Guttersnipe (0.3)

Software Requirement Specifcations

1. Introduction

1.1. Purpose

First Row Movies (FRW) promises to be the premiere community for purchasing and sharing movies over the web.

This Document will detail the features of First Row Movies, and will serve as a guide to developers and as a legal document for prospective clients.

1.2. Scope. iuhihhihiuiuhi

1.2.1. Users

1.2.1.1. User: This is the base class for all users of the system. All users of the System can search through all the movies, browse each movie, read comments and complain about inappropriate comments.

1.2.1.2. Guest User: The Guest User class inherits all the functionality of User. In addition, the Guest User has the capacity to register to become a user, to login as a user, and to retrieve a lost username.

1.2.1.3. Registered User: Registered User is an abstract base class that inherits all the functionality of User. In addition, the Registered user can watch a movie, reset a password, and logout from the system.

1.2.1.4. Admin User: Admin User inherits all the functionality of Registered User. In addition, the Admin User can erase comments, warn users, suspend users, and delete users.

1.2.2. Business Objects

1.2.2.1. Movie: Each movie is categorized according to its genre, director, actors, and date.

1.2.2.2.

1.2.2.3.

1.2.2.4.

1.2.2.5.

By keeping track of each user's history of movie purchases and by comparing the user's interests with other users with similar interests, First Row Movies will be able to recommend new purchases to customers.

1.3. AcronymsandAbbreviations

1.3.1. FRM: First Row Movies

1.3.2. U: User

1.3.3. GU: Guest User

1.3.4. RU: Registered User

1.3.5. AU: Admin User

1.3.6. SRS: Software Requirements Specification

1.3.7. GUI: Graphical User Interface.

1.3.8. FSM: Finite State Machine.

1.3.9. 1 DB: Database.

1.3.10. 1 ERCD: Entity-Relation Class Diagram.

1.4. References

Appendix A: User Interface Prototypes

1.5. Summary

The rest of this SRS is organized as follows:

1.5.1. Section 2: Gives the overall description of FRW. It contains the Use-Case diagram and descriptions for FRW. Section 2 also contains the assumptions and dependencies of the system.

1.5.2. Section 3: Gives specific software requirements and functionalities in the form of Mini Use- Case diagrams along with accompanying Collaboration diagrams, Finite State Machine of the system, and ER Class diagram of the system. This section also contains supplementary software requirements of the systems.

1.5.3. UI Components: The appendix contains user interface prototypes for the system.

PROPOSAL: ***GUTTERSNIPE***

1. What is the site/app?

Guttersnipe is a web portal and mobile app that caters to anarcho-communist street youth (and adults) who desire to subvert capitalism by sharing resources.

It will enable people to broadcast to each other locations of abandoned properties that can be squatted as well as dumpsters which can be dived for food and other resources.

Eventually, other types of resource sharing will be integrated into this product. A calender will be used to map and schedule Really Really Free Markets and Food Not Bombs meals.

Perhaps ridesharing will also be integrated

Perhaps maps of the train system and good times for hopping

Further development will require a close study of the writings of Kropotkin and Fourier.

2. What need does this meet? or problem does it solve?

This application serves the urgent need to overthrow capitalism by helping people to self-organize outside and beyond the market of commerce.

The ultimate intention is to facilitate the creation of alternate avenues of exchange, freely organized by free individuals.

3. Who is going to go/use to this site/app?

In the current incarnation, it is mostly aimed towards the freegans gutterpunks, who live off of dumpstered food, live in squatted housing, and travel by hopping trains.

As we get a better sense on the needs of the anticapitalist community and possibilities for alternative organizing, we will expand the possibilities for anti-market resource sharing.

4. Why will they go to this site/app?

To find food, clothing, shelter, etc.

5. Why will they keep coming back to your site/app?

See above.

6. How is it different from other similar sites?

There are similar sites of various types, but many of them have certain faults.

There is a site called rideshare.com; there is a site named couchsurfer.com; there is freecycle.com, which allows the sharing of goods.

These are all laudable efforts. Some of these are marred by an underlying desire for profit. But some of them are motivated out of genuine desire to promote Mutual Aid.

The very mission of Guttersnipe.net will be to promote the organization of the lumpenproletariat and to create alternative exchanges outside of capitalism. This mission will enable Guttersnipe.net to be singularly focused on this goal.

It will thus be able to bring together whatever resources necessary for the undermining of capitalism: the various services— such as squatting, dumpster diving, hitchhiking, train hopping, resource sharing, etc — will be coordinated on a singular web portal.

In addition, there are several web portals that are dedicated towards the promotion of anarcho-communist goals..

Such sites are

- Freegan.info

- Picture the Homeless
- Squat.net
- Foodsharing (Germany)

Many of our initial design specifications will be taken from the freegan group and Picture the Homeless.

In addition, we intend Guttersnipe to be cross platform, available both via the web and as a mobile app. To my knowledge, there are not yet any apps dedicated with such a task.

7. What steps will a person go through interacting with the site/app?

Most of the various interactions will be handled using forms.

The various services offered by Guttersnipe all boil essentially boil down to two types of transaction:

1. information submission;
2. information retrieval.

One person posts about an abandoned building or a good dumpster; another person searches for such information.

There may be very many different interfaces for the reporting and retrieval. Some of the data will be entered and retrieved using forms and text inputs; some will be accessed through map interfaces; some will be accessed through calendar interfaces.

In addition, there may be some need for identity management. Some users may choose to register accounts. Others may choose to always have singular, anonymous transactions.

Practical Constraints

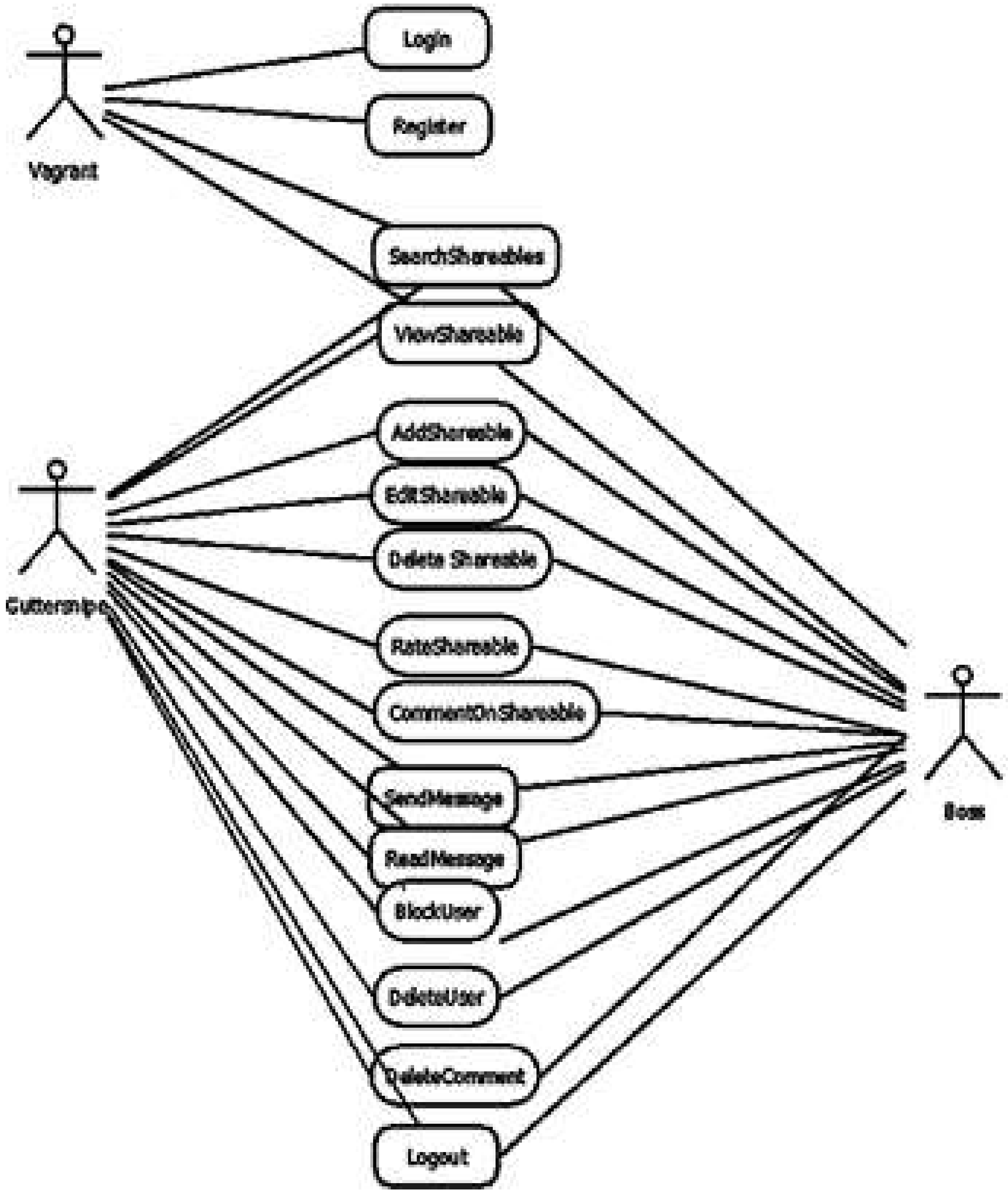
Security

We will have to build in a security infrastructure in the project in order to guarantee anonymity of transactions.

Tor will be used to anonymize transactions.

2. Overview

2.1 User Case Diagram



2.2 Use Case Descriptions

User (U)

ViewShareable: Any U may browse a search result.

SearchShareable: Any U may search for movies using any combination of search vectors (id, title, starts, director, year, etc.)

Vagrant (V)

Login: Any V can login to the system, which transforms V into a G or B .

Register: Any V may register to become a G.

Retrieve Username: Any G who has forgotten username may request (as V) that user information will be emailed.

Guttersnipe (G)

Logout: Any G (G or B) may log out of own account.

Reset Password: Any G (G or B) may reset own password from the Account Settings page.

AddShareable: Can add a Shareable resource.

EditShareable: Can edit own Shareable.

Delete Shareable: Can delete own Shareable.

RateShareable: Can rate any shareable

CommentOnShareable: Can commenton any shareable.

Send Meetup: Can communicate message or receive meetup schedule from any other Guttersnipe.

Read Message: Can read message inbox.

Set Schedule: Can set daily, weekly, yearly calendar.

BlockUser: Can block any other user except Boss.

DeleteUser: Can delete own account.

DeleteComment Can delete own comment.

Boss (B)

Boss has same capabilities as Guttersnipe, but they are unrestricted to apply to all system users.

2.2 Assumptions and Dependencies

a. The client can run on any contemporary web browser that supports standard HTML, CSS 2.0, JavaScript, and PHP 5.0. It should run well on any contemporary web browser, including the latest versions of Internet Explorer, Firefox, Chrome, Safari, and Opera..

b. The server must run a PHP 5.0 processing daemon and a MySQL database. Any LAMP/WAMP setup should suffice, as will any other server (IIS, etc) with PHP and mySQL modules installed.

Collaboration Diagrams

LOGIN

REGISTER

SEARCH SHAREABLE

VIEW SHAREABLE

ADD SHAREABLE

DELETE SHAREABLE

RATE SHAREABLE

COMMENT ON SHAREABLE

SEND MEETUP MESSAGE

READ MESSAGE

BLOCK USER

DELETE USER

DELETE COMMENT

LOGOUT

3. Specific Requirements
3.1 Collaboration Diagrams
3.1.1 Shared Collaboration Diagrams
Figure A1: Search
Figure A2: Browse
Figure A3: View Movie Info
3.1.2 Guest User Collaboration Diagrams
B1: Register
Figure B2: Login
3.1.3 Regular User Collaboration Diagrams
C1: Watch Movie
Figure C2: Rate
Figure C3: Comment
Figure C4: Complain
Figure C5: Add to Cart
Figure C6: View Cart
Figure C7: Checkout
Figure C8: Logout
3.1.4 Admin User Collaboration Diagrams
D1: Erase Comment
Figure D2: Ignore Complaint
Figure D3: Warn User
Figure D4: Logout
Guest User Class
(extends the User Class)
Attributes:
*** private $userID;
*** private $username;
*** private $accountType; *** private $userEmail;
Functions:
*** public function login($userID, $userPassword)
// This allows a guest user to log in to the system and become a registered user or admin user
*** public function register()
// This allows a guest user to become a registered user
*** public function complainComment($commentID)
// This allows a guest user to send a complaint about a comment identified by commentID
RegisteredUser Class
(extends the User Class)

Attributes:
*** private $userID;
*** private $username; *** private $userEmail;
Functions:
*** public function getID()
// This returns the id of the registered user
*** public function getUsername()
// This returns the username of the registered user
*** public function getEmail()
// This returns the email address of the registered user
*** public function complainComment($commentID)
// This allows a registered user to send a complaint about a comment identified by commentID
*** public function login($userID, $userPassword)
// This allows a registered user to send a complaint about a comment identified by commentID
*** public function rateMovie($userID, $movieID, $rating)
// This allows a registered user to give a rating to a movie.
*** public function submitComment($userID, $movieID, $commentText)
// This allows a registered user to post a comment to the movie page identified by movieID
Movie Class
Attributes:
*** private $movieID;
*** private $movieName;
*** private $movieYear;
*** private $movieSummary;
*** private $movieGenre;
*** private $movieDirector;
*** private $movieStars;
*** private $movieRuntime;
*** private $movieImageLocation; *** private $movieLocation;
*** private $movieRating;
*** private $moviePrice;
Functions:
*** function __construct($movieID) // Constructor for the Movie class
*** public function playMovie($userID, $movieID)
// Verifies that user has purchased the movie, plays movie in movieLocation
*** public function browse()
// Displays all information about movie

*** public function getMovieId() // Returns movie id
*** public function getMovieName() // Returns movie name
*** public function getMovieYear() // Returns movie year
*** public function getMovieSummary() // Returns movie summary
*** public function getMovieGenre() // Returns movie genre
*** public function getMovieDirector() // Returns movie director
*** public function getMovieStars() // Returns movie stars
*** public function getMovieRuntime() // Returns movie runtime
*** public function getMovieImageLocation() // Returns file location of movie poster
*** public function getMovieLocation() // Returns location of movie file
*** public function watchMovie() // plays the movie
*** public function getPrice() // Returns movie price
Movie Class
Attributes:
*** private $movieID;
*** private $movieName;
*** private $movieYear;
*** private $movieSummary;
*** private $movieGenre;
*** private $movieDirector;
*** private $movieStars;
*** private $movieRuntime;
*** private $movieImageLocation; *** private $movieLocation;
*** private $movieRating;
*** private $moviePrice;
Functions:
*** function __construct($movieID) // Constructor for the Movie class
*** public function playMovie($userID, $movieID)
// Verifies that user has purchased the movie, plays movie in movieLocation
*** public function browse()
// Displays all information about movie
*** public function getMovieId() // Returns movie id
*** public function getMovieName() // Returns movie name
*** public function getMovieYear() // Returns movie year
*** public function getMovieSummary() // Returns movie summary
*** public function getMovieGenre() // Returns movie genre
*** public function getMovieDirector() // Returns movie director
*** public function getMovieStars() // Returns movie stars
*** public function getMovieRuntime() // Returns movie runtime
*** public function getMovieImageLocation() // Returns file location of movie poster

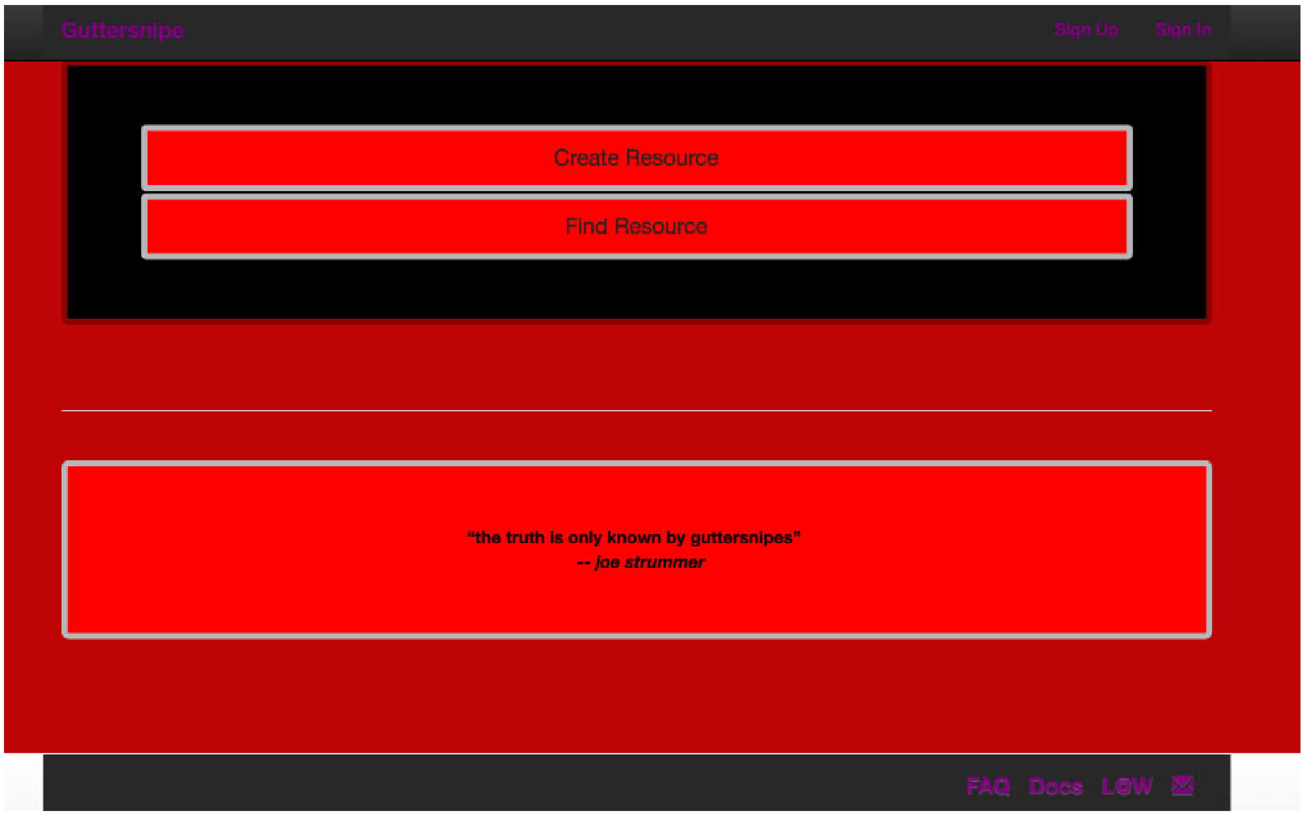
*** public function getMovieLocation() // Returns location of movie file
*** public function watchMovie() // plays the movie
*** public function getPrice() // Returns movie price
Cart Class
Attributes:
*** private $totalPrice;
*** private $checkoutCart = array();
Functions:
*** public function getAmountItems()
// Returns the length of $checkoutCart
*** public int getCurrentTotal()
// Returns the current total price of items in the cart
*** public function removeFromCart($movie) // Removes a movie from the checkout cart
*** public int checkout($userID, $allMovieIDs)
// Executes purchase of all movies by the user.
Comment Class
Attributes:
private $commentID; private $commentText; private $userName; private $timestamp;
Functions:
*** function __construct($commentID, $commentText, $userName, $timestamp) // Constructor for the Comment class
*** public int getCommentID() ***
// Returns the ID of the comment
*** public function getCommentText() // Gets the text of the comment
*** public function getUserName()
// Gets the name of the commenter
*** public function getTimestamp() // Gets the date of the commenter
Complaint Class
Attributes:
*** private $complaintID; *** private $commentID; *** private $reason;
Functions:
*** function __construct($complaintID, $commentID, $reason) // Constructor for the Complaint class
*** public function getComplaintID() // Returns the complaint ID
*** public int getCommentID()
// Returns the ID of the comment being flagged
*** public function getReason()
// Gets the reason for the complaint
Cart Class

Attributes:
*** private $totalPrice;
*** private $checkoutCart = array();
Functions:
*** public function getAmountItems()
// Returns the length of $checkoutCart
*** public int getCurrentTotal()
// Returns the current total price of items in the cart
*** public function removeFromCart($movie) // Removes a movie from the checkout cart
*** public int getCurrentTotal($userID, $allMovieIDs) // Executes purchase of all movies by the user.
Rating Class
Attributes:
*** private $movieID; *** private $average ; *** private $oneStars; *** private $twoStars; *** private $threeStars; *** private $fourStars; *** private $fiveStars;
Functions:
*** function __construct($movieID) // Constructor for the Rating class
*** public function getAverage () // Returns the average rating
*** public int getOnes()
// Returns the number of one star ratings
*** public int getTwos()
// Returns the number of two star ratings
*** public int getThrees()
// Returns the number of three star ratings
*** public int getFours()
// Returns the number of four star ratings
*** public int getFives()
// Returns the number of five star ratings

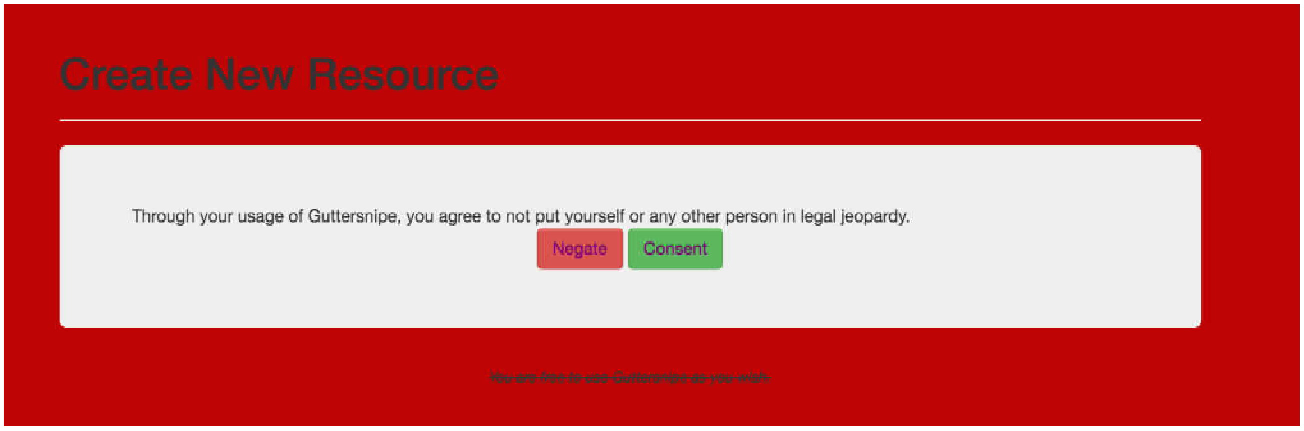
Figure : Complaints.php
Figure :
Figure : register.php
Figure : firstLogin.php
Figure: ViewMovieInfo.php
Figure : browse.php
* Add functionality to current Classes
(
U,
GU, RU, AU, Movie, Cart

)
* Create new Object Classes
(
Player,
Comment,
Session?
All Database Returns Need Object Wrapper
)
* Create new Logic Classes
(
SearchManager, BrowseManager, MovieViewManager, LoginManager, RegisterManager, MovieRatingsInterface, MovieProcessor, CommentManager, CartManager, CheckoutManager, PasswordControl, LogoutControl, CommentControl, FlaggedCommentsManager, WarningSystem, DeleteManager
)
Create new GUI Classes
(
SearchGUI,
BrowseGUI, ViewMovieGUI, LoginGUI, MainCustomerPage, MainAdminPage, RegisterGUI, RegistrationSuccessGUI, MoviesBought, MoviePlayerInterface, CommentGUI, ViewCartGUI,
CartGUI, CheckoutGUI, ConfirmationGUI, ResetPasswordGUI, GuestPage, FlaggedCommentsGUI, ComplaintInterface, ComplaintsInterface, DeleteUserGUI
)
Create Database Tables
(
MOVIES,
R_USERS, RATINGS, PURCHASES, COMMENTS, COMPLAINTS
)
Create Screen Shots
(
SearchGUI,
BrowseGUI, ViewMovieGUI, LoginGUI, MainCustomerPage, MainAdminPage, RegisterGUI, RegistrationSuccessGUI, MoviesBought, MoviePlayerInterface, CommentGUI, ViewCartGUI,
CartGUI, CheckoutGUI, ConfirmationGUI, ResetPasswordGUI, GuestPage, FlaggedCommentsGUI, ComplaintInterface, ComplaintsInterface, DeleteUserGUI
)

GUI COMPONENTS
(v. 0.2.1.5)
angular 1.x
bootstrap 3



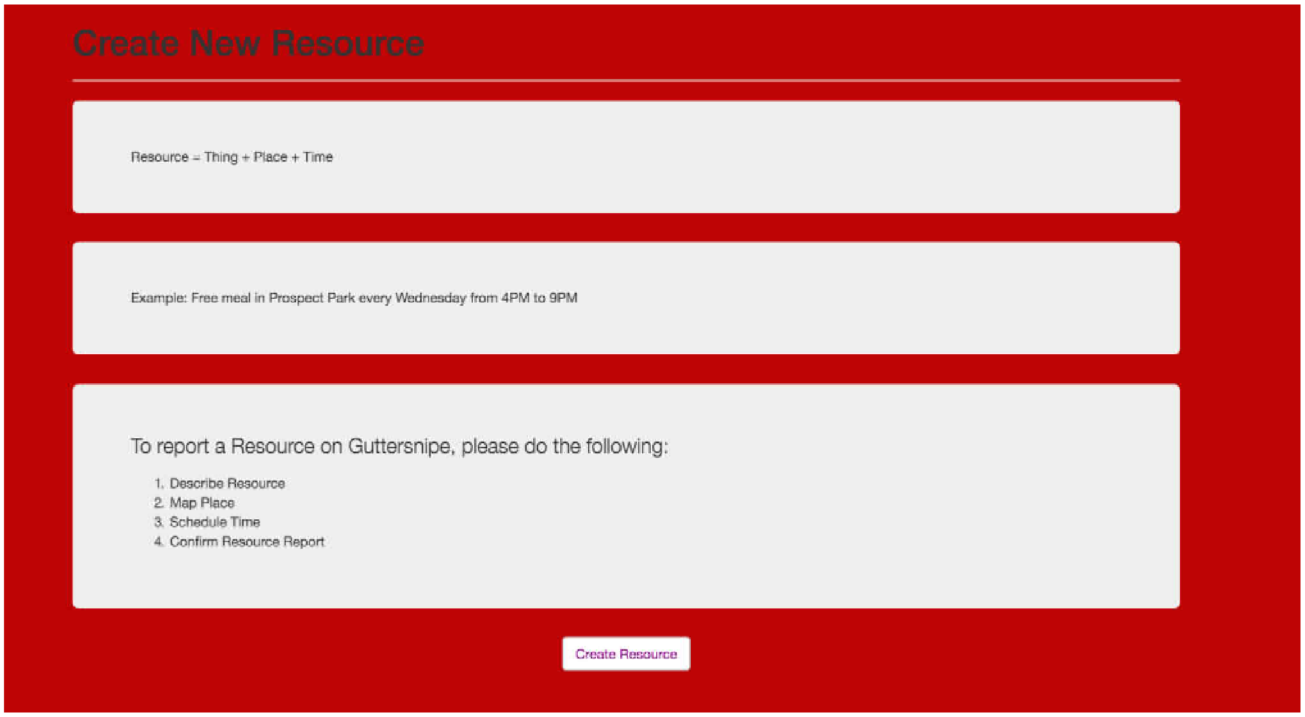
"the truth is only known by guttersnipes"
-- *joe strummer*



Create New Resource

Through your usage of Guttersnipe, you agree to not put yourself or any other person in legal jeopardy.

~~You are free to use Guttersnipe as you wish.~~

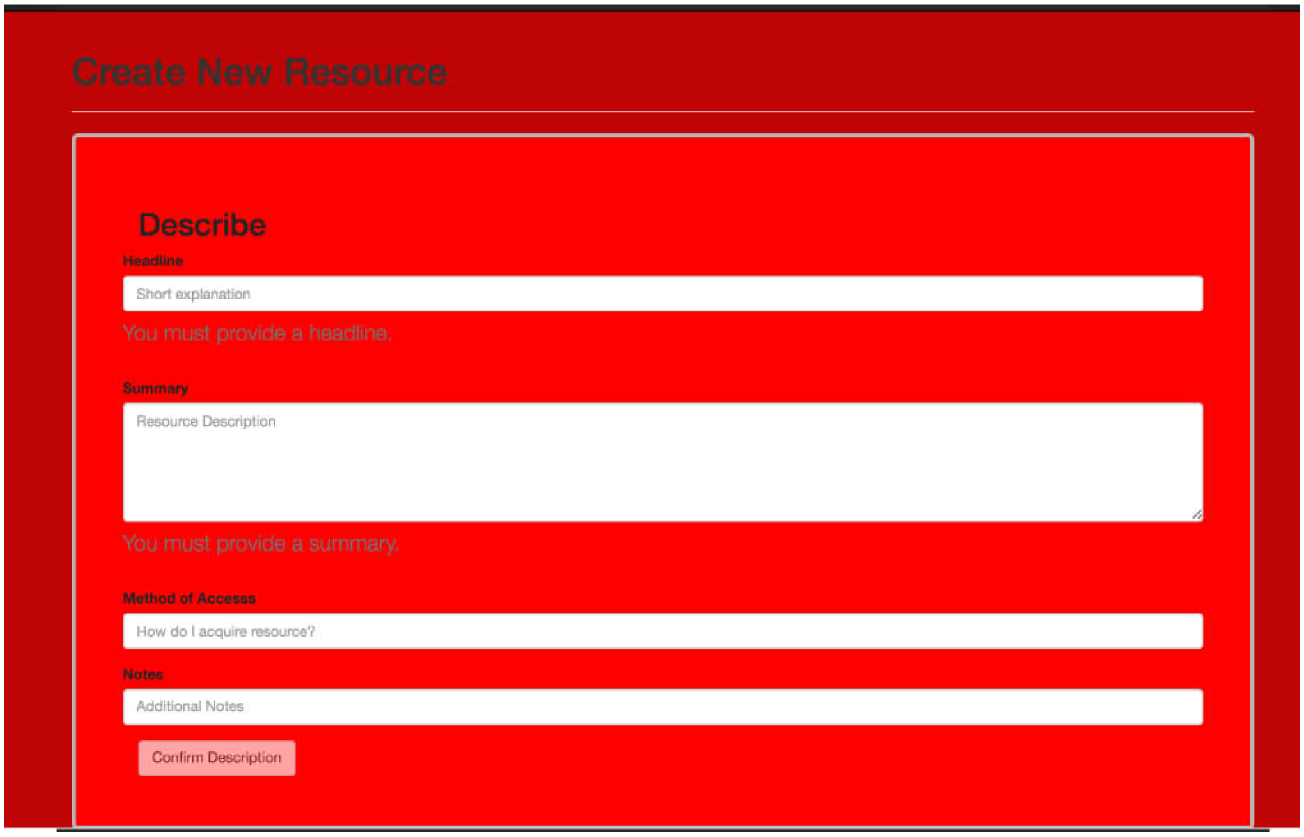

Create New Resource

Resource = Thing + Place + Time

Example: Free meal in Prospect Park every Wednesday from 4PM to 9PM

To report a Resource on Guttersnipe, please do the following:

1. Describe Resource
2. Map Place
3. Schedule Time
4. Confirm Resource Report


Create New Resource

Describe

Headline

Short explanation

You must provide a headline.

Summary

Resource Description

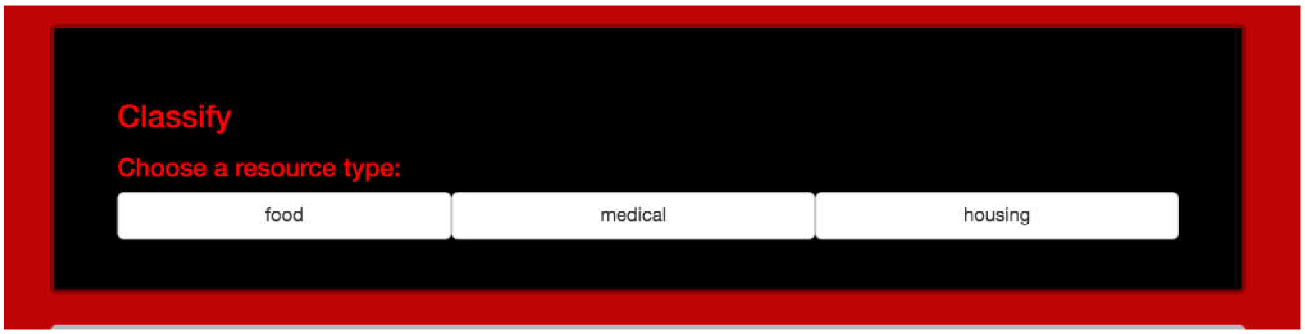
You must provide a summary.

Method of Accesss

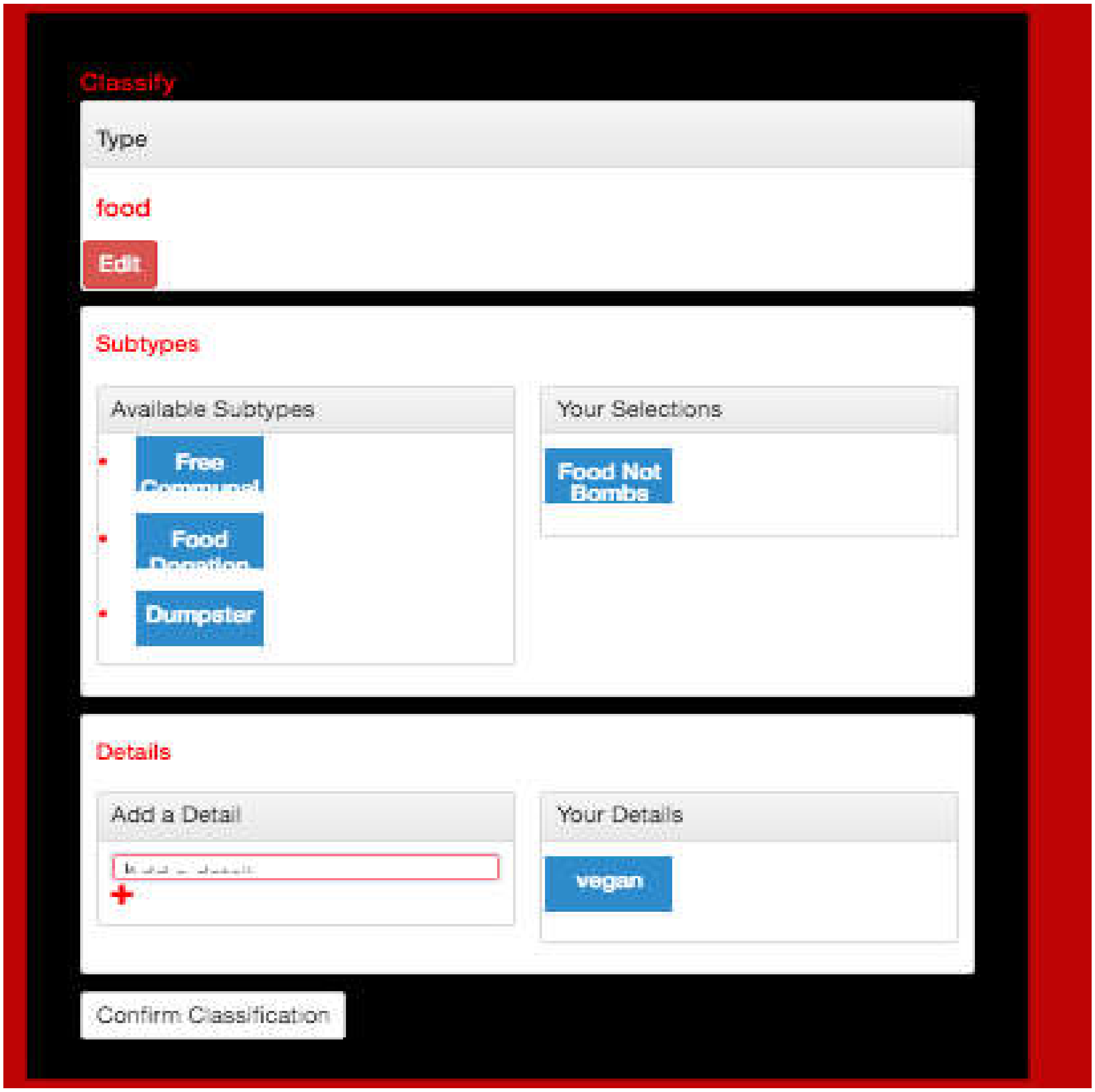
How do I acquire resource?

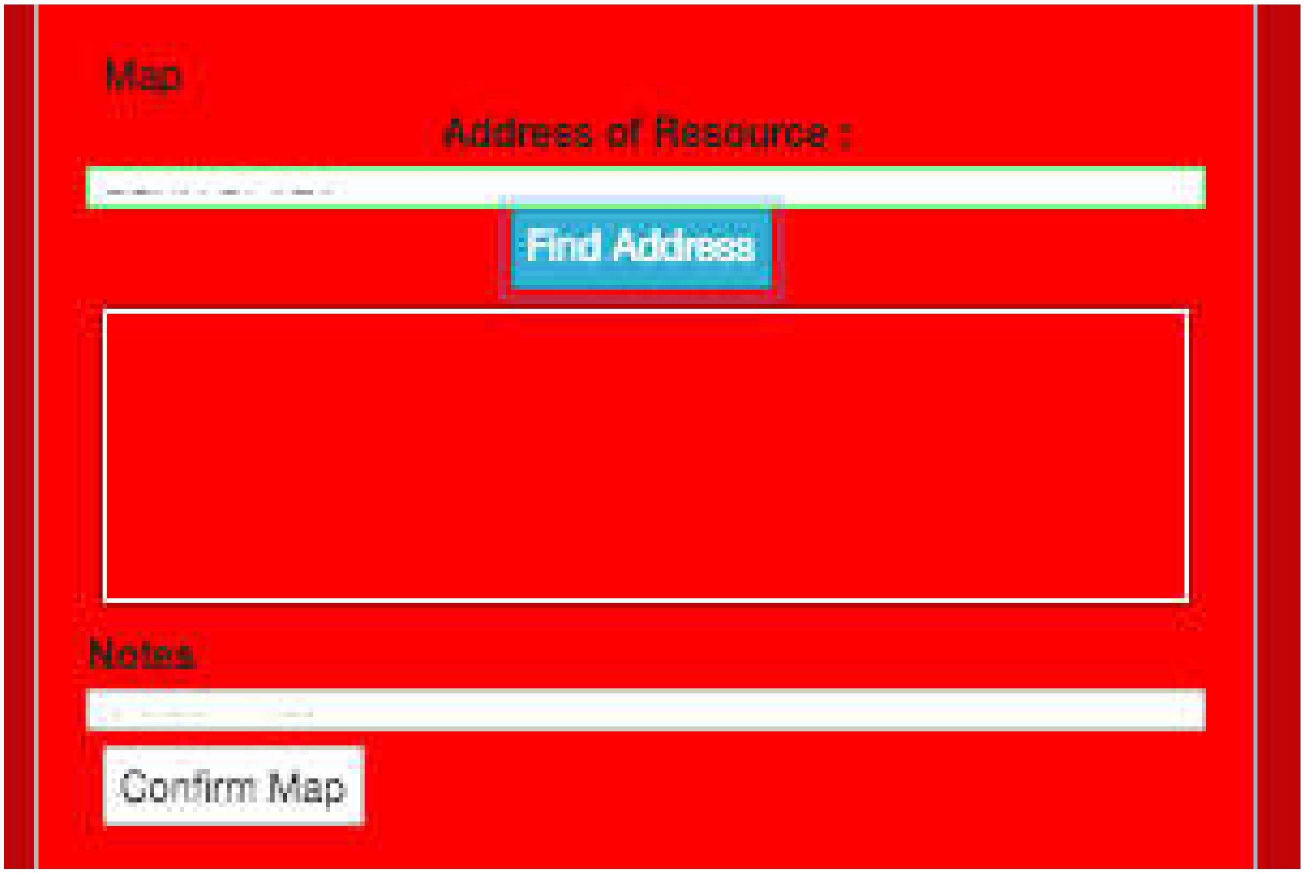
Notes

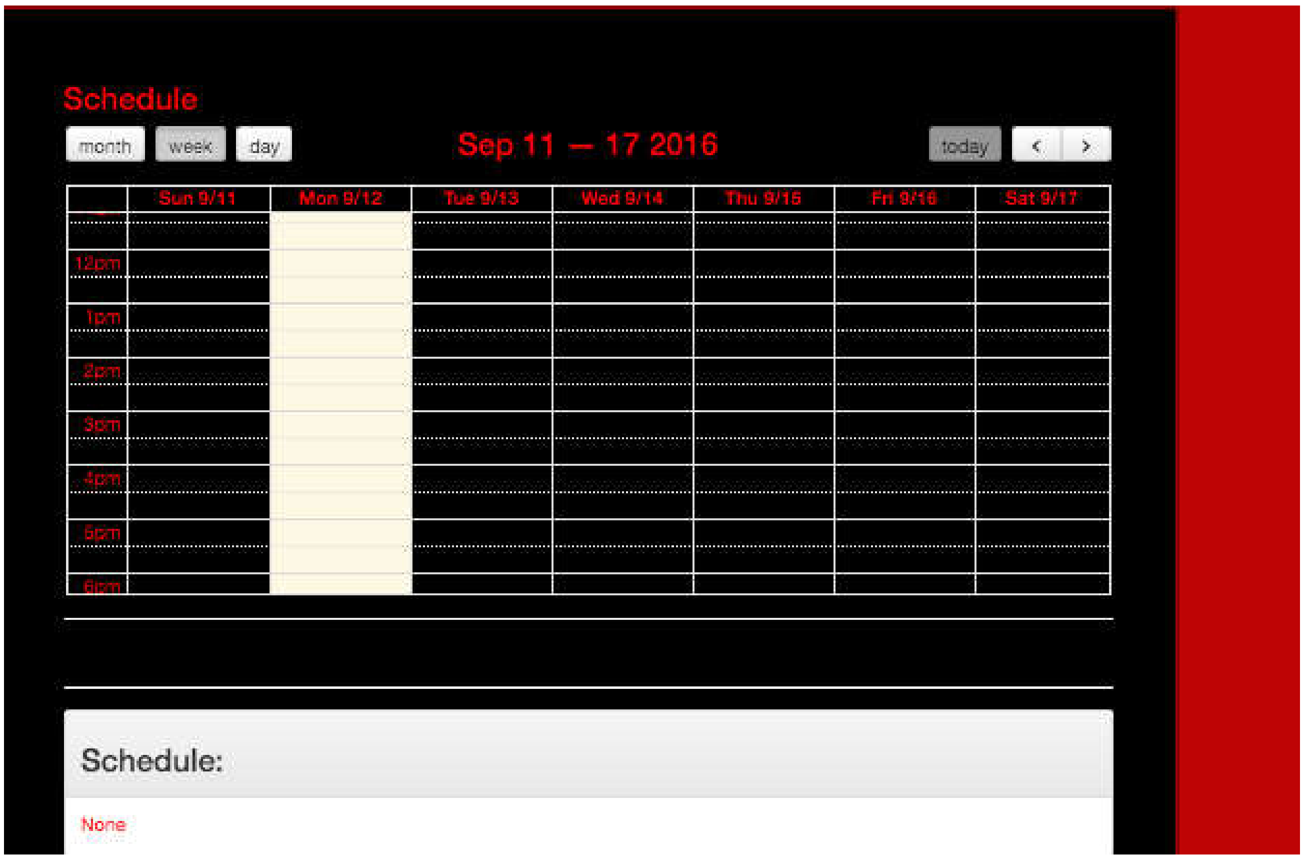
Additional Notes


Classify

Choose a resource type:


Classify
Type
food
Edit
Subtypes
Available Subtypes
Free Communal
Food Donation
Dumpster
Your Selections
Food Not Bombs
Details
Add a Detail
Your Details
vegan
Confirm Classification


Map
Address of Resource :
Find Address
Notes
Confirm Map



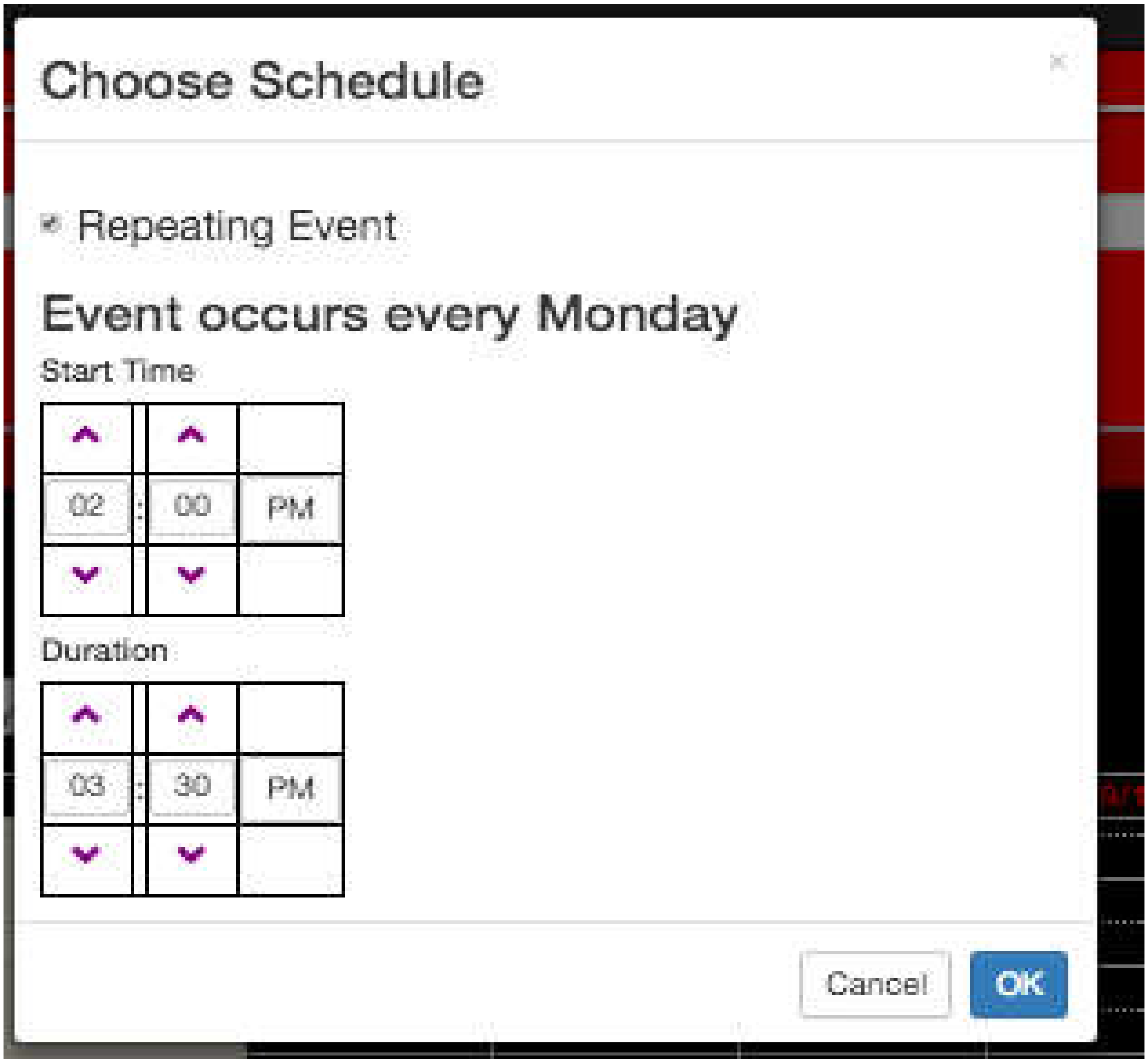
Schedule

Sep 11 — 17 2016

	Sun 9/11	Mon 9/12	Tue 9/13	Wed 9/14	Thu 9/15	Fri 9/16	Sat 9/17
12pm							
1pm							
2pm							
3pm							
4pm							
5pm							
6pm							

Schedule:

None



Choose Schedule

×

☑ Repeating Event

Event occurs every Monday

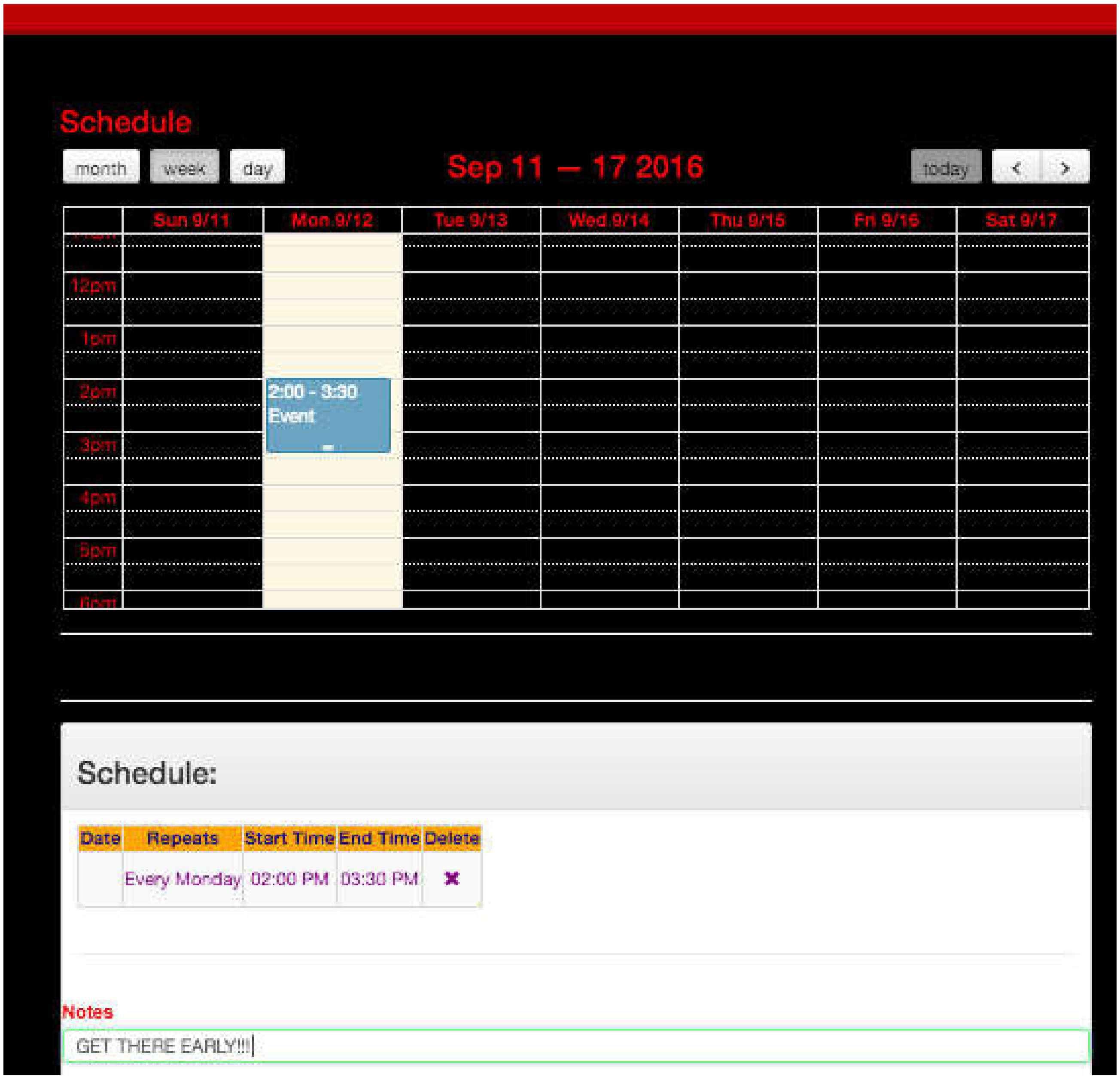
Start Time

02 : 00 PM

Duration

03 : 30 PM

Cancel OK



Schedule

Sep 11 — 17 2016

	Sun 9/11	Mon 9/12	Tue 9/13	Wed 9/14	Thu 9/15	Fri 9/16	Sat 9/17
12pm							
1pm							
2pm		2:00 - 3:30 Event					
3pm							
4pm							
5pm							
6pm							

Schedule:

Date	Repeats	Start Time	End Time	Delete
	Every Monday	02:00 PM	03:30 PM	✖

Notes

GET THERE EARLY!!!