.2) with student_id
set to the database id of the student, and back_page set to inbox.php.

"Group Name": Clicking this link displays the Group View Page (4.2.8.2) with group_id set to the database id
of the group and back_page set to inbox.php.

"Skill Name": Clicking this link displays the Single Skill Authorization Page (4.2.7.4) with req_id set to the
database id of the skill request.
"Accept Selected Requests": If a single checkbox is selected, pressing this button behaves identically to the "Skill Name" link. If multiple checkboxes are selected, JavaScript is used to fill the accept_requests POST variable with the request id of every selected request, and this is sent to the Confirmation Page (4.2.5).

"Reject Selected Requests": This button behaves identically to the "Accept Selected Requests" button, except the reject_requests POST variable is used.

4.2.7 Skills

4.2.7.1 Student View:
This section satisfies the requirements in section 2.3.2.1 in the Requirements Document.

4.2.7.1.1 Details:
Reached from: Main Menu "Skills" link for students

File Name: skill-list.php

HTTP GET Variables:
- errortext - an error message to be displayed in the ALERT AREA
- successtext - a success message to be displayed in the ALERT AREA

HTTP POST Variables: None

4.2.7.1.2 Actions:
This page is used to display a student's skill sheet and does not perform any actions.

4.2.7.1.3 Display:
This page uses PHP to display a student's current skill sheet. The function create_skill_sheet() is used to display a skill sheet. This function takes a single input parameter, which is populated by the user_id stored in the session for this page:

- student_id - the database id of the student

The function uses the following query to retrieve a list of all skills and their categories, as well as the user's status for the skills:

```
SELECT skill.skill_id, skill.category_id, category.name AS category, skill.name AS skill, skill.active AS skill_active, category.active AS category_active, stud_skill.status FROM ((category INNER JOIN skill ON category.category_id = skill.category_id) LEFT JOIN (SELECT ref_status.name AS status, skill_id FROM user_skill INNER JOIN ref_status ON user_skill.status_id = ref_status.status_id WHERE student_id = X) AS stud_skill ON skill.skill_id = stud_skill.skill_id) ORDER BY category_id, skill_id
```

The skill sheet is displayed in a table in the DISPLAY AREA. Each category is displayed in a header row that spans all columns, has a Dark Blue background, and has White text. Skills are displayed in rows with the name of the skill in the first column, indented 10 pixels. This is followed by five columns listing the five statuses for the skill.

4.2.7.1.4 Links:
"Print": Pressing this button prints a copy of the skill sheet using JavaScript.

4.2.7.2 Instructor View:
This section satisfies the requirements in section 2.3.3.6 in the Requirements Document.

4.2.7.2.1 Details:
Reached from: Link of student's name (Inbox, Query Results, Group View)

File Name: student-details.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA
student_id - the database id of the student whose details are being viewed
back_page - the filename of the page to return to when the back button is pressed

HTTP POST Variables: None

4.2.7.2.2 Actions:
This page is used to display a student's skill sheet and does not perform any actions.

4.2.7.2.3 Display:
This page displays a student's information and skill sheet to an instructor or administrator. PHP retrieves the student's information using the following query:

```
SELECT photo, fname, lname, ref_class_level.name AS class FROM (user INNER JOIN ref_class_level ON user.level_id = ref_class_level.level_id) WHERE user_id = X
```

It displays the user's picture, name, e-mail, and class level. Then it uses the function create_skill_sheet() described in section 4.2.7.1 to display the student's skill sheet. This function is passed the student_id GET variable as its single argument.

4.2.7.2.4 Links:
"Back": Pressing this button displays the page stored in the GET variable back_page.

"Print": Pressing this button prints a copy of the skill sheet using JavaScript.

4.2.7.3 Management:
This section satisfies the requirements in section 2.3.4.1 in the Requirements Document.

4.2.7.3.1 Details:
Reached from: Main Menu "Manage Skills" link for administrators

File Name: manageskills.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA
HTTP POST Variables:
inactivate_categories  - a | separated list of category id's to be set to inactive
activate_categories  - a | separated list of category id's to be set to active
inactivate_skills  - a | separated list of skill id's to be set to inactive
activate_skills  - a | separated list of skill id's to be set to active
new_category  - the name of a new category to add
new_skill  - the name of a new skill to add
new_skill_category_id  - the id of the category a new skill is being added to

4.2.7.3.2 Actions:
If inactivate_categories is not null, the status of each category in the list is set to Inactive.
If activate_categories is not null, the status of each category in the list is set to Active.
If inactivate_skills is not null, the status of each skill in the list is set to Inactive.
If activate_skills is not null, the status of each skill in the list is set to Active.
If new_category is not null, a category is created with that name.
If new_skill is not null, a skill is created with that name in the category with id new_skill_category_id.

4.2.7.3.3 Display:
The skills management page uses the following query to obtain a list of all the categories and skills from the database:

SELECT category.category_id, skill.skill_id, category.name AS category, skill.name AS skill, category.active AS category_active, skill.active AS skill_active
FROM (category LEFT JOIN skill ON category.category_id = skill.category_id)

The page displays the categories and skills in a table in the DISPLAY AREA. Each category row will contain the buttons and text input seen in the requirements document.

4.2.7.3.4 Links:
"Add Skill": Pressing this button displays the Confirmation Page (4.2.5) with the POST variable new_skill set to the name of the skill entered in the text box and the POST variable new_skill_category_id set to the category id that belongs to the button.

"Set Skills Inactive": Pressing this button displays the Confirmation Page (4.2.5) with the POST variable inactivate_skills set to a list of the selected skill ids for the category that belongs to the button.

"Set Skills Active": This button behaves identically to the "Set Skills Active" button, except the POST variable activate_skills is used.

"Add Category": Pressing this button displays the Confirmation Page (4.2.5) with the POST variable new_category set to the name of the category entered in the text box.

"Set Categories Inactive": Pressing this button displays the Confirmation Page (4.2.5) with the POST variable inactivate_categories set to a list of the selected category ids.

"Set Categories Active": This button behaves identically to the "Set Categories Inactive" button, except the POST variable activate_categories is used.
4.2.7.4 Single Skill Authorization:
This section satisfies the requirements in section 2.3.3.2 in the Requirements Document.

4.2.7.4.1 Details:
Reached from: Main Menu "Details" link in Inbox

File Name: singleskill.php

HTTP GET Variables:
- errortext - an error message to be displayed in the ALERT AREA
- successtext - a success message to be displayed in the ALERT AREA
- req_id - the database id of the skill request being viewed
- action - the type of action being confirmed. Possible values are Accept and Reject.

HTTP POST Variables: None

4.2.7.4.2 Actions:
This page is used to display detailed information about a skill request. It does not perform any actions itself. Instead, it transfers the instructor's response back to the Inbox.

4.2.7.4.3 Display:
This page displays all details about a single skill request. PHP uses the following query to retrieve the details pertaining to the skill request whose id was passed to the page using the GET variable req_id:

```sql
SELECT photo, fname, lname, username, email, skill.name AS skill,
    time_stamp, signature FROM (SELECT * FROM user_skill INNER JOIN user ON user_skill.student_id = user.user_id WHERE request_id = req_id) AS request
INNER JOIN skill ON request.skill_id = skill.skill_id
```

The page displays the user's picture, name, e-mail, skill name, and professor signature as shown in the requirements document. Approve, Deny, and Back buttons are displayed in the INTERACTION AREA. If the GET variable action is set to "Accept," the Deny button is not displayed. Likewise, if it is set to "Reject," the Approve button is not displayed.

4.2.7.4.4 Links:
"Approve": Pressing this button sets the accept_requests POST variable to the req_id GET variable and sends it to the Inbox (4.2.6).

"Deny": This button is identical to the "Approve" button, except it uses the reject_requests POST variable.

"Back": Pressing this button returns to the Inbox (4.2.6) without sending any POST variables.

4.2.8 Groups

4.2.8.1 Group Management
This section satisfies the requirements in section 2.3.3.3 in the Requirements Document.

4.2.8.1.1 Details:
Reached from: Main Menu "Manage Groups" link for administrators and instructors
HTTP GET Variables:
  errortext - an error message to be displayed in the ALERT AREA
  successtext - a success message to be displayed in the ALERT AREA

HTTP POST Variables:
  new_group_name - the name of a new group to add
  delete_group_ids - a | separated list of group id's to be deleted

4.2.8.1.2 Actions:
If the page receives a non-null new_group_name by POST, PHP compares the value of the variable to the names of all the groups already listed for the user. If a group already exists with this name, the errortext variable is set to a message stating that a group already exists with that name and no new group will be created. If the name entered does not already exist, a new group is created in the database. PHP then sets the GET variable successtext to a success message.

If the page receives a non-empty delete_group_ids array, PHP deletes each group associated with an ID in the array from the database. PHP then sets the GET variable successtext to a success message.

Groups are added and removed from the database using the corresponding SQL statements in section 3.1.2.

4.2.8.1.3 Display:
The group management page uses the following query to retrieve a list of all of the groups belonging to the user ID stored in the session:

```
SELECT group_id, name FROM groups WHERE user_id = X
```

For each group found, PHP adds one row to the group table. Each row contains a check box, a group name, and a number of enrolled students. The number of enrolled students is retrieved using the following query:

```
SELECT COUNT(user_belongs_to_group.user_id) AS students FROM (user_belongs_to_group INNER JOIN (SELECT user_id FROM user WHERE active = 1) AS active_users ON user_belongs_to_group.user_id = active_users.user_id) WHERE group_id = group_id GROUP BY group_id;
```

If no groups are found, the message "You do not have any groups." displays instead.

4.2.8.1.4 Links:
"Group Name": Clicking this link displays the Group View Page (4.2.8.2) for the selected group with group_id set to the database id of the group and back_page set to managegroups.php

"Create New Group": When pressed, this button executes a JavaScript function to check if the text has been entered into the "New Group Name" field. If no text has been entered, the page is reloaded with an error message set in the errortext variable. If text has been entered it is POSTed to the page as the new_group_name variable.

"Delete Selected Groups": When pressed, this button executes a JavaScript function to check if one or more check boxes have been checked. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and and the array of selected group
IDs is POSTed to the page as the delete_group_ids variable.

4.2.8.2 Group View
This section satisfies the requirements in section 2.3.3.4 in the Requirements Document.

4.2.8.2.1 Details:
Reached from: Group name link for administrators and instructors in group list or Inbox

File Name: viewgroup.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA
group_id - the database id of the group being viewed
back_page - the filename of the page to return to when the back button is pressed

HTTP POST Variables:
rem_student_ids - a | separated list of user ids to be removed from the current group

4.2.8.2.2 Actions:
If the page receives a non-empty rem_student_ids array, PHP removes each association between the group identified by group_id and an ID from rem_student_ids from the database. PHP then sets the GET variable successtext to a success message.

Students are removed from groups using the corresponding SQL statement in section 3.3.1.

4.2.8.2.3 Display:
The group view page uses the following query to retrieve a list of all of the students in the group with the ID in the GET variable group_id:

```sql
SELECT user.user_id, fname, lname, username, email, ref_class_level.name AS level
FROM ((user INNER JOIN user_belongs_to_group ON user.user_id = user_belongs_to_group.user_id)
INNER JOIN ref_class_level ON user.level_id = ref_class_level.level_id)
WHERE group_id = X AND user.active = 1 ORDER BY lname
```

For each student found, PHP adds one row to the student table. Each row contains a check box, a photo, a student name, a student username, an email address, and a class level. The photo displayed is either the 'username'.jpg file stored in the user images folder (where 'username' matches the username retrieved from the query) or the default 'awaiting photo' image if the user has no stored picture.

4.2.8.2.4 Links:
"Student Name": Clicking this link displays the Instructor View of the Skills Page (4.2.7.2) with student_id set to the database id of the student, and back_page set to viewgroup.php?group_id=group_id.

"Remove Selected Students": When pressed, this button executes a JavaScript function to check if one or more check boxes have been checked. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and the array of selected student IDs is POSTED to the page as the rem_student_ids variable.
"Add Students to Group": When pressed, this button displays the Add Students to Group Page (4.2.8.3) with the group_id variable set.

"Back": When pressed, this button loads the page given by the back_page variable.

4.2.8.3 Add Students to Group
This section satisfies the requirements in section 2.3.3.5 in the Requirements Document.

4.2.8.3.1 Details:
Reached from: "Add Students to Group" link on group view page

File Name: addstudstogroup.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA
group_id - the database id of the group that students are being added to

HTTP POST Variables:
add_student_ids - a | separated list of user ids to be added to the current group

4.2.8.3.2 Actions:
If the page receives a non-null add_student_ids array, PHP associates the student IDs in the array with the current group_id in the database. PHP then redirects the user to the Group View page, defined in section 4.2.8.2, with the current group_id variable and the GET variable successtext set to a success message.

Students are added to groups using the corresponding SQL statement in section 3.3.1.

4.2.8.3.3 Display:
This page uses the following query to get a list of all students already in the current group:

```
SELECT user.user_id FROM (user INNER JOIN user_belongs_to_group ON user.user_id = user_belongs_to_group.user_id) WHERE group_id = group_id
```

PHP stores these students in an array.

The page then uses the following query to get a list of all students from the database:

```
SELECT user_id, fname, lname, username FROM user WHERE type_id = 1 AND active = 1 ORDER BY lname
```

PHP adds a single row to the HTML table for each student that is not already in the array of students in the group. Each row contains a checkbox, a photo, a name, and a student username. The photo displayed is either the 'username'.jpg file stored in the user images folder (where 'username' matches the username retrieved from the query) or the default 'awaiting photo' image if the user has no stored picture.

4.2.8.3.4 Links:
"Add Users to Group": When pressed, this button executes a JavaScript function that checks to see if one or more check boxes were selected. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and the selected student IDs are POSTed to the page as the add_student_ids variable.
"Back": When pressed, this button takes users back to the Group View Page (4.2.8.2) for the current group.

4.2.9 Queries
This section satisfies the requirments in section 2.3.4.3 in the Requirements Document.

4.2.9.1 Query Building
This section satisfies the requirements in section 2.3.4.3.2 in the Requirements Document.

4.2.9.1.1 Details:
Reached from: "Queries" link in menu for administrators

File Name: query.php

HTTP GET Variables:
- errortext - an error message to be displayed in the ALERT AREA
- successtext - a success message to be displayed in the ALERT AREA

HTTP POST Variables:
- save - value indicating if the 'Save Query' button was pressed.
- constraint_count - array which holds the number of constraint rows currently displayed on the page
- datafield(1-n) - options in the 'Field' drop down box
- operator(1-n) - options in the 'Operator' drop down box
- value(1-n) - options in the 'Value' drop down box
- query_name - text entered into the 'Query Name' field
- delete_query - array of query id's to be deleted

4.2.9.1.2 Actions:
If delete_query is not null, each query in the array is deleted.

If save is true, PHP is used to build a query. The query selects for users based on the constraints. Each datafield[i], operator[i], and value[i] makes up a single constraint. The variable constraint_count indicates how many constraints there are. This query is saved using the user id from the session variable and the name specified by query_name.

4.2.9.1.3 Display:
This page uses the following query to retrieve a list of the saved queries for the user ID stored in the session:

SELECT query_id, query, name FROM query WHERE user_id = X

These queries are listed in a section at the top (or the message "No saved searches found." if the result of the query is empty). Each saved query is listed by 'name' with a check box associated with it. Beneath this list is a "Delete Queries" button. Below the saved queries is an "Add Constraint" button followed by a table of constraints for a query. Each row in this table consists of a drop down box with 'Field' options (a query will find users sorted by 'Field' value), a drop down box with 'Operator' options (such as "is", "is not", ">="), a drop down box with 'Value' options (a query will return entries where the 'Field' value matches/doesn't match/is greater than/is less than the 'Value' option), and a "Remove Constraint" button. Below the list of constraints is a 'Query Name' text field and a "Save Query" button.
4.2.9.1.4 Links:
"Delete Queries": Pressing this button executes a JavaScript function to check if one or more check boxes have been checked. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and the array of selected user IDs is POSTed to the page as the delete_query variable.

"Add Constraint": When pressed, this button executes a JavaScript function which generates a new constraint row.

"Remove Constraint": When pressed, this button executes a JavaScript function to remove a constraint row. If only one constraint row exists and this button is pressed, an error message displays instead and the final row is not removed.

"Run Query": When pressed, this button POSTs the variables constraint_count, datafield, operator, and value to the runquery.php page.

"Save Query": When pressed, this button POSTs the variables constraint_count, datafield, operator, and value and reloads this page.

4.2.9.2 Query Results
This section satisfies the requirements in section 2.3.4.3.3 in the Requirements Document.

4.2.9.2.1 Details:
Reached from: "Run Query" button on the Query Building page

File Name: runquery.php

HTTP GET Variables:
 saved_id - the database id of a query to run

HTTP POST Variables:
 export - value indicating if the "Export Query" button was pressed
 constraint_count - array which holds the number of constraint rows currently displayed on the page
 datafield(1-n) - options in the 'Field' drop down box
 operator(1-n) - options in the 'Operator' drop down box
 value(1-n) - options in the 'Value' drop down box

4.2.9.2.2 Actions:
If the export variable is set, PHP is used to construct a query as described in the Display section. PHP then creates a text file containing the query results according to the format described in section 4.2.9.3.

4.2.9.2.3 Display:
The run query page dynamically constructs a SQL query when a set of constraints and the number of constraints are POSTed to it. PHP generates the SQL statement by iterating from 1 to n, where n is the number of constraints. The SQL statement generated is then run. If the save_id variable is non-empty then the SQL statement associated with this ID is retrieved and run instead. First, a count of the number of students found is displayed. Beneath this line is an HTML table containing the user information for all students found to match the criteria given. Below the table is an "Export Query" button. Below this is a 'Query Name' text field and a "Save Query" button.
4.2.9.4 Links:
"Export Query": Pressing this button POSTs the export variable and reloads the page.

"Save Query": Pressing this button POSTs the variables constraint_count, datafield, operator, and value to the query.php page and redirects the user to that page.

4.2.9.3 Exporting Queries

4.2.9.3.1 Details:
Reached from: "Export Results" button on the Query Results page

File Name: export.txt

HTTP GET Variables:
saved_id - the database id of a query to export

HTTP POST Variables:
constraint_count - array which holds the number of constraint rows currently displayed on the page
datafield(1-n) - options in the 'Field' drop down box
operator(1-n) - options in the 'Operator' drop down box
value(1-n) - options in the 'Value' drop down box

4.2.9.3.2 Display:
The export page dynamically constructs a tab-delimited text file containing all of the entries found to match the given criteria. The file will have the following format for each line:

student_name  student_id  user_id  clinical_locations  clinical_instructors
student_level

Each of these fields are separated by a single tab and each entry occurs on a separate line.

4.2.10 User Management
This section satisfies the requirements in section 2.3.4.2 in the Requirements Document.

4.2.10.1 Details:
Reached from: Main Menu "User Management" entry for administrators

File Name: manageusers.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA
display - the type of user to display. Possible values are Student, Instructor, and Administrator.
order - the way to order the user list. Possible values are name, class, and active.

HTTP POST Variables:
userfile - the file containing information for new users to add
add_user - the value indicating if the 'Add User' button was pressed
4.2.10.2 Actions:
If the add_user variable is set but the userfile variable is empty or not a valid user file, PHP sets the GET variable errortext to an error message. Valid user files are tab-delimited in the following format:

first_name last_name email user_name class_level

If the user file is valid, PHP parses each row in it and first checks if the user identified by 'user_name' is already in the database. If not, PHP generates a random password for the user and also hashes the password using MD5 hashing. PHP then inserts a new [USER] into the database, where [USER] is either a Student, Instructor, or Administrator and corresponds to the value of the display variable. (The 'class_level' is optional and will be set to 1 for Freshman if not entered). The hash of the generated password is included in the user's information. A new user is added to the database using the corresponding SQL statements in section 3.1.1. After each user is added, PHP sends an email to the user informing them of their SITS username and automatically generated password. This mail is generated using the built in PHP function 'mail($to, $subject, $body)' where $to is set to the added user's 'email', $subject is set to the string, "Welcome to SITS", and $body is set to the message.

If the add_picture variable is set but the pic_file variable is empty or not a valid .jpg file name, PHP sets the GET variable errortext to an error message. If the pic_file variable is a valid .jpg file, the file is copied to the 'student_images' folder and renamed as 'username'.jpg. PHP then sets the GET variable successtext to a success message.

If the page receives a non-empty set_class_level array, PHP changes the selected students' class level to the POSTed class_level using the following SQL statement:

UPDATE user SET level_id = class_level_id WHERE user_id = INT;

PHP then sets the GET variable successtext to a success message.

If the page receives a non-empty activate_users variable, PHP sets the selected users' status to 'Active' using the SQL statement specified in section 3.1.1.10. PHP then sets the GET variable 'successtext' to a success message.

If the page receives a non-empty inactivate_users variable, PHP sets the selected users' status to 'Inactive' using the SQL statement specified in section 3.1.1.10. PHP then sets the GET variable successtext to a success message.

4.2.10.3 Display:
The user management page displays a table listing all users in the database. The user can view either students, instructors, or administrators depending on the value of the GET variable display. Each user is displayed in a row with a checkbox, their picture (if provided), name, user name, a file chooser, and an "Add Picture" button. The table is sorted by 'Name', 'Active/Inactive' or 'Class Level' (for Students) depending on the value of the GET variable order. The information for the table is retrieved using the following SQL query:
At the top of the page, above the table of users, is a file chooser to select a user file. The 'Name', 'Class Level' (for Students), and 'Active/Inactive' column headers function as links. In the bottom area of the page are buttons for setting student's class level, setting users as 'Active' or 'Inactive', and selecting which type of user to view on the page.

4.2.10.4 Links:
"Add [USER]s": When pressed, this button executes a JavaScript function that sets the value of the userfile variable to the text entered into the "Filename" file chooser. The add_user and userfile variables are then POSTed to the page.

"Student Name": Clicking this link displays the Instructor View of the Skills Page (4.2.7.2) with student_id set to the database id of the student, and back_page set to manageusers.php?display=Student&order

"Add Picture": When pressed, this button executes a JavaScript function that sets the value of the username variable to the username of the user in the same row as the button that was pressed and the value of the pic_file variable to the text entered into the corresponding file chooser. The add_picture, username, and pic_file variables are then POSTed to the page.

'Name', 'Class Level', 'Active/Inactive': When one of these row headers is pressed, PHP sets the the GET variable order and reloads the page. Ordering by 'Name' displays users in alphabetical order by last name, ordering by 'Active/Inactive' displays all Active users followed by all Inactive users, and ordering by class level displays Students in class level order (Freshman, Sophomore, Junior, Senior, Graduated, Dropped).

"Set Class Level": This button and the accompanying drop down box are only active when the display variable is set to "Student". When pressed, this button executes a JavaScript function to check if one or more check boxes have been checked. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and the array of selected user IDs is POSTed to the page as the select_class_level variable.

"Set Active": When pressed, this button executes a JavaScript function to check if one or more check boxes have been checked. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and the array of selected user IDs is POSTed to the page as the activate_users variable.

"Set Inactive": When pressed, this button executes a JavaScript function to check if one or more check boxes have been checked. If this is not true, the page is reloaded with an error message set in the errortext variable. If it is true, the Confirmation Page (4.2.5) is loaded and the array of selected user IDs is POSTed to the page as the inactivate_users variable.

"Student", "Instructor", "Administrator": When one of these radio buttons is selected, PHP sets the GET variable display to either "Student", "Instructor", or "Administrator" and reloads the page.

4.2.11 User Information
This section satisfies the requirements in section 2.3.1.4 in the Requirements Document.
4.2.11.1 Details:
Reached from: Main Menu "User Information" entry

File Name: userinfo.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA

HTTP POST Variables:
submit_pass - value indicating if the 'Set Password' button was pressed
orig_pass_hash - MD5 hash of the user's current password as entered in the "Current Password" field
new_pass_hash - MD5 hash of the user's desired new password as entered in the "New Password" field
submit_email - value indicating if the 'Set Email' button was pressed
old_email - the user's current email address
new_email - the user's desired new email address as entered in the "New Email" field

4.2.11.2 Actions:
When the submit_pass variable is set, PHP checks the orig_pass_hash against the user's current password in the database. If these do not match, the GET variable errortext is set to an error message. If they do match, this hash is replaced with the new_pass_hash variable. A user's password is changed in the database using the following SQL statement:

```
UPDATE user SET password = new_pass WHERE user_id = X;
```

When the submit_email variable is set, PHP sends an email to both the old_email and new_email addresses with a message to the user that their email address has been changed and they should log in to SITS and access the 'User Information' page if the new address is incorrect. These emails are generated using the built in PHP function 'mail($to, $subject, $body)' where $to is set to either old_email or new_email, $subject is set to the string, "SITS E-Mail Change Notification", and $body is set to the message. The user's email address is then changed using the following query:

```
UPDATE user SET email = 'new_email' WHERE user_id = X;
```

4.2.11.3 Display:
This page displays the user's name, username and current email address. It has two sections for updating a user's password or email address.

4.2.11.4 Links:
"Set Password": When pressed, this button executes a JavaScript function which first checks to see if all three text fields ("Old Password", "New Password", and "Repeat New Password") have text entered. If one of more fields is empty, an error message is set in the ALERT area. Next, the function checks to see if the two entries of the new password match. If they do not, an error message is set in the ALERT area. If the two passwords do match, the original password and the new password are hashed with JavaScript using MD5 hashing and then POSTed to the page, along with the submit_pass variable.

"Set Email": When pressed, this button executes a JavaScript function to check if text has been entered into the "New Email" field; if not, an error message is set in the ALERT area. Next, the function checks if the text
entered is in the valid email format of 'NAME'@'DOMAIN'.'SUFFIX' using a regular expression. If the text is not in the proper format, an error message is set in the ALERT area. If the text entered is in the valid email format, the variables submit_email, old_email, and new_email are POSTed to the page.

4.2.12 Archiving
This section satisfies the requirements in section 2.3.4.4 in the Requirements Document.

4.2.12.1 Details:
Reached from: Main Menu "Archiving" entry for administrators

File Name: archive.php

HTTP GET Variables:
errortext - an error message to be displayed in the ALERT AREA
successtext - a success message to be displayed in the ALERT AREA

HTTP POST Variables:
delete_archived - value indicating if the 'Delete Archived Students' button was pressed
create_arch - value indicating if the 'Archive Inactive Students' button was pressed
unarchive - value indicating if the 'Restore Students from Archive File' button was pressed
archive_file - the file containing information for previously archived users to re-add

4.2.12.2 Actions:
If the delete_archived POST variable is set to 'true', PHP executes the following query:

SELECT user_id FROM user WHERE archived = 1 AND active = 0;

PHP iterates through each of these entries and removes the users from the database. This includes deleting the user's stored picture file, if it it exists, and performing the following queries:

DELETE FROM user_skill WHERE student_id = X;
DELETE FROM user_belongs_to_group WHERE user_id = X;
DELETE FROM user WHERE user_id = X;

If the create_arch POST variable is set, PHP uses the following query to retrieve all students in the database with a status of 'Inactive':

SELECT * FROM user WHERE active=0 AND type_id=1 ORDER BY user.user_id;

PHP then generates a text file that can be saved to the hard drive which stores the results of the query. This file has the following format:

first_name last_name email user_name pass_hash user_type class_level

skill_status

skill_status is a list of skills are their statuses in the following form:

skill_id^^status_id^^instructor_id^^time_stamp!!skill_id^^status_id^^instructor_id^^time_stamp...

If the unarchive POST variable is set, but the archive_file variable is empty or not a valid archive file (as
described above), PHP sets the GET variable errortext to an error message. If the archive file is valid, PHP parses each row in it and first checks if the user identified by 'user_name' is already in the database. If not, PHP inserts the user into the database and restores their skill statuses using the following queries:

\[
\text{INSERT INTO user (fname, lname, email, username, password, type_id, level_id) VALUES}
\]
\[
('first_name', 'last_name', 'email', 'user_name', 'pass_hash', 'user_type', 'class_level');
\]
\[
\text{SELECT user_id FROM user WHERE username='user_name';}
\]
\[
\text{INSERT INTO user_skill (student_id, skill_id, status_id, instructor_id, time_stamp) VALUES ('user_id', 'skill_id', 'status_id', 'instructor_id', 'time_stamp');}
\]

4.2.12.3 Display:
The archiving page is used to archive inactive students, delete archived students, restore students from an archive file. The page displays buttons for deleting archived students and archiving inactive students as well as a file chooser for selecting an archive file and an accompanying button for restoring students from the archive file.

4.2.12.4 Links:
"Delete Archived Students": Pressing this button POSTs the delete_archived variable as 'true' to the Confirmation Page.

"Archive Inactive Students": Pressing this button POSTs the create_arch variable as 'true' and reloads the page.

"Restore Students from Archive File": Pressing this button executes a JavaScript function that sets the value of the archive_file POST variable to the text entered into the file chooser. The unarchive and archive_file POST variables are also set and the page is reloaded.

4.2.13 Syncing

4.2.13.1 Receiving Skill Requests

File Name: request.php

HTTP POST variables:

- pass_hash - a string containing the MD5 hash of the user's password
- file1 - a file reference to the XML file
- file2...filex - a number of file references to GIF files containing signatures

This PHP page accepts a multi-part HTTP POST message containing a single XML file and multiple binary GIF files. The XML file will be decrypted and then parsed according to its definition in section 6.1.2.1. For each request, the PHP will check that the user name of the student and professor are users who exist in the database and that the requested skill exists in the database. If both of these are true, a skill request can be successfully generated and this success will be noted in the return XML. If any of these conditions is not true it will result in a failure message being included in the response XML. For each successful request, its corresponding image file will be copied to the server and renamed with a random 40 character string which will have ".gif" appended to it. This name will be inserted into the signature file table in the database and associated with the skill request. The return XML will contain the success or failure status of all skill requests made.
4.2.13.2 Retrieving Skill Sheets

File Name: skillsheet.php

HTTP POST variables:
   skillreq - a file reference to the XML file containing user information

This PHP page accepts a single XML file as input. It decrypts and parses the XML for the user name and password hash of a student. If the user name and password hash match the database entries, then an XML skill sheet is formed from the database using the format given in section 6.1.3.2. If there is a mismatch, an error is returned in the XML instead.

4.3 Diagrams

4.3.1 Group Management Flow
The following diagram shows how the various pages relating to group management work together.

![Group Management Flow Diagram](image)

*Figure 4.3.1 Group Management Flow*

4.3.2 User Management Flow
The following diagram shows how the various pages relating to user management work together.
4.3.3 Skill Management Flow
The following diagram shows how the various pages relating to skill management work together.
4.3.4 Query Building
The following diagram shows how query building works.

Figure 4.3.4 Query Building
5. PDA Application
This section satisfies the requirements in section 2.4 in the Requirements Document.

The PDA application will be developed using the Microsoft Windows CE Pocket PC 5.0 SDK in Microsoft Visual Studio 2005. The programming will be done using the language C# and will also integrate portions of the Microsoft Tablet PC Platform SDK for the capturing of the signature. The mentioned SDKs can be found with documentation and sample code from www.microsoft.com. The PDA application will consist of four files:

- SITS.exe - This is the main executable file for the PDA application.
- UserData.xml - This is an XML file storing the user's info. The formatting of this file is explained in section 6.1.3
- SkillSheet.xml - This is an XML file storing the user's current skill sheet. The formatting of this file is explained in section 6.1.3
- SkillRequests.xml - This is an XML file storing the user's pending skill requests. The formatting of this file is explained in section 6.1.2

5.1 PDA Application Installation & Environment

Installation:
The installation package will consist of the self-extractable archive file. Once this file is extracted it will create the application home directory "SITS_PDA_Application" in the PDA's application folder. To run the PDA application the user will choose the "SITS" icon in the application folder.

Environment:
The PDA application will run on Windows Mobile 5 with the most recent version of the .NET framework (v2.0 at this printing). Using the form object within this framework screens will be created to handle the creation of users, signing off on skills, and synchronizing the data to the CNHP server. Further flow of how these forms are handled is displayed in the story board model in section 5.3.2.

Two forms will require Internet connections to transfer information to the server. These include:

- Create User From
- Syncing Form

The Create User form will only be displayed when there is no UserData present. This will then ask the pda owner for information and tie the application to their username. Alternatively, this screen can be re-accessed by deleting the UserData.xml and reopening the application. This will then tie a new user to the application on that PDA. Immediately following the Create User form focus will be shifted to the Login form. From here a user can proceed to login and get skills approved, or select Sync and go to the Syncing form. From here the user can submit requests and update their skillsheet.

5.2 PDA Application System Design
The System Design of the PDA application is in C# and split into multiple files. Each file is explained below.

- Program.cs
- UserData.cs
5.2.1 Program Class
This class contains all the objects needed to run the PDA component, including the list of categories, which in turn contains the list of skills, all of the display screens for the interface, and the user data. This class is called anytime the SITS program is run and is responsible for loading all data needed for the program. This class also creates, loads, and saves owner data, generates and saves skill requests, and encrypts the XML files which will be sent to the Web.

5.2.1.1 Fields:
5.2.1.1.1 category_list
The category_list is a List of the Category Object described in section 5.2.3. This is a copy of all the categories and skills within the system.

5.2.1.1.2 category_screen
The category_screen is an object of CategoryScreen described in section 5.2.6.

5.2.1.3 login_screen
The login_screen is an object of LoginScreen described in section 5.2.5.

5.2.1.4 skill_screen
The skill_screen is an object of SkillScreen described in section 5.2.7.

5.2.1.5 submit_screen
The submit_screen is an object of SubmitScreen described in section 5.2.8.

5.2.1.6 user
The user is an object of UserData described in section 5.2.2. The user object holds all information specific to the user.

5.2.1.2 Methods:

5.2.1.2.1 CreateOwner
The CreateOwner method is called when there is no UserInfo.xml file. It will prompt the user to input their username, password, and name and then create a new UserInfo.xml file containing the inputted data.

5.2.1.2.2 EncryptXmlFile
The EncryptXmlFile method is called to apply the 3DES Encryption to an XmlFile as it is being saved. This will be applied to the SkillList.xml and Requests.xml files.

5.2.1.2.3 GenerateRequest
The GenerateRequest method will take in a Skill and add this skill to the SkillRequests.xml file.

5.2.1.2.4 LoadCategoryList
The LoadCategoryList method will fill the category_list field (5.2.1.1) by parsing the SkillSheet.xml file. Each Category object will be added and filled with its list of Skill objects.

5.2.1.2.5 LoadRequests
The LoadRequests method will modify the category_list field (5.2.1.1) by parsing the SkillRequests.xml file and using it to update the skill status. This will be used to display to the user what skills are currently being requested.

5.2.1.2.6 LoadUserData
The LoadUserData method will create an instance of a UserData object and invoke the ImportFromFile method explained in section 5.2.2.7.

5.2.1.2.7 SaveRequests
The SaveRequests method will close and save the SkillRequests.xml file. This will also call the method EncryptXmlFile (5.2.1.2.2) to encrypt the data.

5.2.1.2.8 SaveUserData
The SaveUserData method will close and save the UserInfo.xml file. This will also call the method EncryptXmlFile (5.2.1.2.2) to encrypt the data.

5.2.2 User Data Class
This class stores the PDA owner's data for use with the SITS program. This class can update and retrieve the
user's name, password, and username as well as import this data from a file.

5.2.2.1 Fields:

5.2.2.1.1 username
The username field will store the user’s username.

5.2.2.1.2 password
The password field will store the user’s password.

5.2.2.1.3 name
The student_name field will store the user's first and last name.

5.2.2.2 Methods:

5.2.2.2.1 getPassword
The getPassword method will return the user's password.

5.2.2.2.2 getUsername
The getUsername method will return the user's username.

5.2.2.2.3 getName
The getName method will return the user's name.

5.2.2.2.4 setPassword
The setPassword method will take in a string and set the user's password to that string.

5.2.2.2.5 setUsername
The setUsername method will take in a string and set the user's username to that string.

5.2.2.2.6 setName

Figure 5.2.2 User Data Class
The setName method will take in a string and set the user's name to that string.

5.2.2.2.7 ReadUserData
The ReadUserData method will decrypt the UserInfo.xml using the Triple DES method described in section 7.2, then read in the username password stored in that file.

5.2.2.2.8 WriteUserData
The WriteUserData method will write the User Object to the file Request.xml and then encrypt that file using the Triple DES method described in section 7.2.

5.2.3 Category Class
This class holds a category and the associated list of skills for that category. The category name, a list of skills under that category, and a specific skill name can be retrieved with this class.

5.2.3.1 Fields:
5.2.3.1.1 name
The name field holds the name of the category.

5.2.3.1.2 skill_list
The skill_list field is a List containing each skill contained within the category.

5.2.3.2 Methods:
5.2.3.2.1 addSkill
The addSkill method will take in a skill and append it as an additional skill to the category.

5.2.3.2.2 getName
The getName method returns the name of the category.

5.2.3.2.3 setName
The `setName` method sets the name of the current category.

5.2.3.2.4 `getSkills`
The `getSkills` method will return the List of skills contained by the category.

5.2.3.2.5 `getSize`
The `getSize` method will return the size of the `skill_list` array.

5.2.4 Skill Class
This class holds a skill name and the user's status for that skill. Both of these pieces of data can be updated and retrieved using this class.

![Figure 5.2.4 Skill Class](image)

5.2.4.1 Fields:

5.2.4.1.1 `name`
The name field holds the name of the skill.

5.2.4.1.2 `trials`
The trials field is a list of strings, each of which contains the status of the corresponding stage of skill. The values this field should accept are explained in section 2.2.2 of the SITS requirements doc.

5.2.4.2 Methods:

5.2.4.2.1 `getName`
The `getName` method returns the name of the skill.

5.2.4.2.2 `getTrials`
The `getTrials` method returns a list containing the current statuses of each stage for the current skill.

5.2.4.2.3 `setName`
The `setName` method takes in a string and sets it as the name of the skill.

5.2.4.2.4 `addTrial`
The `addTrial` method takes in a string and sets the it as the status of the next unassigned stage of the skill to
its value.

5.2.5 Login Screen Class
This section satisfies the requirements in section 2.4.1 in the Requirements Document.

![Login Screen Class Diagram]

Figure 5.2.5 Login Screen Class

5.2.5.1 Fields:

5.2.5.1.1 b_exit
The user presses this button to exit the application.

5.2.5.1.2 b_login
The user presses this button to login to the application.

5.2.5.1.3 b_sync
The user presses this button to open the sync application.

5.2.5.1.4 l_username
The l_username field is a label containing the text "Username".

5.2.5.1.5 t_username
The t_username field is a textbox that that accepts the user’s username.

5.2.5.2 Methods:

5.2.5.2.1 b_exit_Click
This is the method called when the exit button is clicked. This will initiate the ProcessExit method (5.2.5.2.4).

5.2.5.2.2 b_login_Click
This is the method called when the exit button is clicked. This will initiate the ProcessLogin method.
5.2.5.2.3 **LoginScreen**
The LoginScreen constructor creates an instance of the LoginScreen form for taking input.

5.2.5.2.4 **ProcessExit**
This process will exit the program.

5.2.5.2.5 **ProcessLogin**
This process will make sure the username entered in the *user object (5.2.5.1.5)* matches the one in the UserData object stored in Program *(5.2.1.1.6)*. If these do not match an error message is displayed. If they do match, the focus is then changed to the **CategoryScreen object(5.2.6)**.

5.2.5.3 **Action Listeners:**

5.2.5.3.1 **buttonClick**
The LoginScreen object has an action listener for button clicked.

5.2.6 **Category Screen Class**
This section satisfies the requirements in section 2.4.2 in the Requirements Document.

![CategoryScreen Class Diagram](image)

**Figure 5.2.6 Category Screen Class**

5.2.6.1 **Fields :**

5.2.6.1.1 **b_back**
The user presses this button to go back in the application. This will initiate the **ProcessBack method (5.2.6.2.5)**.

5.2.6.1.2 **b_select**
The user presses this button to select the current category. This will initiate the **ProcessSelect method** (5.2.6.2.6).

5.2.6.1.3 category_list
This is a ListBox object that is populated with the category_list object in the main Program (5.2.1.1.1). When a new category is selected, the method **Category_SelectedIndexChanged (5.2.6.2.3)** will be called.

5.2.6.1.4 l_category_list
The l_category_list is a field that stores the currently selected category. This will be set by **Category_SelectedIndexChanged (5.2.6.2.3)**.

5.2.6.2 Methods :

5.2.6.2.1 b_back_Click
This is the method called when the back button is clicked. This will initiate the **ProcessBack method** (5.2.6.2.5).

5.2.6.2.2 b_select_Click
This is the method called when the select button is clicked. This will initiate the **ProcessSelect method** (5.2.6.2.6).

5.2.6.2.3 Category_SelectedIndexChanged
This is the method called when the selected item in the **category_list (5.2.6.1.3)** is changed. Upon this change l_category_list will update to display the name of the selected category.

5.2.6.2.4 CategoryScreen
The CategoryScreen constructor creates an instance of the CategoryScreen form for taking input.

5.2.6.2.5 ProcessBack
This method will return the focus to the **Login Screen (5.2.5)**

5.2.6.2.6 ProcessSelect
This method will take in the currently selected category from the category_list and then create a new instance of SkillScreen and pass in the category. Focus will then be changed to display the SkillScreen form.

5.2.6.3 Action Listeners :

5.2.6.3.1 buttonClicked
The CategoryScreen object will have a buttonClicked action listener that is triggered when either the b_back button (5.2.6.1.1) or b_select button(5.2.6.1.2) is clicked.

5.2.6.3.2 IsItemChanged
The **category_list object (5.2.6.1.4)** will have an IsItemChanged action listener that triggers when a new category is selected from the list.

5.2.7 Skill Screen Class
This section satisfies the requirements in section 2.4.3 in the Requirements Document.
5.2.7.1 Fields:

5.2.7.1.1 b_back
The user presses this button to go back in the application. This will initiate the ProcessBack method (5.2.7.2.3).

5.2.7.1.2 b_select
The user presses this button to select the current skill. This will initiate the ProcessSelect method (5.2.7.2.4).

5.2.7.1.3 l_stage_one
This is the label displaying the selected skill's status for stage one.

5.2.7.1.4 l_stage_two
This is the label displaying the selected skill's status for stage two.

5.2.7.1.5 l_stage_three
This is the label displaying the selected skill's status for stage three.

5.2.7.1.6 l_stage_four
This is the label displaying the selected skill's status for stage four.

5.2.7.1.7 l_stage_five
This is the label displaying the selected skill's status for stage five.

5.2.7.1.8 l_status
This is a label predefined with the string "Status".
5.2.7.1.9 skill_list
This is a ListBox object that is populated with the skills contained in the category passed into SkillScreen. When a new skill is selected from this ListBox the method `skill_list_SelectedIndexChanged(5.2.7.2.5)` will be called.

5.2.7.2 Methods:

5.2.7.2.1 b_back_Click
This is the method called when the back button is clicked. This will initiate the `ProcessBack method (5.2.7.2.3)`.

5.2.7.2.2 b_select_Click
This is the method called when the select button is clicked. This will initiate the `ProcessSelect method (5.2.7.2.4)`.

5.2.7.2.3 ProcessBack
This method will return the focus to the `CategoryScreen (5.2.6)`

5.2.7.2.4 ProcessSelect
This method will take in the currently selected skill from the skill_list and then create a new instance of `SubmitScreen (5.2.8)` and pass in the skill. Focus will then be changed to display the SubmitScreen form.

5.2.7.2.5 skill_list_SelectedIndexChanged
This is the method called when the selected item in the `skill_list (5.2.7.1.9)` is changed. Upon this change, the labels `l_stage_one`, `l_stage_two`, `l_stage_three`, `l_stage_four`, and `l_stage_five` will be updated to display the user’s status for each stage for the selected skill.

5.2.7.2.6 SkillScreen
The SkillScreen constructor creates an instance of the SkillScreen form for taking in input.

5.2.7.3 Action Listeners:

5.2.7.3.1 buttonClicked
The SkillScreen object will have a buttonClicked action listener that is triggered when either the `b_back button (5.2.7.1.1)` or `b_select button(5.2.7.1.2)` is clicked.

5.2.7.3.2 IsItemClicked
The `skill_list object (5.2.7.1.9)` will have an IsItemChanged action listener that triggers when a new skill is selected from the list.

5.2.8 Submit Screen Class
This section satisfies the requirements in section 2.4.4 in the Requirements Document.
5.2.8.1 **Fields:**

5.2.8.1.1 **b_back**
The user presses this button to go back in the application. This will initiate the ProcessBack method (5.2.8.2.5).

5.2.8.1.2 **b_submit**
The user presses this button to submit a request for the selected skill. This will initiate the ProcessSubmit method (5.2.8.2.6).

5.2.8.1.3 **b_clear**
The user presses this button to clear any writing that appears within the cSignature Panel (5.2.8.1.4).

5.2.8.1.4 **cSignature**
This is a Signature Object. This is a panel that will be used to capture a professor's signature.

5.2.8.1.5 **l_professor**
This is a label with the predefined value of "Professor's Drexel User Id".

5.2.8.1.6 **l_signature**
This is a label with the predefined value of "Signature:"

5.2.8.1.7 **l_skill**
This is a label for displaying the skill that was selected on the SkillScreen form (5.2.7).
5.2.8.1.8 l_studentname
This is a label for displaying the student's name. This is pulled from the UserData object (5.2.2).

5.2.8.1.9 l_username
This is a label for displaying the student's username. This is pulled from the UserData object (5.2.2).

5.2.8.1.10 t_faculty_id
This is a textbox used to take in the professors Drexel user id.

5.2.8.1.11 cmbColorList
This is list box that contains the list of colors of the signature.

5.2.8.2 Methods:

5.2.8.2.1 b_back_Click
This is the method called when the back button is clicked. This will initiate the ProcessBack method (5.2.8.2.5).

5.2.8.2.2 b_submit_Click
This is the method called when the submit button is clicked. This will initiate the checkGenerateError method (5.2.8.2.3).

5.2.8.2.3 checkGenerateError
This method will verify the following:

1. There is text in the t_faculty_id text box (5.2.8.1.10).
2. The Requests.xml files exists.

If each of these pass, no error is generated and the ProcessSubmit method (5.2.8.2.6) will be invoked.

5.2.8.2.4 GenerateImageSig
This method is called with this ProcessSubmit Method, this will then use the cSignature field to save the signature with in it to a gif file.

5.2.8.2.5 ProcessBack
This method will return the focus to the SkillScreen form (5.2.7) with the previous skill selected.

5.2.8.2.6 ProcessSubmit
This method will invoke the GenerateRequest method from the main Program (5.2.1).

5.2.8.2.7 SubmitScreen_load
The SubmitScreen_load method loads the signature panel to the SubmitScreen form on load.

5.2.8.2.8 cmbColorList_SelectedIndexChanged
The cmbColorList_SelectedIndexChanged recognizes when the cmbColorList object is changed and then changes the color of the pen.

5.2.9 SyncScreen Class
This section satisfies the requirements in section 2.5 in the Requirements Document.
5.2.9.1 Fields:

5.2.9.1.1 b_exit
The user presses this button to exit the application.

5.2.9.1.2 b_sync
The user presses this button to sync their data to the CNHP server. This button will call the Process Sync function.

5.2.9.1.3 l_username
This is a label with the predefined value "Username".

5.2.9.1.4 l_password
This is a label with the predefined value "Password".

5.2.9.1.5 t_username
This is a text box that will accept the student’s username.

5.2.9.1.6 t_password
This is a textbox that will accept the user’s password.

5.2.9.2 Methods:

5.2.9.2.1 b_exit_Click
This is the method called when the exit button is clicked. This will exit the application.

5.2.9.2.2 b_sync_Click
This is the method called when the sync button is clicked. This will initiate the ProcessSync function.
5.2.9.2.3 **ParseXML**
This is the method will be used to reparse the category list after it has been updated.

5.2.9.2.4 **FileEncrypt**
This is the method will be to encrypt the new skillsheet when it is received from the CHNP server.

5.2.9.2.5 **FileDecrypt**
This is the method will be to decrypt the new skillsheet when it is received from the CHNP server.

5.2.9.2.6 **Process Sync**
This function will connect to the CNHP server and transfer the Request.xml and all associated signatures to it. It will then accepts a response, decrypt it and check for errors. After errors are displayed, this function will then send a request for the current user’s skillsheet, encrypt it and save it as SkillSheet.xml.

5.2.10 **CreateUserScreen**
This section satisfies the requirements in section 2.6 in the Requirements Document.

---

**CreateUser**

<table>
<thead>
<tr>
<th>Class</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fields</strong></td>
<td></td>
</tr>
<tr>
<td>AppPath : string</td>
<td></td>
</tr>
<tr>
<td>b_create_user : Button</td>
<td></td>
</tr>
<tr>
<td>b_exit : Button</td>
<td></td>
</tr>
<tr>
<td>l_confirm : Label</td>
<td></td>
</tr>
<tr>
<td>l_first_name : Label</td>
<td></td>
</tr>
<tr>
<td>l_password : Label</td>
<td></td>
</tr>
<tr>
<td>l_student_id : Label</td>
<td></td>
</tr>
<tr>
<td>t_confirm : TextBox</td>
<td></td>
</tr>
<tr>
<td>t_name : TextBox</td>
<td></td>
</tr>
<tr>
<td>t_password : TextBox</td>
<td></td>
</tr>
<tr>
<td>t_student_id : TextBox</td>
<td></td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
</tr>
<tr>
<td>b_create_user_Click(object sender, EventArgs) : void</td>
<td></td>
</tr>
<tr>
<td>b_exit_Click(object sender, EventArgs) : void</td>
<td></td>
</tr>
<tr>
<td>FileDecrypt(string sInputFilename, string sOutputFilename) : void</td>
<td></td>
</tr>
<tr>
<td>FileEncrypt(string sInputFilename, string sOutputFilename) : void</td>
<td></td>
</tr>
<tr>
<td>ParseXml(string App) : ArrayList</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 5.2.10 CreateUserScreen*

5.2.10.1 **Fields:**

5.2.10.1.1 **b_create_user**
This button will call the CreateUser function when pressed.

5.2.10.1.2 **b_exit**
This button will exit the application.
5.2.10.1.3 1_confirm
This label is predefined with the text "Confirm Password".

5.2.10.1.4 1_first_name
This label is predefined with the text "First Name".

5.2.10.1.5 1_password
This label is predefined with the text "Password".

5.2.10.1.6 1_student_id
This label is predefined with the text "Student Id".

5.2.10.1.3 t_confirm
This is a text box that accepts the password of the user. This will be compared against the text entered in the t_password text box.

5.2.10.1.4 t_name
This is a text box that accepts the first name of the user.

5.2.10.1.5 t_password
This is a text box that accepts the password of the user.

5.2.10.1.6 t_student_id
This is a text box that accepts the student id of the user.

5.2.10.2 Methods:

5.2.10.2.1 b_createUser_Click
This button will initiate the CreateUser function when pressed.

5.2.10.2.2 b_exit_Click
This will initiate when the button is pressed, and then exit the application.

5.2.10.2.3 FileEncrypt
This is the method will be to encrypt the new skillsheet when it is received from the CHNP server.

5.2.10.2.4 FileDecrypt
This is the method will be to decrypt the new skillsheet when it is received from the CHNP server.

5.2.10.2.5 ParseXML
This is the method will be used to re-parse the category list after it has been updated.

5.2.10.2.6 ProcessCreateUser
When this function is called it will initiate a Request the CNHP server. I will send the UserData.xml file and listen for a response. If the username and password entered are correct, the sever will respond with the users SkillSheet.xml. If the transaction failed this function will display an Error. If the transaction does not fail, the SkillSheet.xml will be encrypted.

5.2.11 Request Class
5.2.11.1 Fields:

5.2.11.1.1 category
This string containing the name of the category.

5.2.11.1.2 filename
This string containing the name of the file that contained the Instructor signature.

5.2.11.1.3 instructor
This string containing the name of the instructor.

5.2.11.1.4 skill
This string containing the name of the skill.

5.2.11.1.5 student
This string containing the username of the student.

5.2.11.1.6 timestamp
This string containing the timestamp of when the request was created.

5.2.11.2 Methods:

5.2.11.2.1 getCategory
Returns the category field.

5.2.11.2.2 getFilename
Returns the filename field.
5.2.11.2.3 getInstructor
Returns the instructor field.

5.2.11.2.4 getSkill
Returns the skill field.

5.2.11.2.5 getStudent
Returns the student field.

5.2.11.2.6 getTimestamp
Sets the timestamp field.

5.2.11.2.7 setCategory
Sets the category field.

5.2.11.2.8 setFilename
Sets the filename field.

5.2.11.2.9 setInstructor
Sets the instructor field.

5.2.11.2.10 setSkill
Sets the skill field.

5.2.11.2.11 setStudent
Sets the student field.

5.2.11.2.12 setTimestamp
Sets the timestamp field.

5.2.11.2.13 WriteRequest
Writes the Request Object to a file in XML format.

5.2.12 RequestList Class

![RequestList Class Diagram]

Figure 5.2.12 RequestList Class
5.2.12.1 *Fields:*

5.2.12.1.1 list
This field is an ArrayList containing a list of Request Objects.

5.2.12.2 *Methods:*

5.2.12.2.1 addRequest
This method will take a request as input and add it to the list field.

5.2.12.2.2 FileEncrypt
This is the method will be to encrypt the Request.xml file.

5.2.12.2.3 FileDecrypt
This is the method will be to decrypt the new Request.xml file.

5.2.12.2.4 getList
This method will return the list field.

5.2.12.2.5 getRequest
This method will take an integer n as input and return the nth Request in the list field.

5.2.12.2.6 getSize
This method will return the current size of the list field.

5.2.12.2.7 parseList
This method will take in a file name and then parse the file and populate the list field with all the requests contained in that file. This file will be decrypted using the Triple DES encryption.

5.2.12.2.8 writeList
This method will write the RequestList Object to Request.xml in proper xml format. It will then encrypt the file using Triple DES encryption.

5.3 *PDA Diagrams*

5.3.1 *UML Class Diagram*
Figure 5.3.1 PDA Design Diagram
5.3.2 PDA Story Board Diagram

Figure 5.3.2 PDA Storyboard
6. Communication Protocol

Hypertext Transfer Protocol (HTTP) version 1.1 will be used to communicate between the PDA and server components. The PDA is the source of all transactions. All the information passed between the PDA and the server, except for the signature images, is sent in XML format for ease of generation and parsing. The messages are encrypted using the 3DES algorithm (7.1).

6.1 Messages

6.1.1 Message Format

Messages are sent across the Internet from the PDA to the server using HTTP POST. The messages consist of two parts: an HTTP header and a message body.

6.1.1.1 Message Header

The header contains routing information and information about the message format. The first line specifies the HTTP request type (SITS always uses POST), the URL, and the HTTP version being used. The following fields must be included in the header:

- Host: The address of the SITS server.
- User-Agent: A string identifying the Syncing Application, including a version number (6.2).
- Content-Type: The type of message. This is either text/xml for plain text messages or multipart/mixed if images are being sent.
- Content-Length: The length of the message body in bytes.

The following is an example HTTP header:

```
POST /sync.php HTTP/1.1
Host: sits.cnhp.drexel.edu
User-Agent: SITS-PDA/1.0
Content-Type: text/xml
Content-Length: 780
```

6.1.1.2 Message Body

The message body is an XML file that can also be accompanied by signature image files. While each message contains unique information, the outermost tag of any message is always a SITSMessage tag. It has the following attributes:

- type: The type of message being sent (SkillRequest or SkillSheet).
- status: The status of the message (Request, Success, or Error).
- version: The version (6.2) of the communications protocol being used.

Example message:

```
<SITSMessage type="SkillRequests" status="Request" version="1.0">
```
6.1.3 Server Response

Whenever the server receives a message, it sends a response back to the Syncing Application. The response is either a SitsMessage XML file or a skill sheet depending on the message type.

The return message is sent in the response to the HTTP POST message as follows:

```
HTTP/1.1 200 OK
Content-Type: text/xml
```

6.1.2 SkillRequest

The Syncing Application sends skill requests as a multipart HTTP POST message. The first part of the message is an XML file containing a list of all the skill requests made on the PDA that have not yet been uploaded to the server. The rest of the message is a series of binary image files in the JPEG format, each of which corresponds to a skill request.

The multipart POST message has the following header:

```
POST /sync.php HTTP/1.1
Host: sits.cnhp.drexel.edu
User-Agent: SITS-PDA/1.0
Content-Length: 523780
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: text/xml;

... [XML message] ...
--AaB03x
Content-Disposition: file; filename="signature1.gif"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

... [signature1.gif] ...
--AaB03x
Content-Disposition: file; filename="signature2.gif"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

... [signature2.gif] ...
--AaB03x--
```

6.1.2.1 Format

The SitsMessage contains a series of Request tags. Each request tag has the following sub-tags:

- **Student**: The user ID of the student.
- **Instructor**: The user ID of the instructor.
• Skill: The name of the skill the instructor has signed off on.
• Filename: The name of the file that the signature for this skill was sent as.
• Timestamp: The date and time the instructor signed for the skill, with the following format:
  YYYY-MM-DDThh:mm:ssTZD (eg 1997-07-16T19:20:30+01:00)
where:
  YYYY = four-digit year
  MM = two-digit month (01=January, etc.)
  DD = two-digit day of month (01 through 31)
  hh = two digits of hour (00 through 23)
  mm = two digits of minute (00 through 59)
  ss = two digits of second (00 through 59)
  TZD = time zone designator (Z or +hh:mm or -hh:mm)

The following is an example SkillRequest message:

```xml
<SITSMessage type="SkillRequest" status="Request" version="1.0">
  <Request>
    <Student>dao23</Student>
    <Instructor>jpopyack</Instructor>
    <Skill>Wound Irrigation</Skill>
    <Filename>signature12.jpg</Filename>
    <Timestamp>2007-02-23T14:12:36+05:00</Timestamp>
  </Request>
</SkillSheet>
```

6.1.2.2 Response

When the server receives the XML file, each skill request is parsed and added to the database. However, there may be bad data in the request. Therefore, the response message is identical to the message sent to the server, but each Request tag contains a status attribute. The status attribute is set to one of the following:

• Success: The request was successfully added to the database.
• StudentError: The Student tag did not contain a student in the SITS database.
• InstructorError: The Instructor tag did not contain an instructor in the SITS database.
• UserError: Both the the Student and Instructor were not found in the SITS database.
• SkillError: Both users belong to the system, but the Skill tag did not contain a skill in the SITS database.
• SignatureError: All request information was accepted, but the signature file for the request could not be found.

An error can also occur if there is a signature file that is not used by any skill requests. In this case, a BadSignature tag is added to the response message. This tag has a Filename sub-tag specifying the name of the orphan signature file.

If any of the above errors occur, the status of the SITSMessage tag is set to "Error"; otherwise it is set to "Success". The following is an example SkillRequest response:

```xml
<SITSMessage type="SkillRequest" status="Error" version="1.0">
  <Request status="SkillError">
```

6.1.3 SkillSheet

The PDA requests an updated copy of the user's skill sheet using a SkillSheet message.

6.1.3.1 Format

The message contains a User tag with the PDA owner's user id and MD5 password hash. The following is an example SkillSheet message:

```xml
<SITSMessage type="SkillSheet" status="Request" version="1.0">
  <User>
    <ID>dao23</ID>
    <password>d41d8cd98f00b204e9800998ecf8427e</password>
    <Name>Dave</Name>
  </User>
</SITSMessage>
```

6.1.3.2 Response

If the user id does not belong to a student in the database or the password does not match, there is an error. The server responds with a message identical to the request message, except the status attribute is set to "Error". The following is an example error message:

```xml
<SITSMessage type="SkillSheet" status="Error" version="1.0">
  <User>
    <ID>dao23</ID>
    <password>d41d8cd98f00b204e9800998ecf8427e</password>
  </User>
</SITSMessage>
```

If the user id and password are correct, the response message is not a SITSMessage. Instead, it is the student's skill sheet. The skill sheet is an XML file with the format described by the following context-free grammar. All text within quotes is literal. All other text represents non-terminals, except TEXT, which can be any alphanumeric string. Start is the start symbol.
6.2 Version

The communication protocol has a version number. The version of the initial release will be 1.0. This version number can be changed if the protocol used by SITS ever changes for maintenance purposes. This allows each component to recognize what protocol is being used and gracefully handle cases where one module has been updated (i.e. the server) and another has not (i.e. a student's PDA).

6.3 Swimlane Diagram

The following diagram shows how the PDA and Server interact in the synchronization process. The diagram focuses on the interaction between the Synchronization Application and the Web Server; therefore, it does not detail the error handling processes. For more information, refer to the section for each message type. The
message types are indicated on each transition from the PDA to the server.
7. Encryption

In order to prevent tampering with sensitive information, all data files on the PDA and messages sent between the PDA and server are encrypted.

7.1 3DES String Encryption

The 3DES encryption algorithm is used to encrypt sensitive data. 3DES supports string encryption, so it is useful for storing and transmitting the text-based XML files and messages used in the SITS system.

7.2 Encryption on the PDA

7.2.1 3DES.NET
3DES Encryption is used to encrypt information on the PDA using the built in .NET System.Cryptography Library.

7.2.2 Encryption Method

This method of encryption is initiated by creating an instance of the 3DES service provider. The method of doing this is explained below:

Create an instance of the provider (here we will name it tdes):

```csharp
TripleDESCryptoServiceProvider tdes = (TripleDESCryptoServiceProvider) TripleDESCryptoServiceProvider.Create();
```

Set the encryption Key & IV:

```csharp
tdes.Key = ASCIIEncoding.ASCII.GetBytes("THIS_IS_SAMPLE_TEST_KEY_"); 
tdes.IV = ASCIIEncoding.ASCII.GetBytes("SAMPLEIV");
```

Set the encryption mode:

```csharp
tdes.Mode = CipherMode.CBC;
```

Create the decryptor or encryptor:

```csharp
DECRYPTOR: ICryptoTransform tdesdecrypt = tdes.CreateDecryptor();
ENCRYPTOR: ICryptoTransform tdesencrypt = tdes.CreateEncryptor();
```

Manipulate the Stream:

```csharp
CryptoStream cipherstream = new CryptoStream(Stream_to_Decrypt, tdesdecrypt, CryptoStreamMode.Read);
```

These Streams (ie. cipherstream) can then be written to files, displayed in message boxes, posted to the web, and converted to strings.

7.3 Encryption in PHP

7.3.1 mcrypt Encryption Functions

The MCRYPT_3DES encryption scheme will be used in conjunction with the following mcrypt functions where $key is the common public key and $input is the text to be encrypted/decrypted:

```php
mcrypt_ecb (MCRYPT_3DES, $key, $input, MCRYPT_ENCRYPT);
mcrypt_ecb (MCRYPT_3DES, $key, $input, MCRYPT_DECRYPT);
```

More information on the mcrypt library and PHP can be found at http://us2.php.net/mcrypt.