



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

University of Wisconsin–Madison | Department of Computer Sciences

Human-Computer Interaction Laboratory



## MILESTONE 4

Trey Cahill   Katie Greenwald   Samad Jawaid   Kevin Ridsen

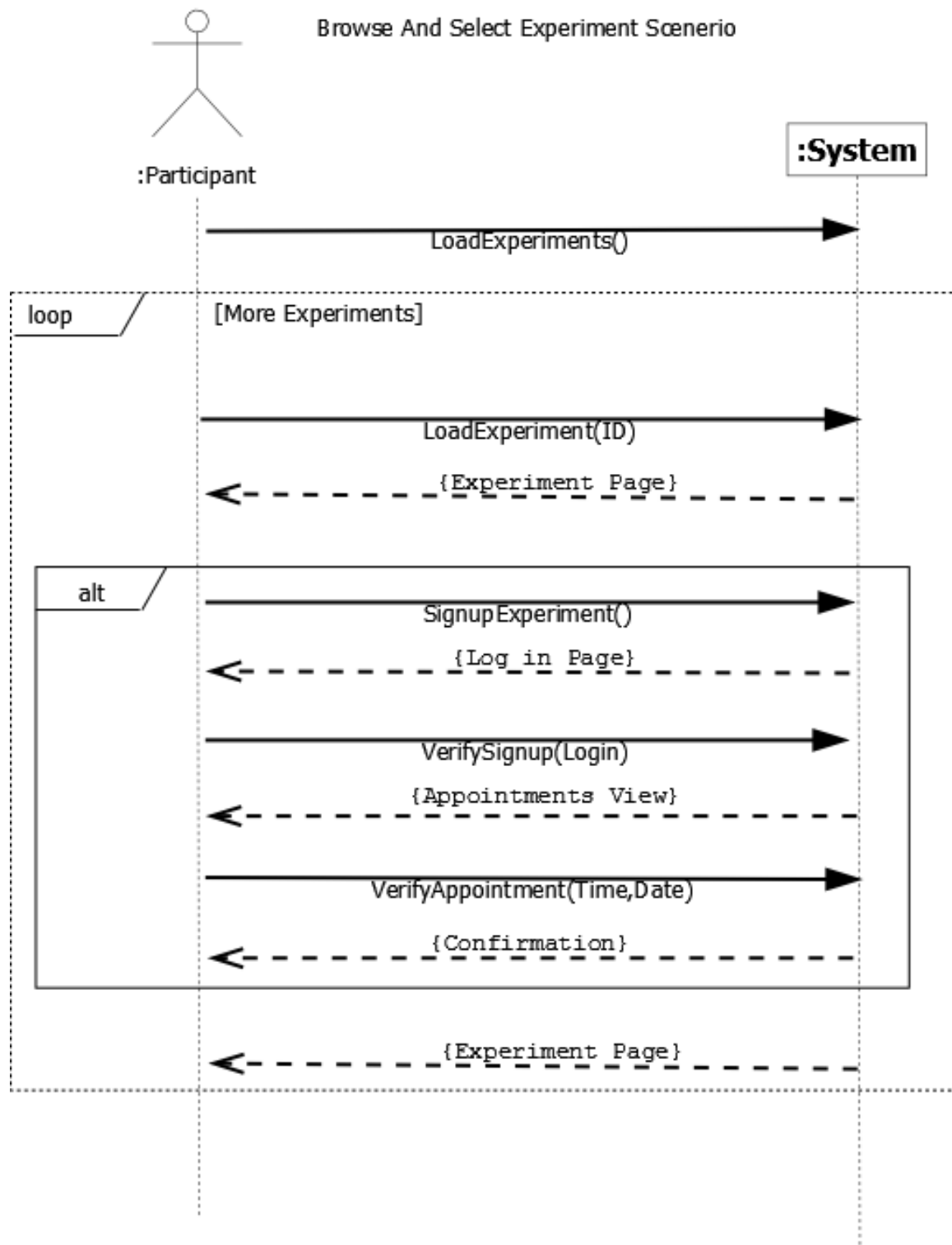
January 27, 2012

# Contents

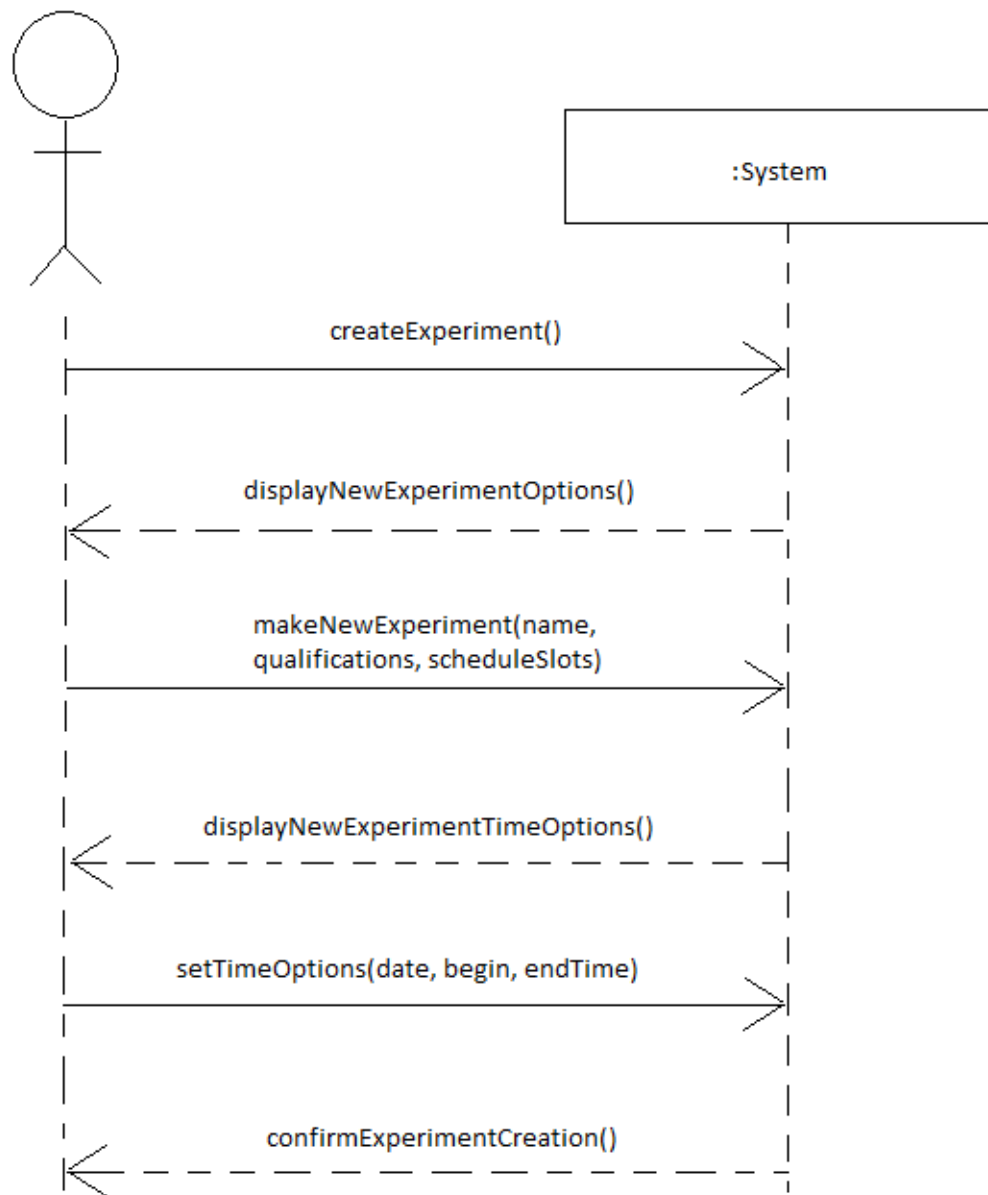
<b>1</b>	<b>System Sequence Diagrams</b>	<b>2</b>
1.1	Browse and Select Experiment . . . . .	2
1.2	Create Experiment . . . . .	3
1.3	Modify Experiment . . . . .	4
1.4	Create Account . . . . .	5
1.5	Login . . . . .	6
<b>2</b>	<b>Operations Contracts</b>	<b>6</b>
2.1	CreateExperiment . . . . .	6
2.2	ModifyExperiment . . . . .	7
2.3	LoadExperiments . . . . .	7
2.4	LoadExperiment . . . . .	7
2.5	Sign Up Experiment . . . . .	7
2.6	Verify Sign up . . . . .	7
2.7	Verify Appointment . . . . .	7
2.8	Create Account . . . . .	8
2.9	Login . . . . .	8
<b>3</b>	<b>Interaction Diagrams</b>	<b>8</b>
3.1	Sign Up For Experiment . . . . .	8
3.2	Create Account . . . . .	8
3.3	Login . . . . .	8
<b>4</b>	<b>Package Diagram</b>	<b>9</b>
<b>5</b>	<b>Class Diagram</b>	<b>10</b>
<b>6</b>	<b>GRASP Principles</b>	<b>11</b>
6.1	Creator . . . . .	11
6.2	Info Expert . . . . .	11
6.3	Controller . . . . .	11
6.4	High Cohesion . . . . .	11
6.5	Low Coupling . . . . .	11
6.6	Pure Fabrication . . . . .	11
6.7	Indirection . . . . .	12
6.8	Polymorphism . . . . .	12
6.9	Protected Variation . . . . .	12
<b>7</b>	<b>References</b>	<b>12</b>
<b>8</b>	<b>Appendix</b>	<b>12</b>

# 1 System Sequence Diagrams

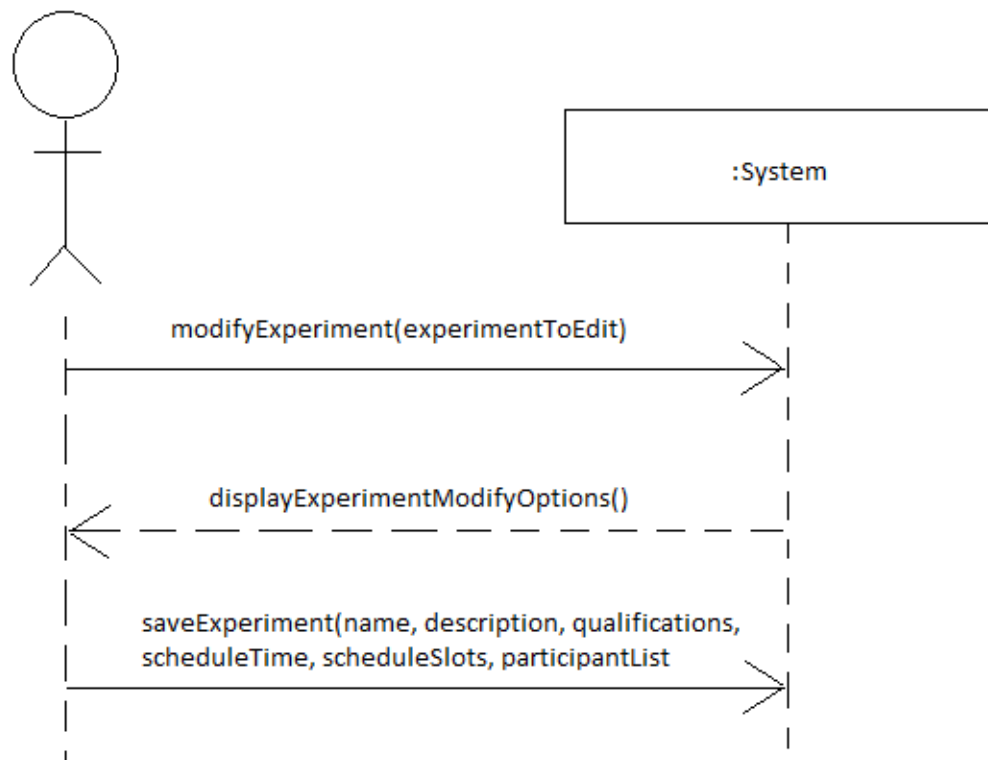
## 1.1 Browse and Select Experiment



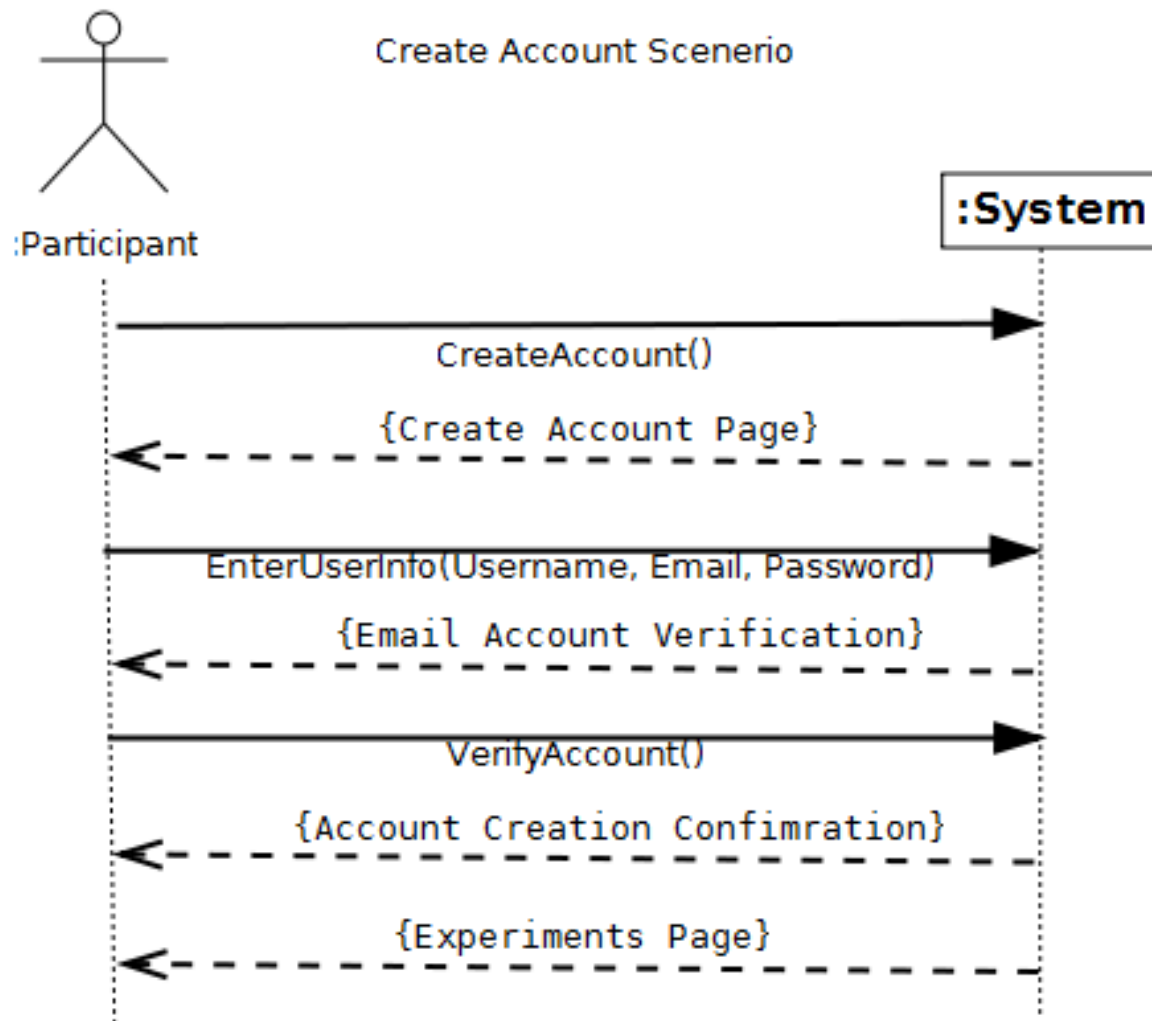
## 1.2 Create Experiment



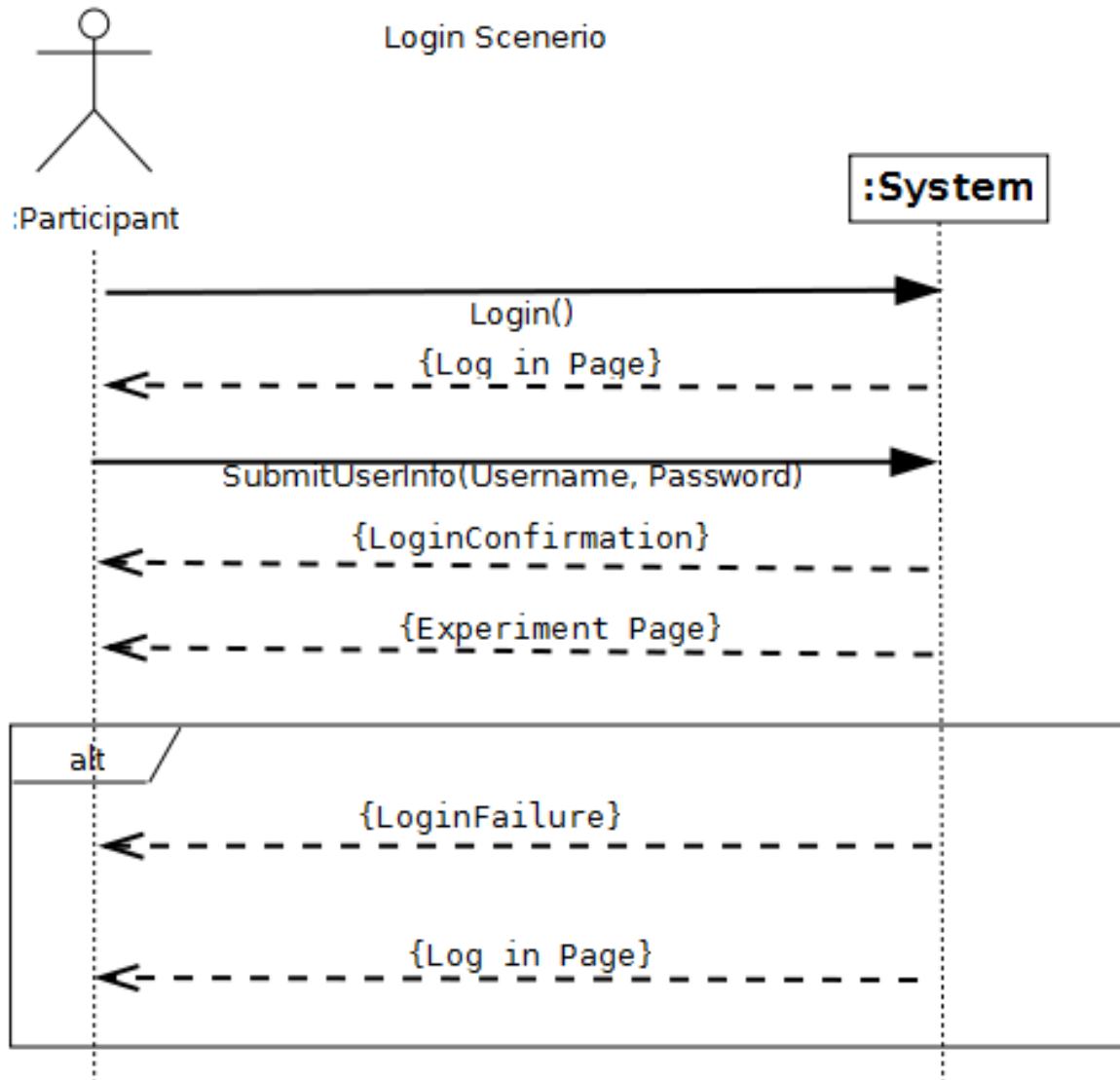
### 1.3 Modify Experiment



## 1.4 Create Account



## 1.5 Login



## 2 Operations Contracts

This section provides operation contracts for vital operations.

### 2.1 CreateExperiment

Operation:	CreateExperiment()
Cross References:	Uses Cases: Add Experiment
Preconditions	User is an Administrator and/or a Researcher and has authenticated
Postconditions:	Experiment object will have been created, or an error message will have been displayed

## 2.2 ModifyExperiment

Operation:	ModifyExperiment(ID)
Cross References:	Uses Cases: Modify Experiment
Preconditions	User is an Administrator and/or a Researcher and has authenticated
Postconditions:	The experiment object will have its fields modified, will have been deleted, or an error message will have been displayed

## 2.3 LoadExperiments

Operation:	LoadExperiments()
Cross References:	Uses Cases: Select Experiment
Preconditions	The participant has loaded the web page.
Postconditions:	Experiments collection was created (instance creation)

## 2.4 LoadExperiment

Operation:	LoadExperiment(ID)
Cross References:	Uses Cases: Select Experiment
Preconditions	The participant as clicked on an experiment.
Postconditions:	Experiment was created (instance creation)
	Experiment attributes was loaded into the web page

## 2.5 Sign Up Experiment

Operation:	SignupExperiment()
Cross References:	Uses Cases: Sign up for Experiment
Preconditions	The participant has clicked on an experiment to sign up for
Postconditions:	LogIn was created (instance creation)
	LogIn.logged became loggedIn (attribute modification)

## 2.6 Verify Sign up

Operation:	VerifySignup(Login)
Cross References:	Uses Cases: Sign up for Experiment
Preconditions	The participant has logged into or created an account
Postconditions:	this.hasConflict is set to false (attribute modification)

## 2.7 Verify Appointment

Operation:	VerifyAppointment(Time,Date)
Cross References:	Uses Cases: Sign up for Experiment
Preconditions	The participant has selected a time and date slot.
Postconditions:	experiment.slots has been modified (attribute modification)
	Database is updated



## 2.8 Create Account

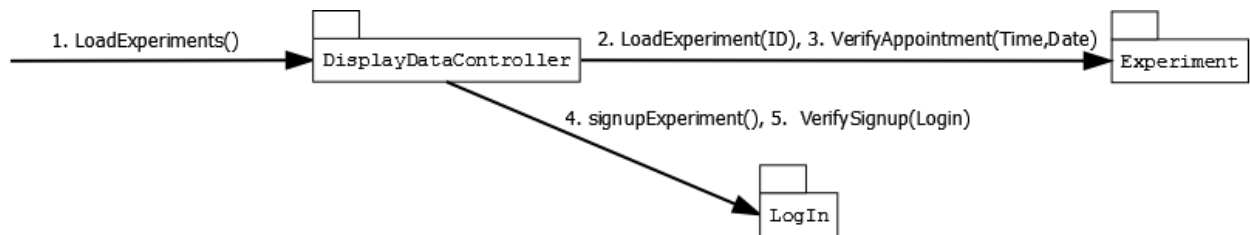
Operation:	CreateAccount()
Cross References:	Uses Cases: Create Account
Preconditions	None.
Postconditions:	The user has an account.
	A verification email has been sent to the specified email address.
	The user is logged in.

## 2.9 Login

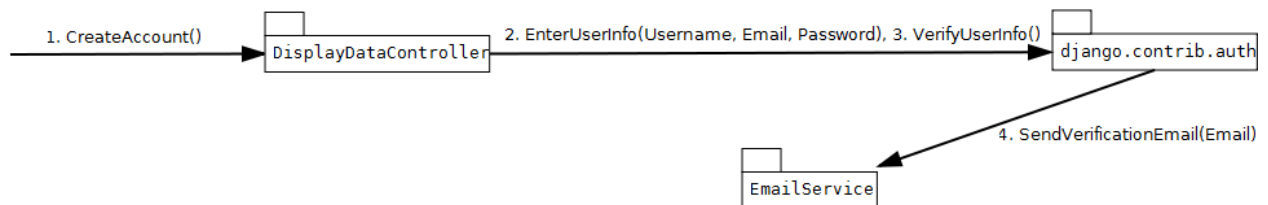
Operation:	Login()
Cross References:	Uses Cases: Login
Preconditions	The user has an account.
Postconditions:	The user is logged in.
	Appropriate rights have been given to the account logged in.

## 3 Interaction Diagrams

### 3.1 Sign Up For Experiment



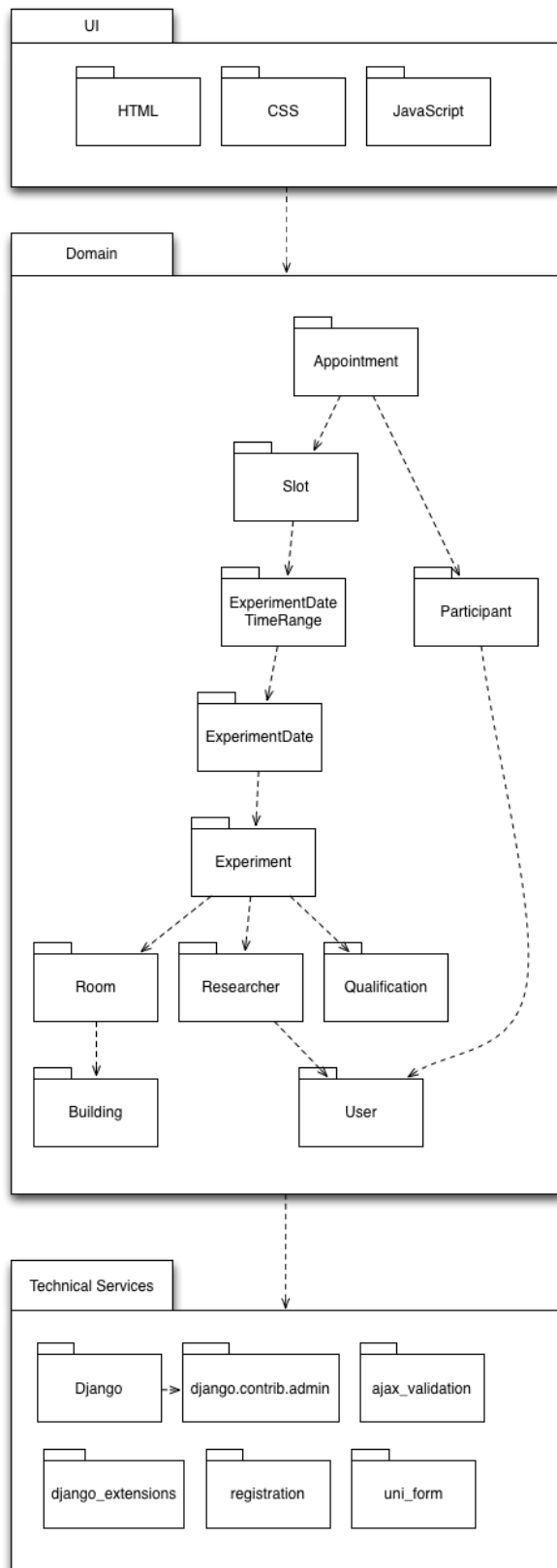
### 3.2 Create Account



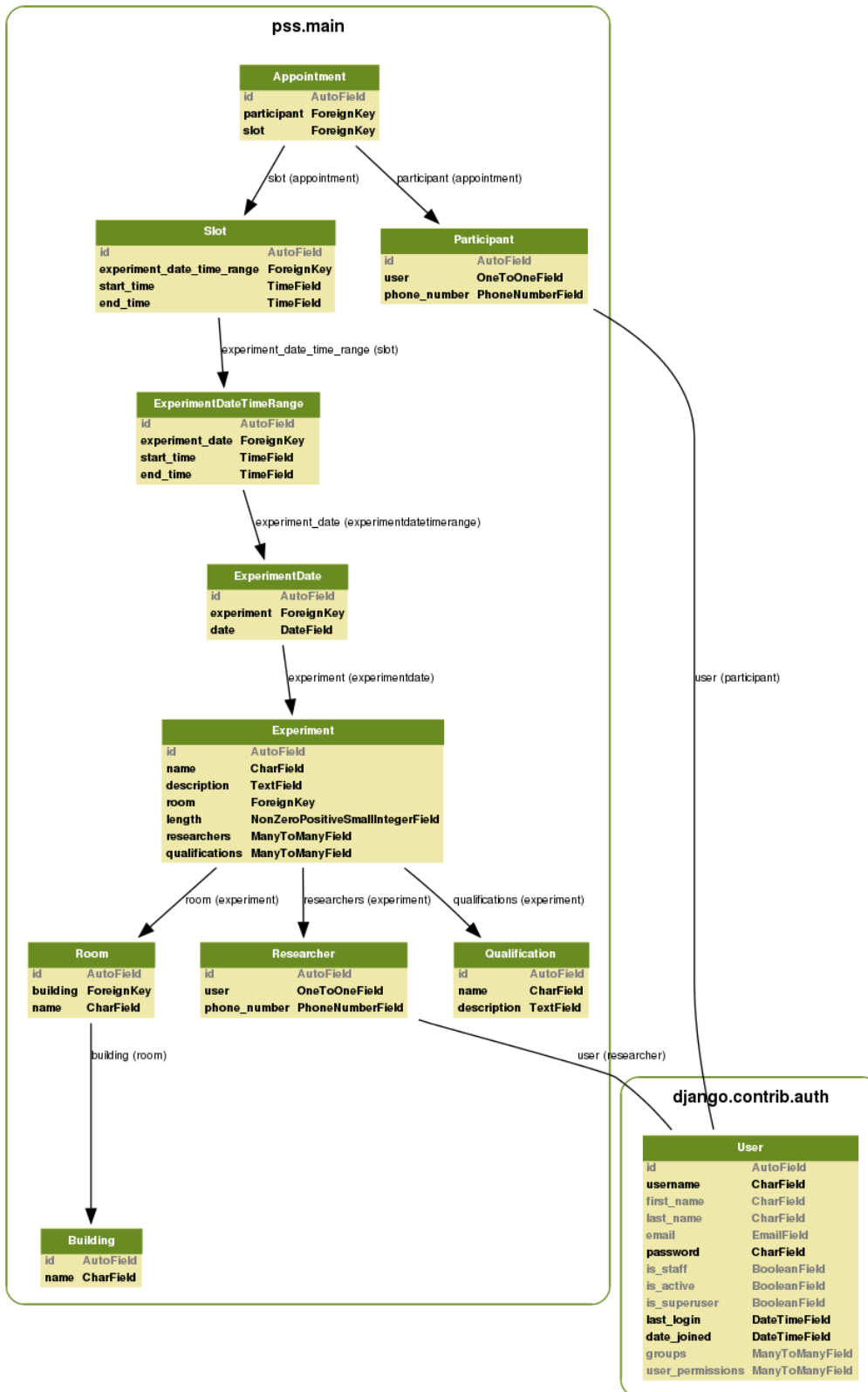
### 3.3 Login



## 4 Package Diagram



## 5 Class Diagram



## 6 GRASP Principles

### 6.1 Creator

In our system, there are many objects which must be created. One notable example of this is the creation of participation slots. A date-time range, the time period over which the experiment is being run, can create for itself several slots, or chunks within the time period, which participants can sign up for. Since these slots only pertain to one date-time range, the range is the one to create and manage them.

### 6.2 Info Expert

For information expert, an example from our system would be the Experiment class. An Experiment has knowledge of ExperimentDate, ExperimentDateTimeRange, Slots, Appointments, Participants, Room, Researchers, and everything else pertaining to an instance of an experiment. When something needs to be done by or to an experiment, that experiment will have all of the necessary information, so it can be relied on to complete those tasks.

### 6.3 Controller

One nice example of a controller in our system is the URLs. Each URL can be broken up so the system will know what should handle the user's input. For example, in the case of experiments/dates/edit/[an experiment id], the system would turn to the experiment editing code. Each URL our system uses works this way, meaning that all the power of what is displayed on each page is delegated exclusively by the URL handling.

### 6.4 High Cohesion

High cohesion is achieved in many parts of our code, but an example of where cohesion is extremely strong is in the Administration code. All of our Administration procedures, fields, responsibilities, and all similar objects are located in the administration package, specifically in the `django.contrib.admin` package as seen in the package diagram in section 4 of this document. By having administration as its own package, it leaves other administration tasks out of other unrelated classes in our system.

### 6.5 Low Coupling

By separating out objects and responsibilities into their own packages, we achieve a fairly low coupling. Each separate package contains a responsibility and is connected only to the classes to which it must be connected. Our class diagram in section 5, shows how our classes interact with each other and also shows that each class is used only by a class that needs its information. For example, Qualification is only connected to Experiment; it could possibly be connected to participant, but because you will not be able to completely verify the participants qualifications until they show up for the experiment, there is no need to make the connection between Participant and Qualifications.

### 6.6 Pure Fabrication

Pure Fabrication becomes useful in our system by creating ExperimentDateTimeRange and ExperimentDate. Neither of these two classes are in the domain model, but since they make the code easier to work with and separate out responsibility, increasing cohesion, they become very useful as classes. Should we not have

these two classes Slot or Experiment would have to contain this information, which would decrease cohesion and generally add to the complexity of the Slot or Experiment or both.

## 6.7 Indirection

The GRASP principle indirection directly relates to our system for how we need to represent experiment dates and the experiment datetime ranges. An experiment must keep track of the dates and time slots for each day that it is offered. We decided that having two intermediate classes, ExperimentDate and ExperimentDateTimeRange, would reduce coupling and ensure easier maintainability of the system. The ExperimentDate class keeps track of the slots that the ExperimentDateTimeRange can generate. This enables the user to enter a time range and the system will then calculate the specific time slots. The Experiment class just has to have the ExperimentDates. This makes the coupling of time slots to experiments cleaner. An alternative would be for the experiment to have a massive list of all the dates and slots that it is offered. This would make it difficult to add or remove slots later and not know which slots are current filled participants.

## 6.8 Polymorphism

Currently, the Participant Scheduling System requires both researchers and participants. In order to accomplish this, a User class was introduced to provide a standard base class and then the Researcher class was derived from this. This provides our solution with the polymorphism GRASP principle. The other option was to create two separate classes for researcher and user, but then there would be duplicated code. Furthermore, if there needs to be another type of user then it will be simpler to just extend the current User class.

## 6.9 Protected Variation

In our system, protected variation builds off our decision for polymorphism. The User class enables the protected variation GRASP principle since it protects us from changes in the type of users who need to use the system. If the client comes back with a request for another type of user besides participant and researcher, the system is setup to handle this by just extending the User class. This provides the most elegant solution to the problem since the other option would have been to create the different user classes separately and would make it difficult to extend later.

# 7 References

# 8 Appendix