Team Members: Gregory Aiello, Alex Curtin, David Mayer,

Douglas Naphas, Jeff Ramspacher

Date: 05/09/2014

Version #2

DESIGN DOCUMENT

Contents

Overview

This document describes the design and architecture of the Random Fruit software project management system.

Abstract

Random Fruit is an open-source project management site for students developing software.

Like existing tools, it allows developers to break work down into tickets, assign the tickets to each other, view and update a page capturing all the information about a ticket, search tickets, and use the information captured in tickets to generate reports. Random Fruit is tailored to the needs of student projects using earned-value reporting, adopting reporting tools likely to be useful in an academic setting and enabling coordination among teams by an instructor.

Random Fruit aims to help students manage their work while demonstrating their progress to an instructor. It tracks time budgets and time incurred for each ticket, as well as hours worked on the project overall. The reporting feature aggregates this information to produce earned-value charts showing the changes in the remaining work volume and highlighting the achievement of milestones. Students can use the system to visualize planned value, earned value, and actual value at any time.

In addition to acting as a superuser with team-member privileges over all the groups, the instructor can use the system to quickly view how the course as a whole is progressing towards its goals. The instructor can compare groups to spot potential problems before they become emergencies.

A dashboard landing page shows users a top-level view of their project's status. Markdown-enabled comments tell the story of a ticket on its view page. Users can filter and sort tickets, and save reports.

For distribution, Random Fruit's installer application places its file structure on the server of a course instructor or department at a university, where it serves a web page to student and instructor users. It is written in PHP, JavaScript, and HTML, and served by a MySQL database. It is accessed through the web.

Components and their Interfaces

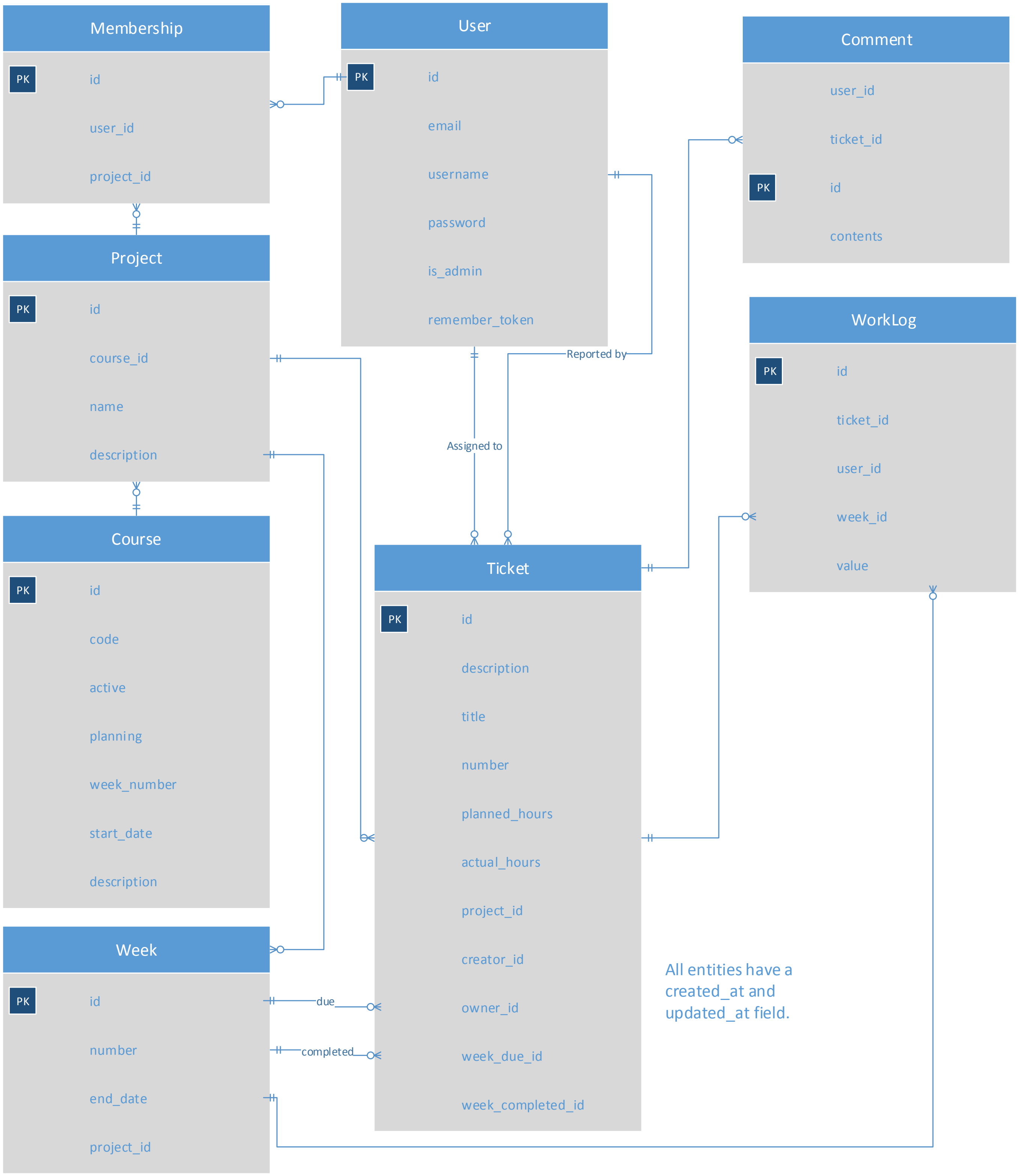
Random Fruit is a relational database accessed through a web interface. The database stores information about the entities that exist in a project management system. The web interface allows users to change these entities and view information about them.

These relationships are described in the following statements of Random Fruit's components and the services they provide to each other and to the user.

1. Component: Database
   a. Description: The database stores information about objects represented by the program that persist between sessions of the web interface. It provides services to the web interface.
   b. Services
      i. Establish a connection.
      ii. Execute a query.
      iii. Close a connection.
2. Component: Web interface
   a. Description: The web interface presents the information represented by the program to the user and allows the user to change this information. It provides services to the user.
   b. Services
      i. Create a course.
      ii. Modify a course.
      iii. Configure a group.
      iv. Create a ticket.
      v. Edit a ticket.
      vi. Generate reports.
      vii. Log in.
      viii. Search tasks.

Database

The core of our application is a relational database, illustrated in the following diagram.



All entities have a created_at and updated_at field.

Users include instructors (for whom user.is_admin is true) and students (for whom it is false). user.is_admin controls access privileges. Students can create and edit tickets, and log work against them, but cannot create courses, modify their own course, or do or see anything related to groups in which they are not members. Instructors have control over every aspect of every course in which they are members. The weak Membership entity tracks who is in which group.

There are multiple projects in each course. “Project” is synonymous with “group.” A project is in one course.

Comments are text added to tickets and displayed on the view-ticket page for the individual ticket to which they belong. All comments are stored, so that the comment thread can be shown on a ticket.

Tickets have planned_hours associated with them. This is the amount of time the task should take. This field can only be edited if the course is in planning mode, which is controlled by a Boolean property of the course table. Tickets also have a week due and a week completed. The week due is used to compute planned vale: planned value as of a date is the sum of planned hours for tickets on the project with a week due that ends on or before the reporting date. Week completed is used to compute earned value: earned value as of a date is the sum of planned hours for tickets on the project with a week completed that is not null and that ends on or before the reporting date.

Weeks are database objects. They have an ending date and a number. They are created based on their course’s first-week ending date (course.start_date) and their course’s number of weeks (course.week_number). These are set when a course is created. Weeks know the project to which they belong, but all projects in a course have identical sets of weeks.

Work logs are occasions of work being done on a ticket. If user U incurs H hours on task T in week W, a work log with work_log.user_id = U, work_log.value = H, work_log.ticket_id = T, and work_log.week_id = W is created. Work logs are used to compute actual value. Actual value as of a date is the sum of work_log.value entries logged against tickets in the project for weeks whose week ending date is on or before the reporting date.

Courses know whether they are in active mode via the Boolean courses.active field. Courses in active mode show up on the Overview/Dashboard page, which is the landing page after login and shows the graphs. This is done so that an instructor using the system over multiple semesters will not have a continuously growing set of old courses cluttering the display.

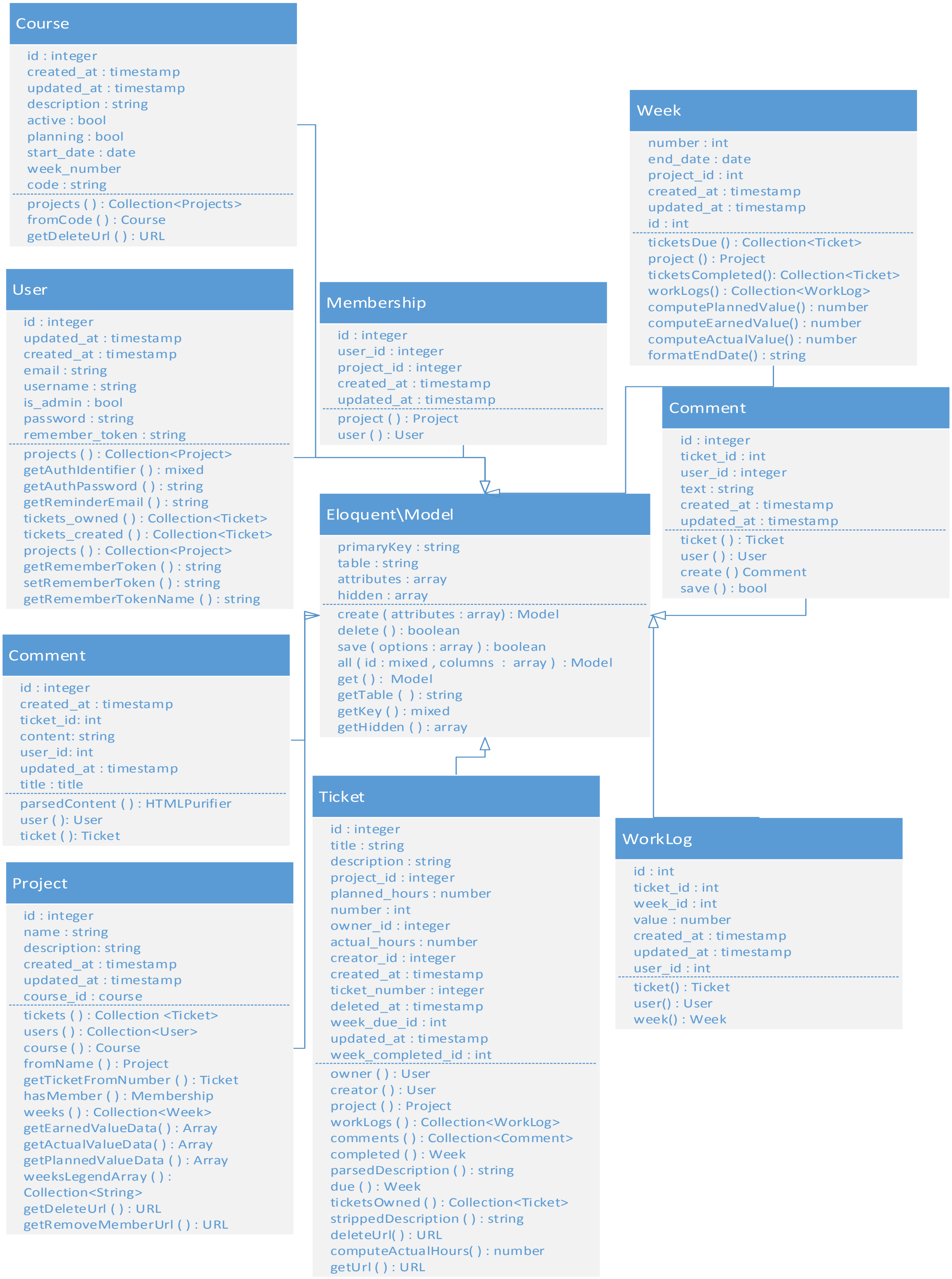
The database does not directly touch graphs or reports. It serves the web interface, which uses the data to create the views presented to the user.

Classes

The entities translate into classes. We use the Model-View-Controller pattern. Classes called models represent the entities in the system. Controller classes change the models and create views to present to the users.

Database entities match the models very closely. The model classes in the class diagram below relate to each other just as the entities in the database diagram do.

We are using the MVC-based framework Laravel. It provides base classes that our classes extend, and automatically creates model classes with access methods based on the definition of our database. Models are mutated by controllers.



Course

id : integer
created_at : timestamp
updated_at : timestamp
description : string
active : bool
planning : bool
start_date : date
week_number
code : string

projects ( ) : Collection<Projects>
fromCode ( ) : Course
getDeleteUrl ( ) : URL

User

id : integer
updated_at : timestamp
created_at : timestamp
email : string
username : string
is_admin : bool
password : string
remember_token : string

projects ( ) : Collection<Project>
getAuthIdentifier ( ) : mixed
getAuthPassword ( ) : string
getReminderEmail ( ) : string
tickets_owned ( ) : Collection<Ticket>
tickets_created ( ) : Collection<Ticket>
projects ( ) : Collection<Project>
getRememberToken ( ) : string
setRememberToken ( ) : string
getRememberTokenName ( ) : string

Comment

id : integer
created_at : timestamp
ticket_id: int
content: string
user_id: int
updated_at : timestamp
title : title

parsedContent ( ) : HTMLPurifier
user ( ): User
ticket ( ): Ticket

Project

id : integer
name : string
description: string
created_at : timestamp
updated_at : timestamp
course_id : course

tickets ( ) : Collection <Ticket>
users ( ) : Collection<User>
course ( ) : Course
fromName ( ) : Project
getTicketFromNumber ( ) : Ticket
hasMember ( ) : Membership
weeks ( ) : Collection<Week>
getEarnedValueData( ) : Array
getActualValueData( ) : Array
getPlannedValueData ( ) : Array
weeksLegendArray ( ) :
Collection<String>
getDeleteUrl ( ) : URL
getRemoveMemberUrl ( ) : URL

Membership

id : integer
user_id : integer
project_id : integer
created_at : timestamp
updated_at : timestamp

project ( ) : Project
user ( ) : User

Eloquent\Model

primaryKey : string
table : string
attributes : array
hidden : array

create (attributes : array) : Model
delete ( ) : boolean
save (options : array) : boolean
all (id : mixed , columns : array) : Model
get ( ) : Model
getTable ( ) : string
getKey ( ) : mixed
getHidden ( ) : array

Ticket

id : integer
title : string
description : string
project_id : integer
planned_hours : number
number : int
owner_id : integer
actual_hours : number
creator_id : integer
created_at : timestamp
ticket_number : integer
deleted_at : timestamp
week_due_id : int
updated_at : timestamp
week_completed_id : int

owner ( ) : User
creator ( ) : User
project ( ) : Project
workLogs ( ) : Collection<WorkLog>
comments ( ) : Collection<Comment>
completed ( ) : Week
parsedDescription ( ) : string
due ( ) : Week
ticketsOwned ( ) : Collection<Ticket>
strippedDescription ( ) : string
deleteUrl( ) : URL
computeActualHours( ) : number
getUrl ( ) : URL

Week

number : int
end_date : date
project_id : int
created_at : timestamp
updated_at : timestamp
id : int

ticketsDue () : Collection<Ticket>
project () : Project
ticketsCompleted(): Collection<Ticket>
workLogs() : Collection<WorkLog>
computePlannedValue() : number
computeEarnedValue() : number
computeActualValue() : number
formatEndDate() : string

Comment

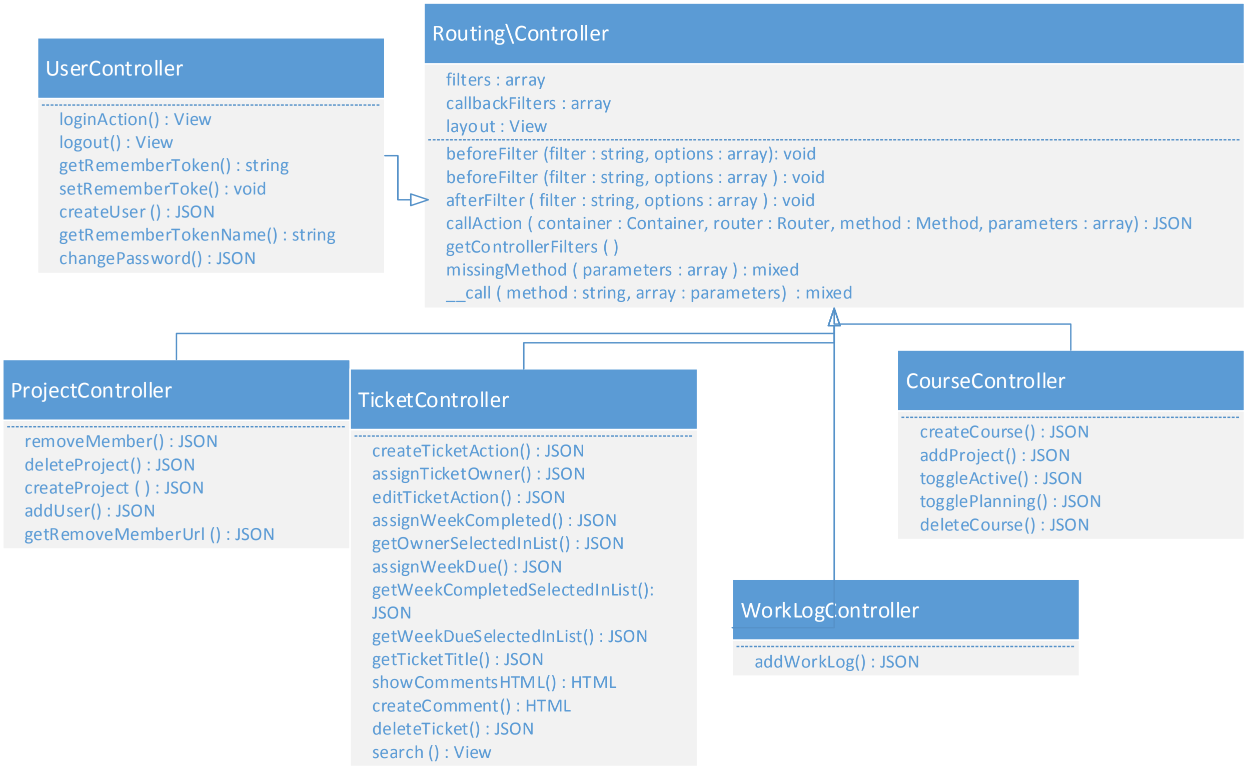
id : integer
ticket_id : int
user_id : integer
text : string
created_at : timestamp
updated_at : timestamp

ticket ( ) : Ticket
user ( ) : User
create ( ) Comment
save ( ) : bool

WorkLog

id : int
ticket_id : int
week_id : int
value : number
created_at : timestamp
updated_at : timestamp
user_id : int

ticket() : Ticket
user() : User
week() : Week

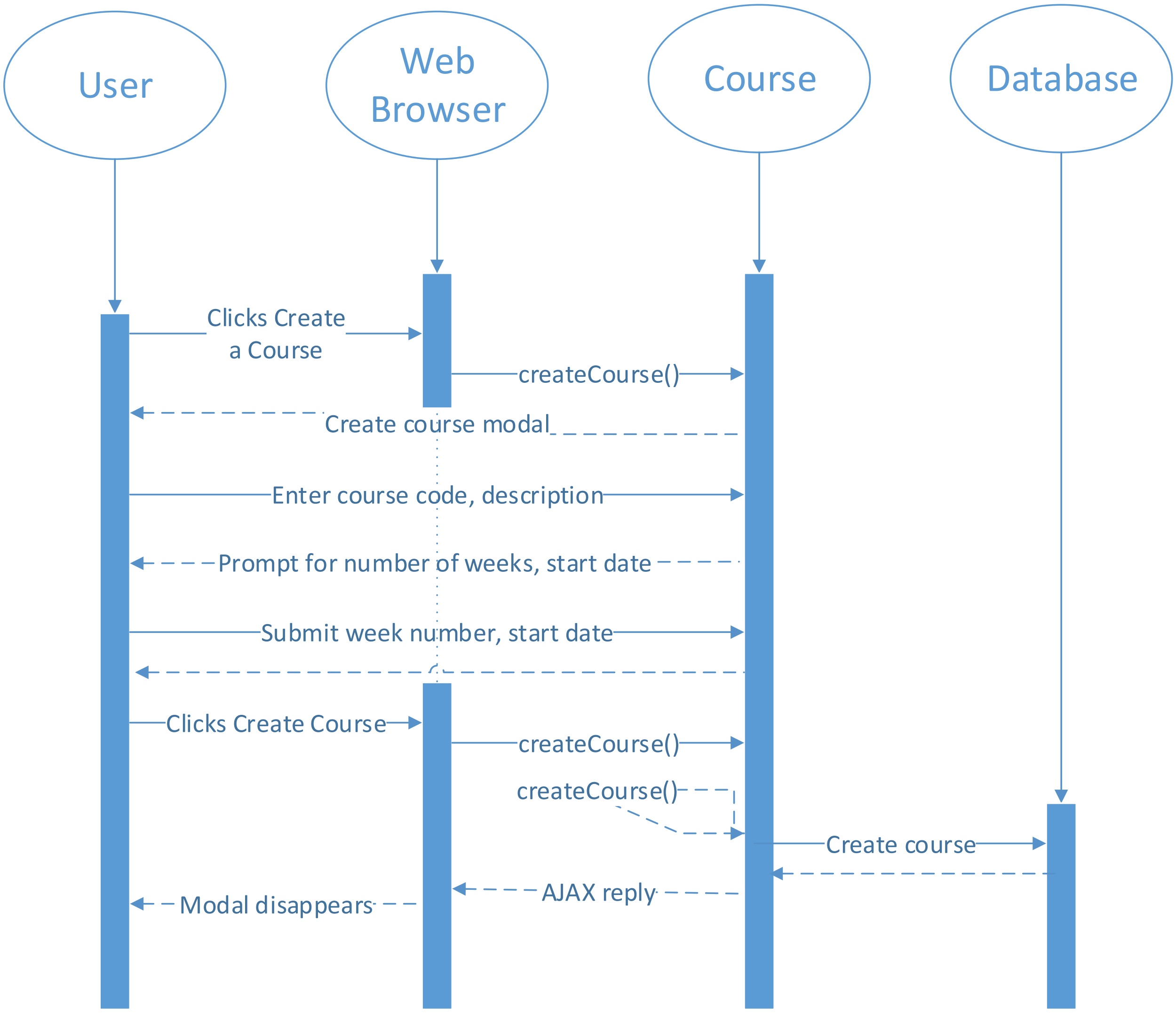

UserController
loginAction() : View
logout() : View
getRememberToken() : string
setRememberToke() : void
createUser () : JSON
getRememberTokenName() : string
changePassword() : JSON
Routing\Controller
filters : array
callbackFilters : array
layout : View
beforeFilter (filter : string, options : array): void
beforeFilter (filter : string, options : array) : void
afterFilter (filter : string, options : array) : void
callAction (container : Container, router : Router, method : Method, parameters : array) : JSON
getControllerFilters ( )
missingMethod (parameters : array) : mixed
__call (method : string, array : parameters) : mixed
ProjectController
removeMember() : JSON
deleteProject() : JSON
createProject ( ) : JSON
addUser() : JSON
getRemoveMemberUrl () : JSON
TicketController
createTicketAction() : JSON
assignTicketOwner() : JSON
editTicketAction() : JSON
assignWeekCompleted() : JSON
getOwnerSelectedInList() : JSON
assignWeekDue() : JSON
getWeekCompletedSelectedInList(): JSON
getWeekDueSelectedInList() : JSON
getTicketTitle() : JSON
showCommentsHTML() : HTML
createComment() : HTML
deleteTicket() : JSON
search () : View
CourseController
createCourse() : JSON
addProject() : JSON
toggleActive() : JSON
togglePlanning() : JSON
deleteCourse() : JSON
WorkLogController
addWorkLog() : JSON

More information on the classes and variables are in the Random Fruit API documentation. To view this, download our Git repository from git@github.com:douglasnaphas/RandomFruit.git, or from git@babyhuey.cis.temple.edu:RandomFruit/naphas, checkout the naphas-docs branch, then open RandomFruit/RandomFruit/app/models/docs/index.html and RandomFruit/RandomFruit/app/controllers/docs/index.html in a browser. Click on the class name hyperlinks there for more information about each class.

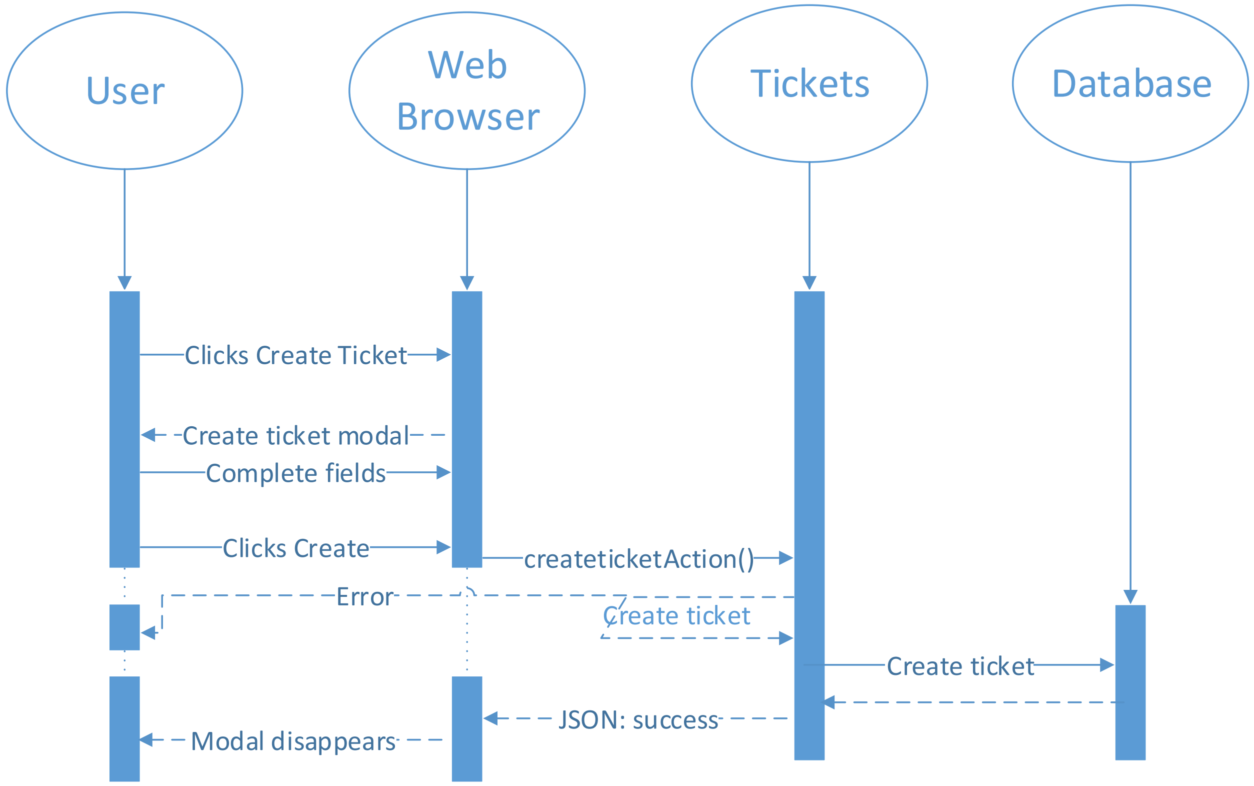
Sequence and State Diagrams

At the core of our requirements and testing procedures is a set of use cases. The following sequence diagrams, each labeled with the related use case, show the abstract passing of messages between components and entities as each use case is executed. The progression through states is shown in state diagrams for those use cases involving states.

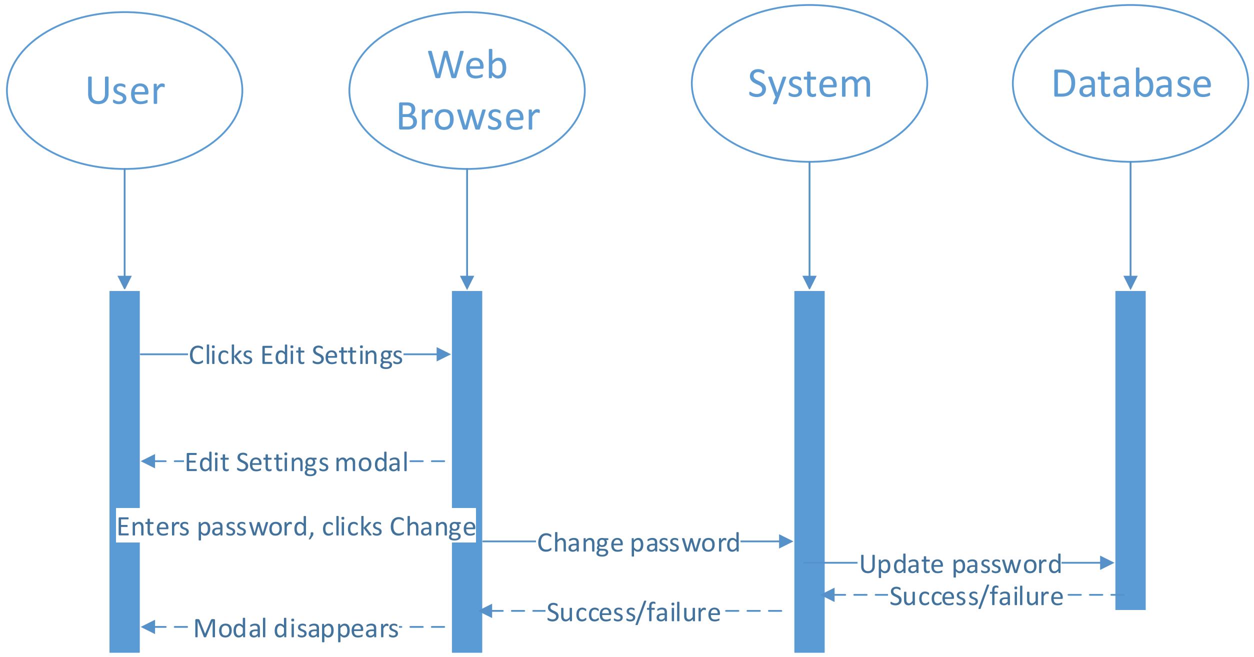
Create a course



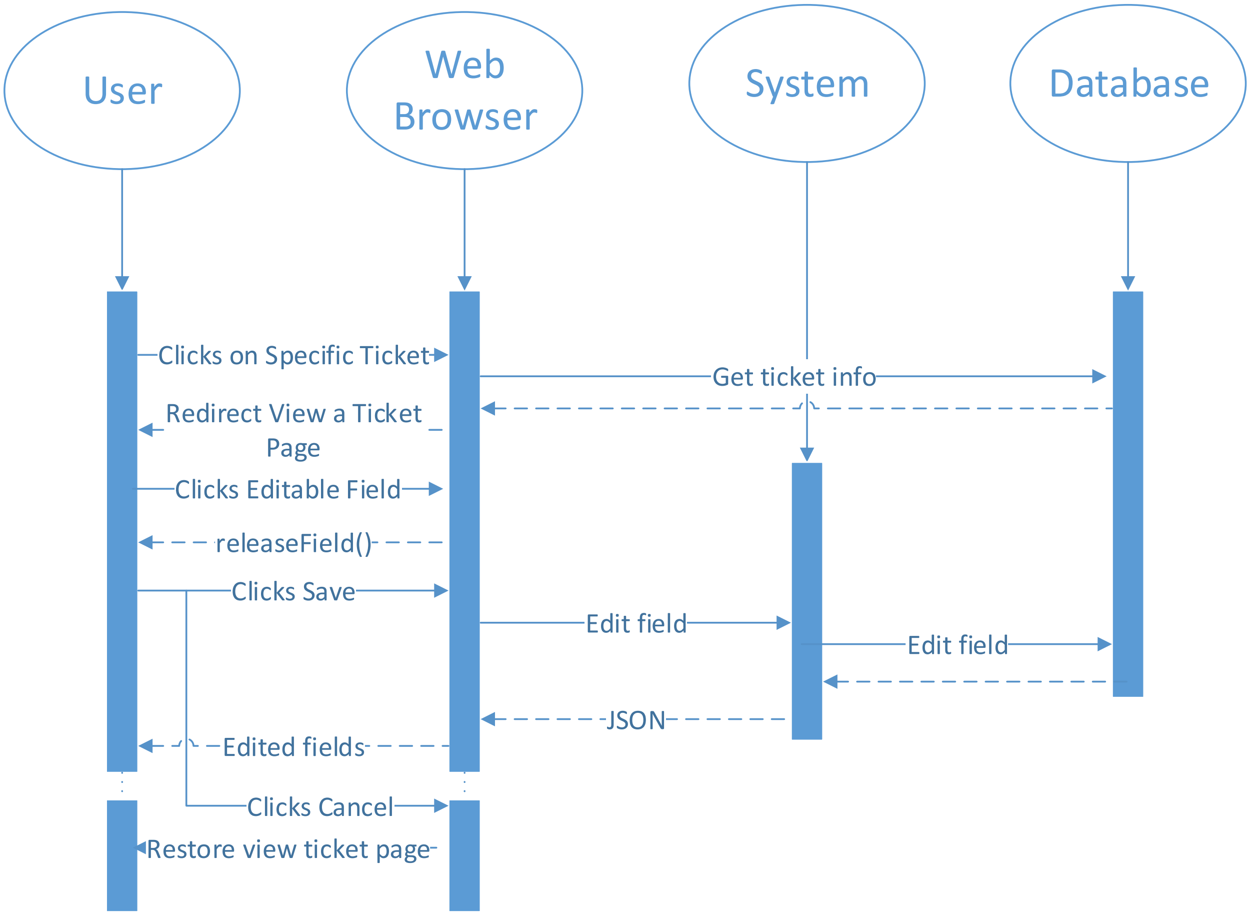
Create a task



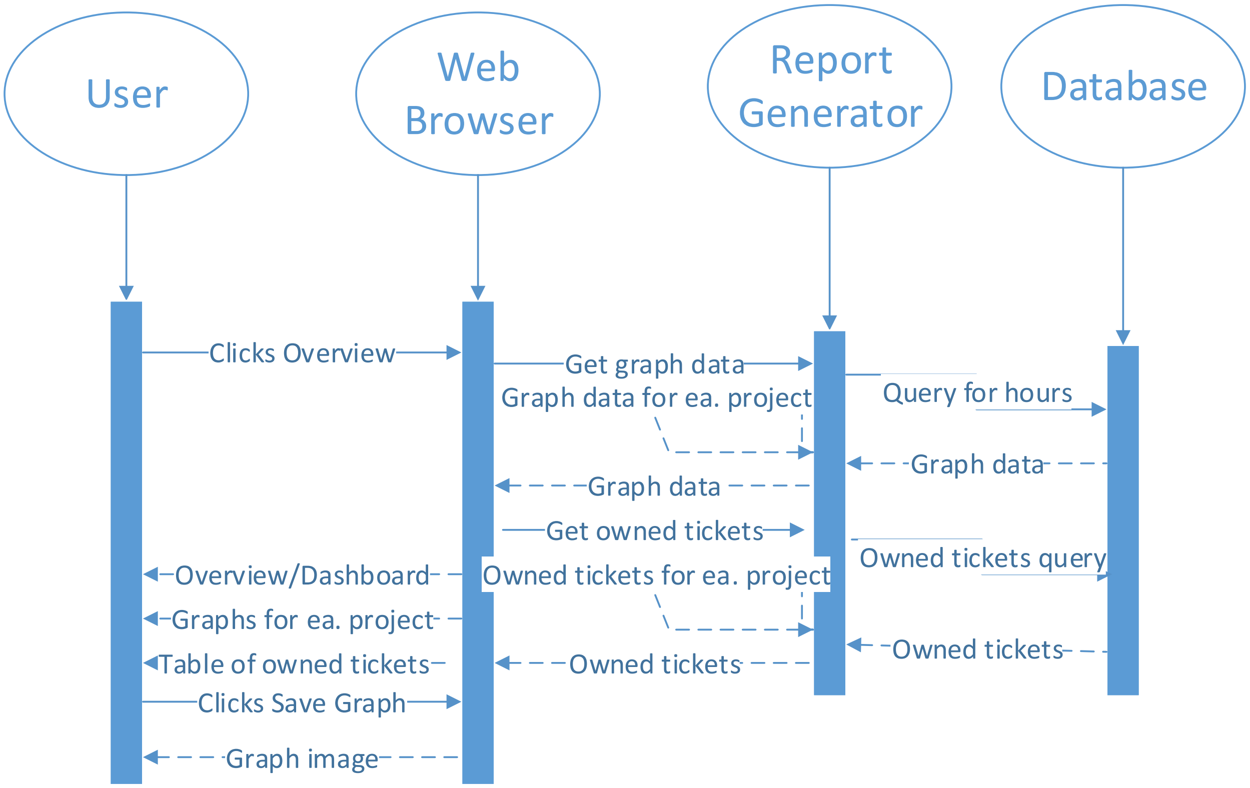
Edit profile



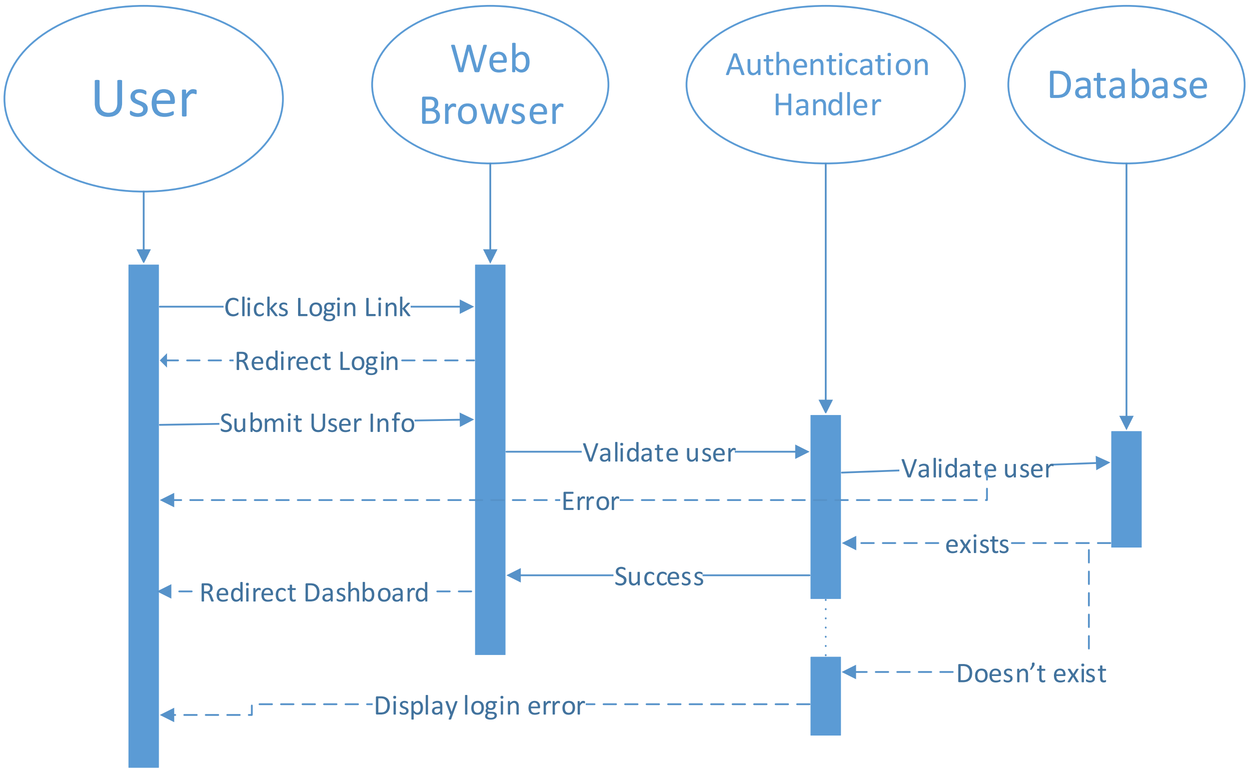
Edit a Ticket



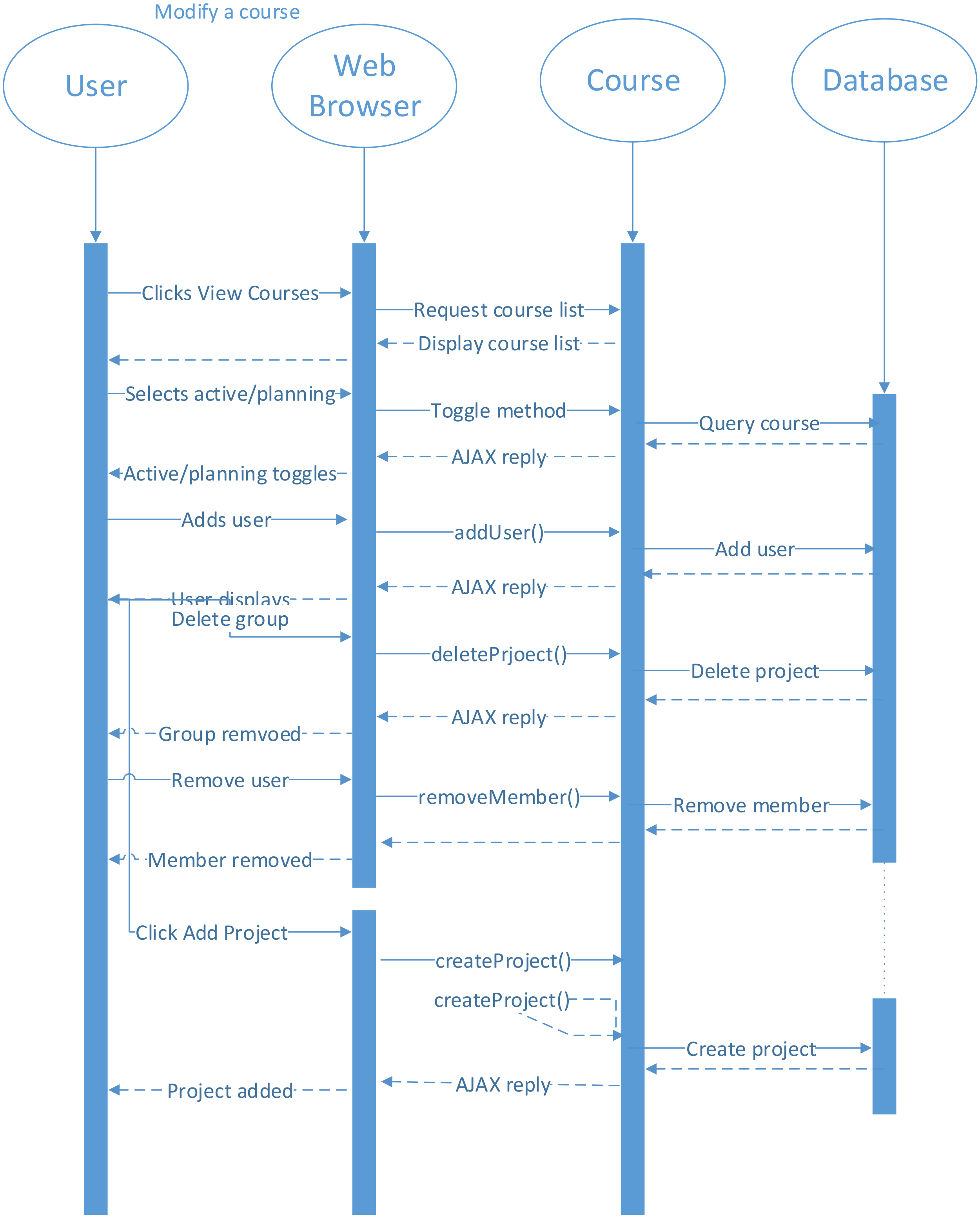
Generate reports



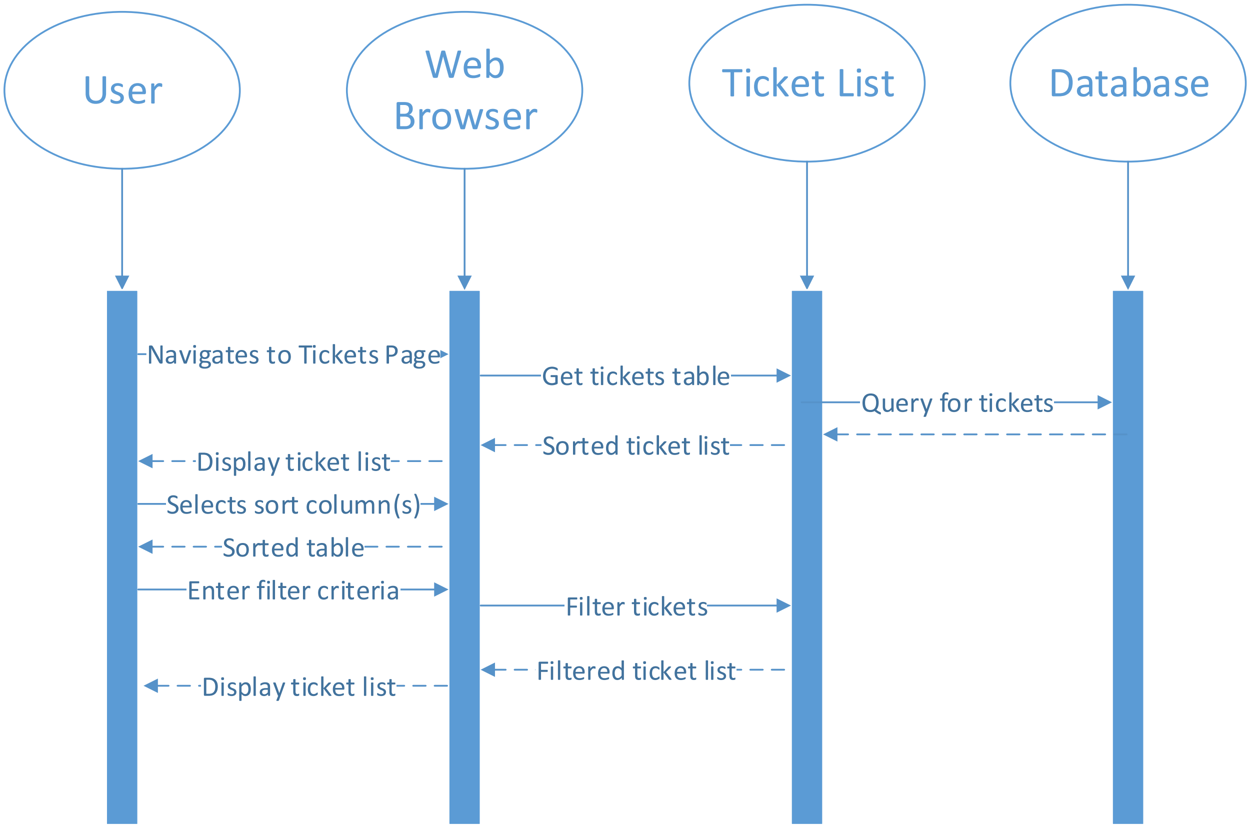
Login


User
Web Browser
Authentication Handler
Database
Clicks Login Link
Redirect Login
Submit User Info
Validate user
Validate user
Error
exists
Success
Redirect Dashboard
Doesn't exist
Display login error

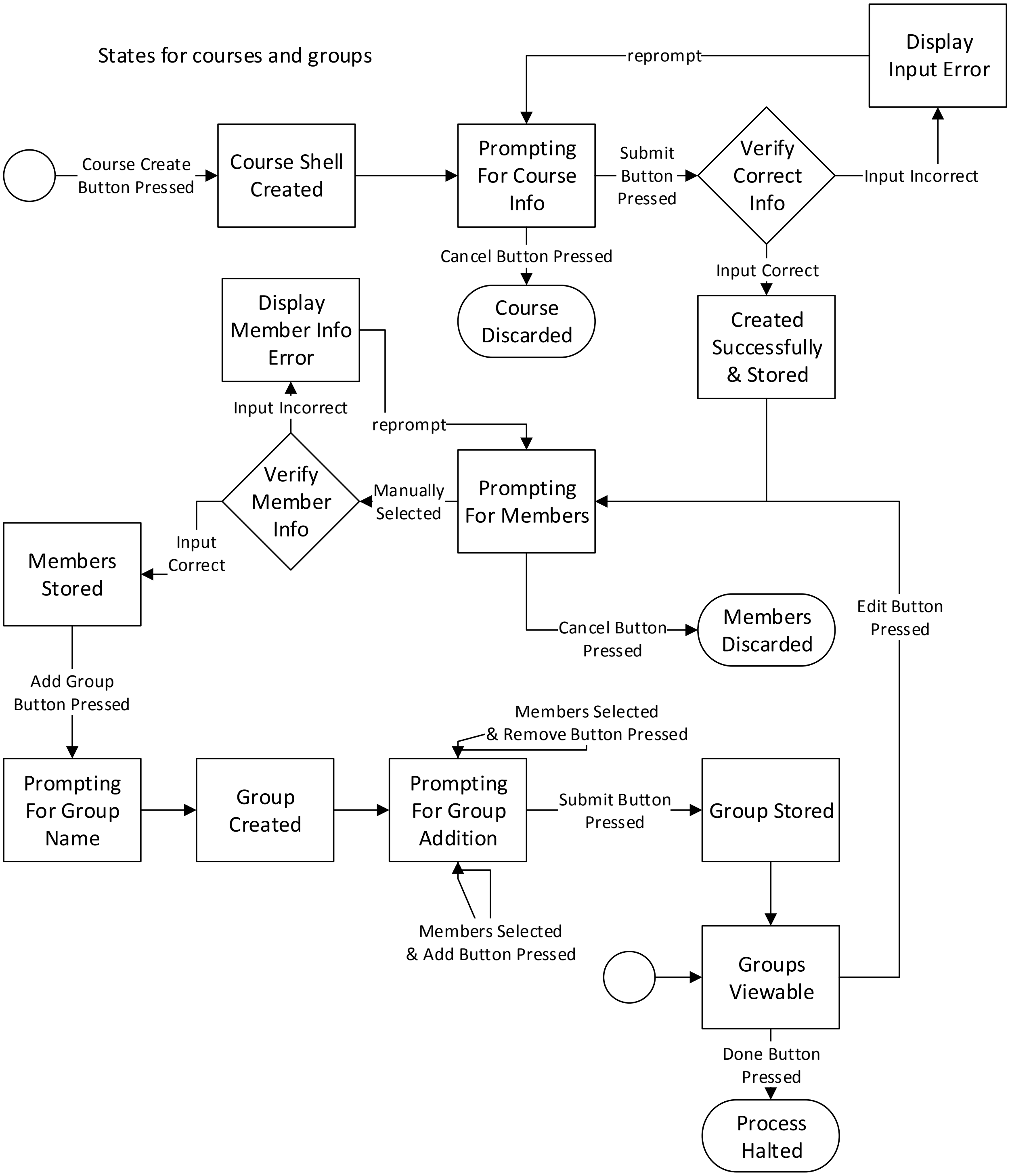
Modify a course



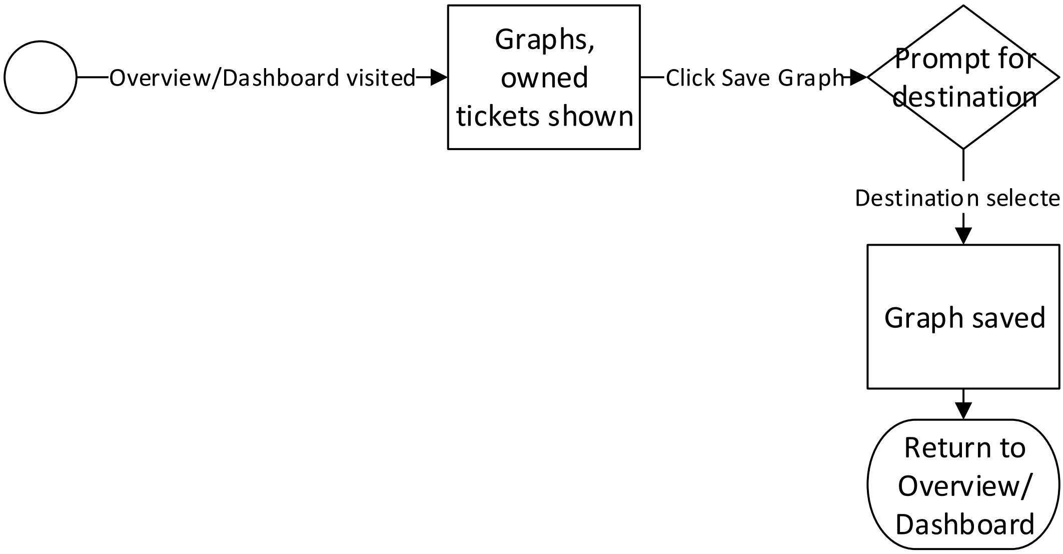
Search Tickets



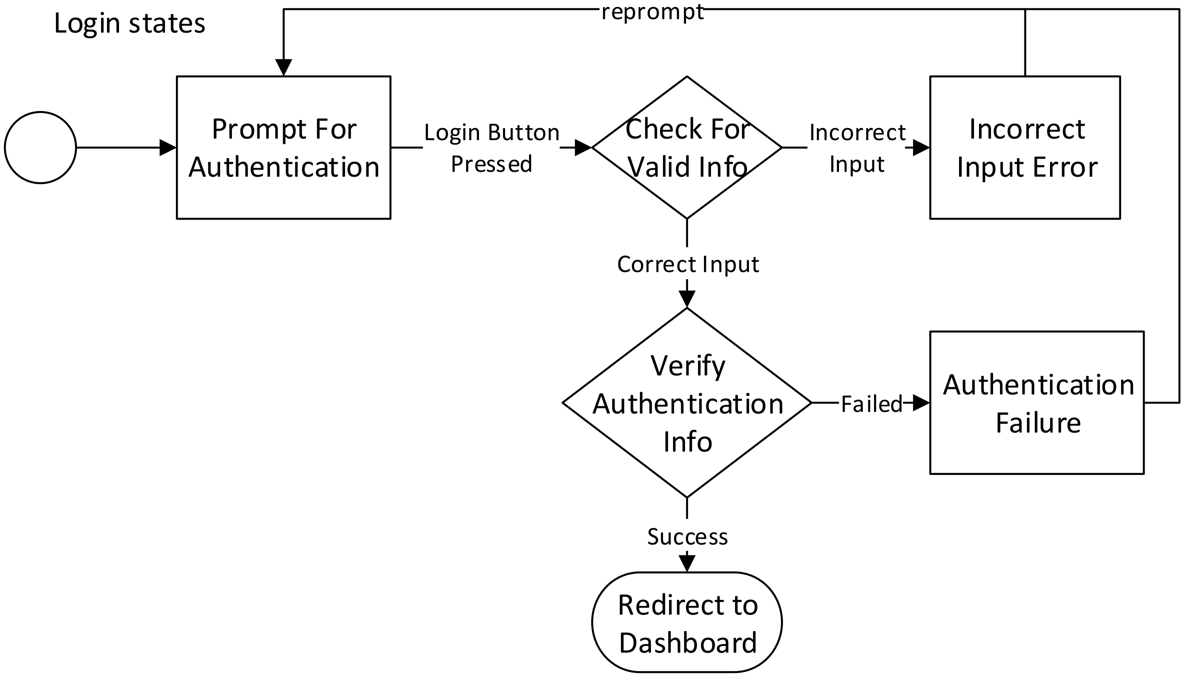
States for courses and groups



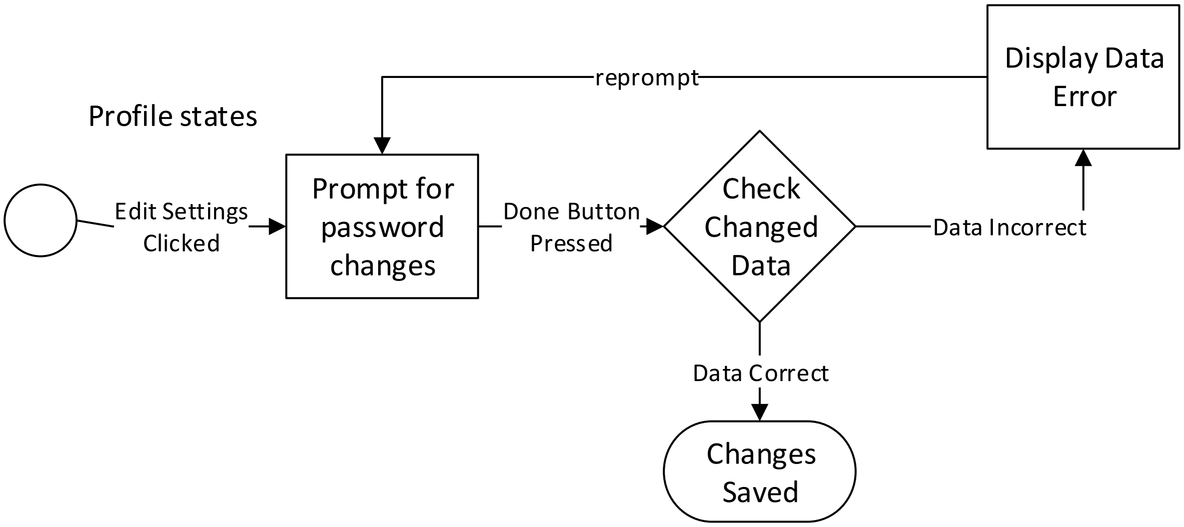
Reporting states

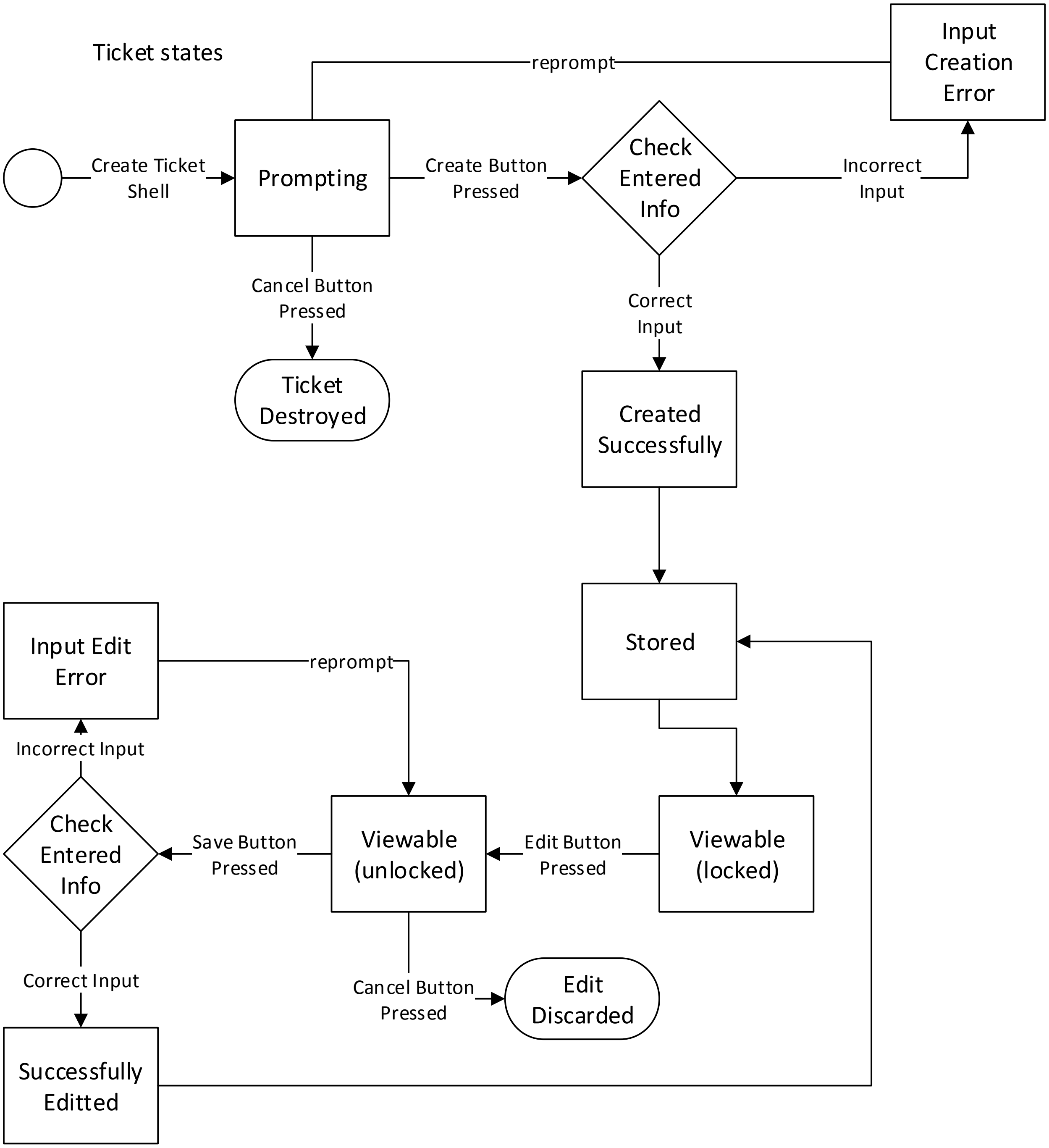

Overview/Dashboard visited
Graphs, owned tickets shown
Click Save Graph
Prompt for destination
Destination selected
Graph saved
Return to Overview/ Dashboard

Login states



Profile states




Ticket states
reprompt
Input Creation Error
Create Ticket Shell
Prompting
Create Button Pressed
Check Entered Info
Incorrect Input
Cancel Button Pressed
Correct Input
Ticket Destroyed
Created Successfully
Stored
Input Edit Error
reprompt
Incorrect Input
Check Entered Info
Save Button Pressed
Viewable (unlocked)
Edit Button Pressed
Viewable (locked)
Correct Input
Cancel Button Pressed
Edit Discarded
Successfully Editted