LabAssist
Design Document

Adam Cantrell

West Virginia University Institute of Technology

Benjamin Culkin

West Virginia University Institute of Technology

Version: 1

1. Introduction

1.1. Document Overview

This document describes the design of the components that LabAssist uses:

- A web interface used for interacting with the database and performing the outlined functions.
- A database used for storing and querying information.
- A background job used for sending mail notifications.

1.2. References

2. Software Architecture Overview

LabAssist is built from three main components: A web interface for interacting with that database, a database that stores information and prepares statistics from it, and a background job that sends mail notifications.

The only component that the users need know exist is the web interface. It provides users the ability to access all of the provided features of the software, and it does this by interacting with the database; both sending data to update the database, and running queries to retrieve data.

The database is the central component of the system, by virtue of being the place where all of the data that the system needs is stored and processed.

The background job is a task that is set to run every twenty to thirty minutes. Its job is to retrieve any pending notifications from the database, merge together any messages that can be merged, and then send out the merged notifications via email.

The system is designed to run off of any system running the following:

- PHP 7
- Web Server
- Postgres 9.6

3. Software Design Description

3.1. Login

This major component handles logging users into the system via one of two authentication methods. First time users will be prompted to complete a registration form handled by the registration subcomponent when logging in from either authentication method.

3.1.1. Component 1: Login

This component exists solely to allow users gain access to the LabAssist system. This page is the default page on the server, as such all users start with this component. The login component has two methods. Method 1 is labeled Kiosk login and requires only an Employee ID or Student ID as input. Method 1 only grants access to the clock-in/clock-out functionality. Method 2 is labeled User Login requiring both a username and password. Method 2 only grants access to the Portal component of the system.

3.1.1.1. Component Interfaces

Logic

attemptPassLogin()

Checks username and password combination by binding the LDAP server with the supplied credentials.

attemptKioskLogin()

Queries to see if Employee/Student ID is registered. Internally calls kioskLoginCheck() and directs user to either registration or to the clock in/out system.

isRegistered()

Queries database to determine whether a database record exists for a user attempting to login.

kioskLoginCheck()

Queries the database to determine if a record exists for the given ID.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts dynamic code into static HTML code prints.

printEndBody()

Prints static HTML code for closing the page.

3.1.1.2. Component Design Description

The component is implemented as an Action-Domain-Responder model where login.php would be the action, login-logic.php the domain, and login-page.php the responder. The register action first checks to ensure that the user is currently logged into the system via a session variable and redirects all users to the clock-in/clock-out component if the kiosk flag is set or to the portal component if the userLogin flag is set.

When using method 1, the domain queries the database to determine if the user has an account already registered. If not, the system redirects to the register component. If the user is already registered, users are directed to the clock-in/clock-out component.

When using method 2, the domain queries the LDAP server with the supplied username and password pair via a bind call. If the bind fails, the responder displays an appropriate message. If the bind succeeds, the domain then queries the database to determine if a tuple exists with the appropriate user information. If no tuple is found then the domain redirects to the register component, otherwise the domain redirects to the portal component.

3.1.2. Component 2: Register

This component exists solely to register users with the LabAssist system. Users are directed to this component when a valid login attempt is made with the login component, and no record is found in the database that matches.

3.1.2.1. Component Interfaces

Logic

restoreForm()

Restore previous values if form reloads.

generateRegistration()

Generates the HTML registration form with values provided by the LDAP controller.

attemptRegistration()

Attempts to register the user. Performs input validation and internally calls insertRegistration.

insertRegistration()

Queries the database attempting to add user information to the database.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printForm(html, error)

Takes the dynamic HTML registration and generated errors, and inserts the dynamic code into a static template and prints it

printEndBody()

Prints static HTML code for closing the page.

3.1.2.2. Component Design Description

The component is implemented as an Action-Domain-Responder model where user_registration.php would be the action, user_registration-logic.php the domain, and user_registration-page.php the responder. The register action first checks to ensure that the user has only come from the login system via a session variable and redirects all users to the login screen if the variable is empty or not set. Next, the domain queries the LDAP server for basic user information such as email, and name by reverse looking up the information they provided at the login form submission. The responder then prints the pre-filled form to the screen. On form submission the action page informs the domain to attempt to register the user. On success, the user will proceed on to the next component in the sequence. On failure the user may update the input and try again or cancel. Cancelling returns the user to the login screen. At no point is a password ever stored in the local database before, during, and after registration.

3.1.3. Workflow

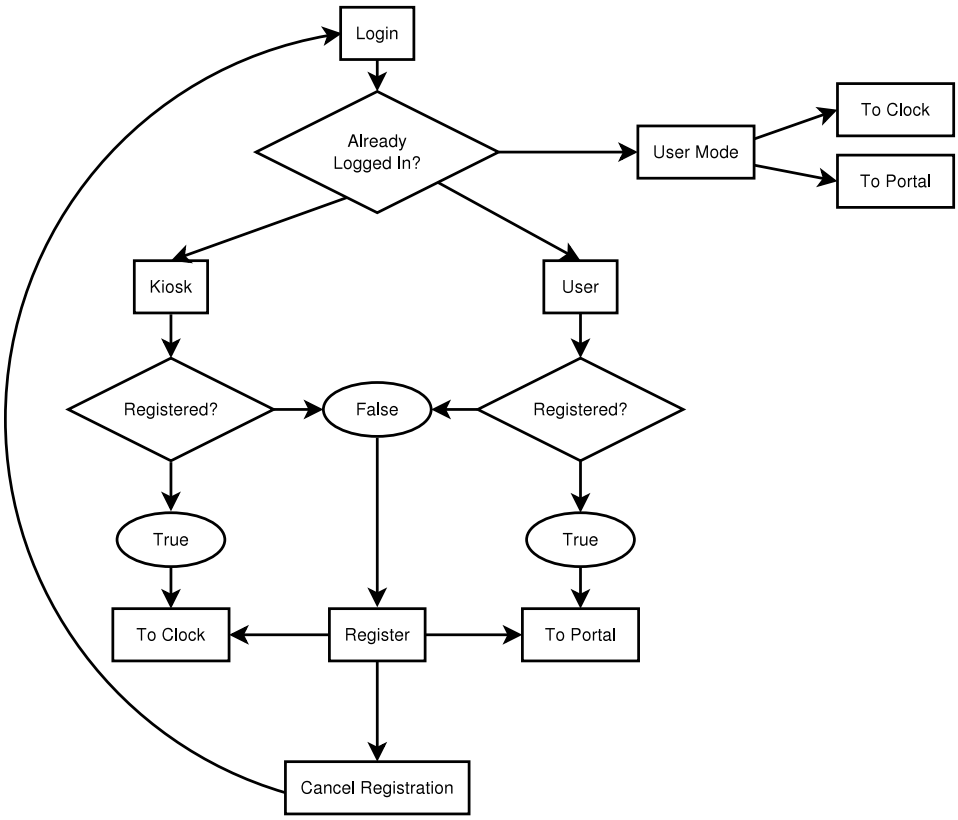

Fig: Login Sequence Diagram

3.2. Clock-in/Clock-out

This major component handles logging the time in/time out as well as the section for students arriving to and from a lab.

3.2.1. Component Clock In/Out

Students arrive here from a successful kiosk mode login, and are given the option to clock-in if not already clocked in, or the ability to clock out if already clocked in. This component is only accessible from the kiosk mode login.

3.2.1.1. Component Interface

Logic

generateClockinClockoutForm()

Calls an internal function to generate the form. Renders either clock in or clock out form based on results from isClockin().

isClockin()

Queries database to check for outstanding user clock-ins. Returns a tri-valued int (true/false/error).

genClockoutForm()

Generates the form for a clock out.

genLab(department)

Creates the department listing for the clock-in form. Takes the previous values as a parameter for restoration if needed.

genClass(department, class)

Creates the class listing for the clock-in form based on a department. Takes the previous class as a parameter for restoration if needed.

attemptClockinClockout()

Master function that calls attemptClockin() or attemptClockout() internally depending on the result of isClockin().

attemptClockin()

Queries the database attempting to insert a record of a clock in.

attemptClockout()

Queries the database attempting to update the appropriate outstanding clock-in with a clock-out.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printForm(html, error)

Takes the dynamic HTML registration and generated errors, and inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.2.1.2. Component Design Description

Users arrive at this component after a successful kiosk login. The component is implemented as an Action-Domain-Responder model where tutor_session.php would be the action, tutor_session-logic.php the domain, and tutor_session-page.php the responder. The action module makes an initial check to ensure the user arrived from a successful kiosk login and then queries the domain to render the form. The domain checks the database to determine if the user has an outstanding clock in, if so, then the clock out form is generated and passed to the responder. If the user does not have an outstanding login, the login page is generated and passed to the responder. The generated page contains two dropdowns, one for department and one for the class. After the user fills out the form and clicks submit, the action module calls attempt ClickinClockout() to insert or modify the database. If cancel is chosen, the action module redirects the user to the login form.

3.2.2. Workflow

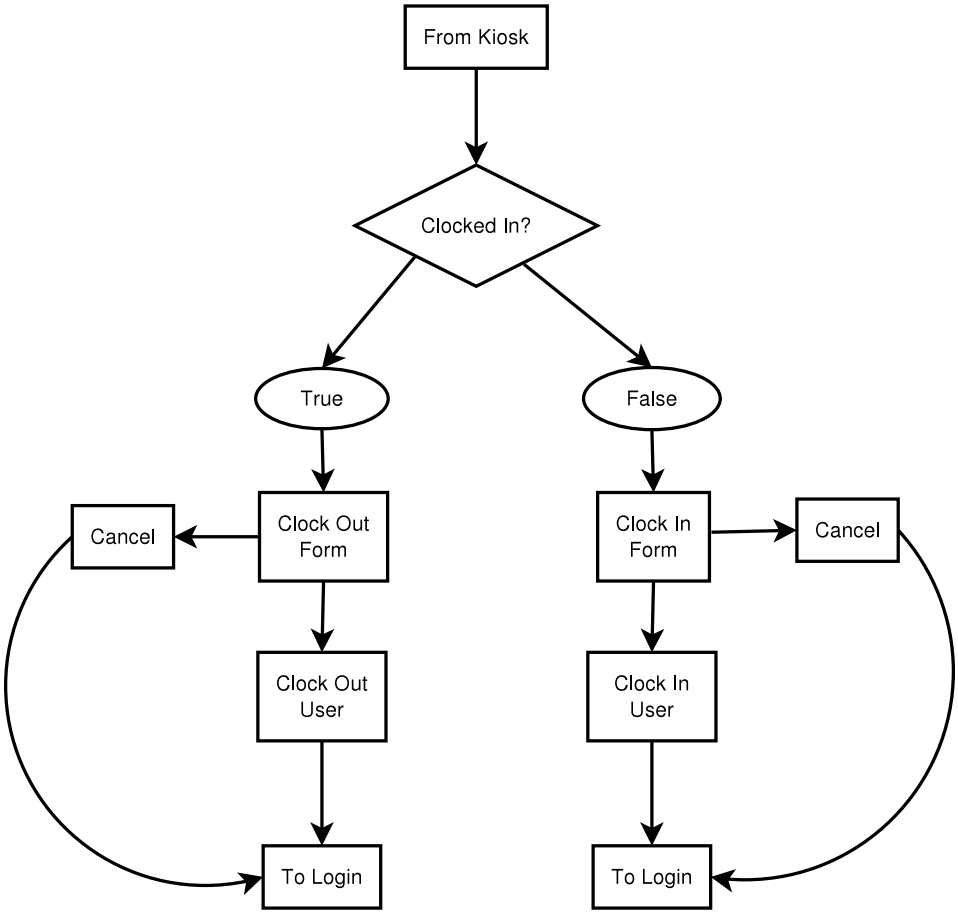

Fig: Clock In/Out Sequence Diagram

3.3. Portal

This major component contains the core functionality of this project. Namely, this is the ability to access the Question and Answer system, as well as reporting, scheduling, and management functionality. Access to this major component is limited to users who have successfully logged into user mode.

3.3.1. Component 1: Home

Displays important information and updates to the user, such as a notification of responses to questions.

3.3.1.1. Component Interface

Logic

generateBasicNotifications()

Generates basic notifications, such as new replies.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.1.2. Component Design Description

This page serves as the default landing page after a user logs into the system from the user login. It is implemented as an Action-Domain-Responder model where portal-home.php would be the action, portal-logic, the domain, and portal-page as the responder. The action page performs a check to make sure the user is logged in and redirects them to the login page if they are not. The domain queries the database for any notifications and generates a templated response that gets passed to the responder. The responder then displays the output to the user. No input is taken from the user on this page.

3.3.2. Tutor Schedules

This component displays the current lab schedules.

3.3.2.1. Component Interface

Logic

generateForm()

Calls internal functions to generate the form.

genDepartment()

Generates a dropdown for the user to choose the desired lab section.

querySchedule()

Retrieves the schedule from the database for display.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.2.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system. The action page checks that a user has successfully logged in and redirects them to the login page if they are not. The domain queries the database for a list of departments and generates a dropdown populated with the departments for the responder. The responder then displays the form to the user. Once a selection has been made, the action page notifies the domain to create a schedule listing for the specified department. The result is passed to the responder for displaying.

3.3.3. Q/A System

This component allows users to submit questions, and allows tutors to respond to a student's questions.

3.3.3.1. Component Interface

Logic

generateForm()

Calls internal functions to generate the form.

createQuestion()

Generates a form for the creation of a new question.

generateListing()

Generates a listing of all threads created by the user.

generateThread()

Generates a specified thread.

queryThreads(user)

Queries listings of threads by a user.

queryThread(id)

Requests a thread, by ID, from the database.

queryCreateQuestion()

Inserts the user question into the database.

queryCreateResponse(id)

Inserts a response to a question into the database.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.3.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system. The action page checks that a user has successfully logged in and redirects them to the login page if they are not. The domain queries the database for a list of threads for the responder. The responder then displays the form to the user. Once a selection has been made, the action page notifies the domain to generate either the new question form, or display the thread. The result is passed to the responder for displaying. If a new question is submitted, or a new response is generated the system will then attempt to update the database with the new records. The thread that was updated/created will subsequently be generated by the domain and passed to the responder for viewing.

3.3.4. User Account Settings

This page allows the user to view their user information and update any modifiable fields.

3.3.4.1.

Logic

generateForm()

Calls internal functions to generate the form.

updateUserInfo()

Attempts to update the user information in the database.

getUserInfo()

Retrieves the user information from the database.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.4.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system. The action page checks that a user has successfully logged in andredirects them to the login page if they are not. The domain queries the database for the current user's information before passing to the responder. The responder then displays the form to the user. Once a change has been made, the action page notifies the domain to update the database accordingly. Upon successful submission or cancelation, the user is redirected back to the home.

3.3.5. Tutor Scheduling

Not to be confused with Tutor Schedules, this component modifies and generates the schedules displayed by the aforementioned component.

3.3.5.1. Component Interface

Logic

generateForm()

Calls internal functions to generate the form.

genDepartment()

Generates a dropdown for the user to choose the desired lab section.

querySchedule()

Retrieves the schedule from the database.

genCreate()

Generates the form for creating a new entry.

updateEntry()

Generates the form for updating an entry.

attemptCreate()

Attempts to insert a new entry into the database.

attemptUpdate()

Attempts to update a database entry.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.5.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system and a user with the role of tutor or greater. The action page checks that a user has successfully logged in andredirects them to the login page if they are not. The domain queries the database for the current department schedule before passing to the responder. The responder then displays the form to the user. Once a change has been made, the action page notifies the domain to update the database accordingly. Upon successful submission or cancelation, the user is redirected back to the edit schedule page.

3.3.6. Manage Users

3.3.6.1. Component Interface

Logic

displayAll()
Calls internal functions to generate the base form with a listing of users.

genEditForm()
Generates a 'edit user' form to be displayed. Calls formattedUserInformation internally.

formattedUserInformation()
Fills the edit form with the user information.

generateTable()
Formats list of users into a table listing of users.

getAccessLevelSelect(currentUserRole)
Creates a dropdown menu of roles that the current user is allowed to assign.

getDepartmentList(currentDepartment)
Creates a dropdownm menu of available departments to assign the user to.

filterRoleList(array)
Processes an arrary of available roles and returns the roles the current user is available to assign.

getDetailedUserInfo(userid)
Pulls information of the user being edited.

getUserList()
Queries database for a list of users.

Display

printHeader()
Prints static HTML code for the page start.

printStartBody()
Prints static HTML code for the opening body tags.

printPortalHead()
Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)
Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)
Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()
Prints static HTML code for closing the page.

3.3.6.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system and a user with the role of staff or greater. The action page checks that a user has successfully logged in andredirects them to the login page if they are not. The domain queries the database for the current listing of users before passing to the responder. The responder then displays the form to the user. Once a the edit user button is clicked, the action page notifies the domain to generate the edit user form for the responder to display. Once an edit has been submitted the domain updates the database accordingly. Upon successful submission or cancelation, the user is redirected back to the edit user page.

3.3.7. Manage Classes

3.3.7.1. Component Interface

Logic

displayAll

Calls internal functions to generate the base form with a listing of classes.

genEditForm()

Generates an 'edit class' form to be displayed. Calls formattedClassInformation internally.

formattedClassInformation()

Fills the edit form with class information.

generateTable()

Formats a list of class into a table that lists those classes.

getDepartmentList(currentDepartment)

Creates a dropdown of available departments to assign the class to.

getDetailedClassInfo(classid)

Pulls information of the class being edited.

getClassList()

Queries database for a list of classes.

genCreate()

Generates the page to create a new class.

attemptCreate()

Attempts to insert a new entry into the database.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.7.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system and a user with the role of staff or greater. The action page checks that a user has successfully logged in andredirects them to the login page if they are not. The domain queries the database for the current listing of classes before passing to the responder. The responder then displays the form to the user. If the edit class button is clicked, the action page notifies the domain to generate the edit class form for the responder to display. Once an edit has been submitted the domain updates the database accordingly. Upon successful submission or cancelation, the class is redirected back to the edit class page. If the add class button is clicked, the domain

generates the create class form and passes it to the responder to display to the user. On a submit, the domain attempts to insert the record into the database. If successful or the user canceles sumbission at any point, the user is redirected to the manage classes page.

3.3.8. Manage Class Sections

3.3.8.1. Component Interface

Logic

displayAll()

Calls internal functions to generate the base form with a listing of sections.

genEditForm()

Generates an 'edit section' form to be displayed. Calls formattedSectionInformation internally.

formattedSectionInformation()

Fills the edit form with the section information.

generateTable()

Formats list of sections into a tabular listing of sections.

getDepartmentList(currentDepartment)

Creates a dropdown of available departments to assign the section to.

getDetailedSectionInfo(sectionid)

Pulls the information of the section being edited.

getSectionList()

Queries database for the list of sections.

genCreate()

Generates the page to create a new section.

attemptCreate()

Attempts to insert a new section into the database.

Display

printHeader()

Prints static HTML code for the page start.

printStartBody()

Prints static HTML code for the opening body tags.

printPortalHead()

Prints the visible header bar for the portal.

printSideBar(userinfo, navigation)

Prints the sidebar containing basic user information, as well as a navigation bar.

printForm(html, error)

Takes the dynamic HTML registration form and any generated errors. Inserts the dynamic code into a static template and prints it.

printEndBody()

Prints static HTML code for closing the page.

3.3.8.2. Component Design Description

This page is accessable from any location by a valid user. A valid user for this page is defined as a user who has successfully logged in via the portal system and a user with the role of admin or greater. The action page checks that a user has successfully logged in andredirects them to the login page if they are not. The domain queries the database for the current listing of sections before passing to the

responder. The responder then displays the form to the user. If the edit section button is clicked, the action page notifies the domain to generate the edit section form for the responder to display. Once an edit has been submitted the domain updates the database accordingly. Upon successful submission or cancelation, the section is redirected back to the edit section page. If the add section button is clicked, the domain generates the create section form and passes it to the responder to display to the user. On a submit, the domain attempts to insert the record into the database. If successful or the user canceles sumbission at any point, the user is redirected to the manage sections page.

3.4. Database

The database serves as the place where all of the data that LabAssist uses is stored and analyzed.

3.4.1. Component Interfaces

This component interfaces with both the web interface and the mailer. It gets information from the web interface, and then sends information out to both the web interface and the mailer.

3.4.2. Component Design Description

The database is stored as a collection of SQL tables split into the following categories:

Users
: This group of tables is concerned with storing users and what roles they have.

Class Usage
: This group of tables is concerned with storing data about classes and when people are using the lab.

Question & Answer
: This group of tables is concerned with storing the data necessary for the question and answer system to work.

Scheduling
: This group of tables is concerned with storing the data necessary for the tutor scheduling feature to work.

3.4.3. Workflows and Algorithms

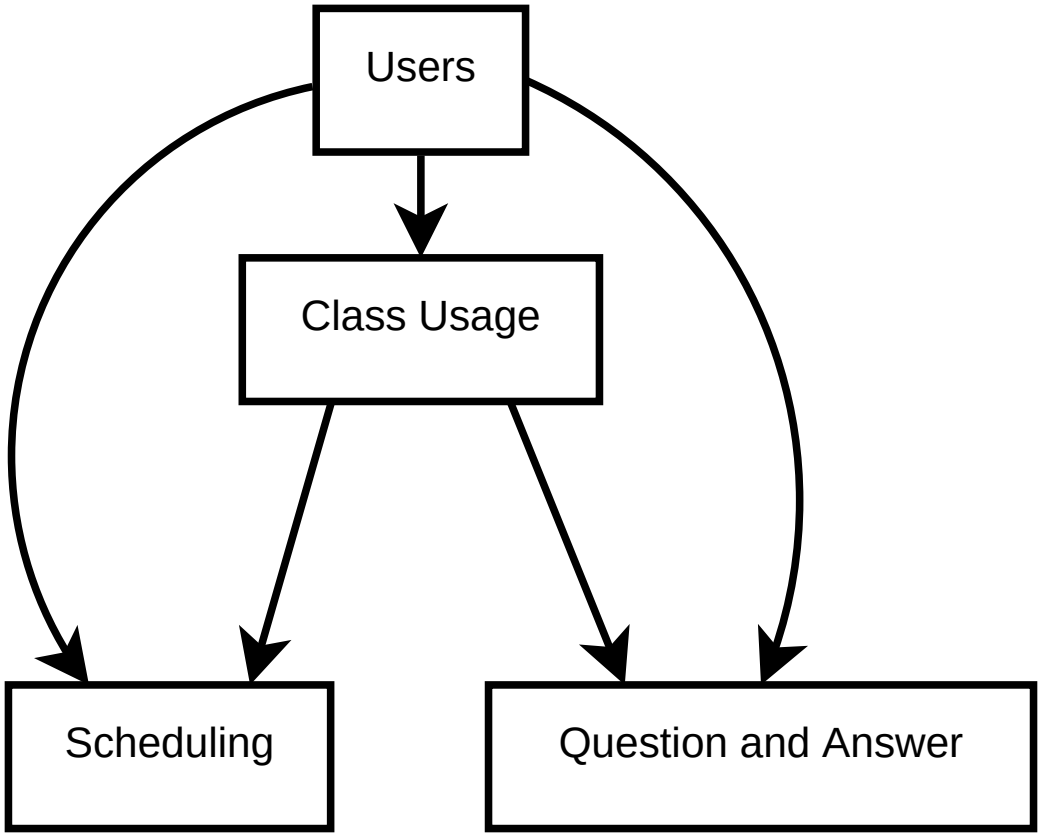

Fig. 2: Table Category Dependancies

For a more detailed description of the database, see the attached database schema.

3.5. Component III: Background Mailer

The job of the background mailer is, as previously noted, to connect to the database, retrieve messages, merge them and then send out the merged messages.

3.5.1. Component Interfaces

This component interfaces with both the database and an external SMTP server, using the database to retrieve messages to send, and the SMTP server to send the messages to users they are supposed to go tgo.

3.5.2. Component Design Description

The mailer is designed as a java application split into three main parts: Grabber, Batcher and Sender. The roles of the components are as follows

Grabber
: The grabber component functions to retrieve messages from the database and convert them into Message objects.

Batcher
: The batcher component functions to take a bunch of Message objects, and then combine compatible messages to cut down on the total number of mail messages sent.

Sender
: The sender takes Message objects and converts them into SMTP messages to send them.

3.5.3. Workflows and Algorithms

To allow for easier editing of message types and to simplify the addition of more message types, the mailer uses a text-based template for each message type with placeholders that will be filled in by the body variables from the message.

To cut down on the amount of notifications sent, the mailer uses a merging algorithm consisting of performing the following steps on every message:

1. Get the stored messsage of the correct type for the recipients of the current message, or create one if there is no stored message for the recipients of the current message.
2. Merge the body variables of the current message into the stored message. The way this happens is specific to the type of the message, but in general it is done by appending the two variables together, possibly with a seperator.

3.5.4. Class Descriptions

The mailer is composed of three main ’operating’ classes, and two data classes. The operating classes are Mailer, MessageBatcher, and MessageGrabber, using Message and MessageType for data.

Table of Contents