

# TRental Project Overview

We are designing a textbook rental website called TRental. On TRental, textbook owners can create book postings that describes the details of their rental offer such as textbook title, textbook authors, and price. On the other hand, textbook borrower can find postings based on textbook name, availability, and rent price. When an owner made contact with a borrower and the deal is made, the owner will (manually) mark the corresponding posting unavailable until the book is returned to him and becomes available again. This way, a posting can be reused multiple times and rent to multiple user at different time frames. When the textbook owner no longer wants to rent out his books, he can then remove the posting. TRental also supports a rating system where users can leave comments and reviews under the ratings section of a particular posting based on factors such as the content and the condition of the associated book. Users who are interested can read previous ratings on a postings and make their decisions. If they have any question, they will use the inbox messaging system to contact the book owner.

The goal of TRental is to provide a free platform where textbooks can be effectively reused and recycled. Students often purchase textbooks at high prices, but end up leaving them unused on bookshelves for ages. On the other hand, there are people who wish to use a book only for a short period of time and not have to pay full price to buy it. Our idea is to create a site for these users; book holders who are not going to touch their book for a while can lend them out to the others who need it temporarily and also make money in the process.

## Detail Designs:

### Structure/architecture:

Our website is built using express and EJS template machine (similar to Jade) and follows a standard MVC structure. We separated each file modularly and keep each component away from each other. Our models such as users, postings, ratings, and messages are kept in MongoDB and are modified using the Mongoose ODM through the views and controller component. For instance, when a user clicks on the signup button (the view), the JavaScript and Node server (controller) sends a request to MongoDB to modify and create a new user (model). The three components actively communicates with each other. For example, when a model changes, the views that depends on it changes as well; when a new message is sent to the inbox of a user, the webpage updates itself and displays that message.

In the project, we separated each component and place them in different directory. The EJS template are placed in the "views" folder. Repeated and reused views are placed inside the "partials" sub directory. Our server logics such as route endpoints and APIs are placed inside the "route" folder. The models are placed inside the "models" folder. And finally, the static files such as images, CSS, JavaScript, are located in the static folder.

### Resources and Models:

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

Our website handles four types of resources, users (account), postings, ratings, and messages. A user represents an account and contains information such as name, email and phone number. A posting represents a textbook that a user wants to rent out to the others. A posting contains information such as posting title, book title, book authors, rent price and more. A posting also contains a list of ratings. In other words, each ratings is tied to a posting (and a user who made that rating). A rating represents a review on a textbook posting, it includes a score, a comment section, the rater, and more. The last resource is message. A message represents a text send from a user to another different user. A message holds information such as the messenger, the receiver, and the text. Logically, in a hierarchical view, a user has a list messages, and a list of postings, each of which has a list of ratings.

## Views and UIs:

Our website presents a navigation bar at the top. This navigation bar changes appearance depending on whether the current user is logged in or logged out. When logged out, it provides four tabs, home, about us, login, and signup. When logged in, the navigation bar tabs change to home, about us, search postings, view users, my account, inbox, and logout. We decided that different tabs should be present depending on the user state. For instance, user should not be able to access the login page and signup page when they are logged in. That is why those pages are missing on the navigation bar after user logged in. The following sections will explain the list of pages and UIs in this tab. For all pages, there is a page header, navigation bar, and footer.

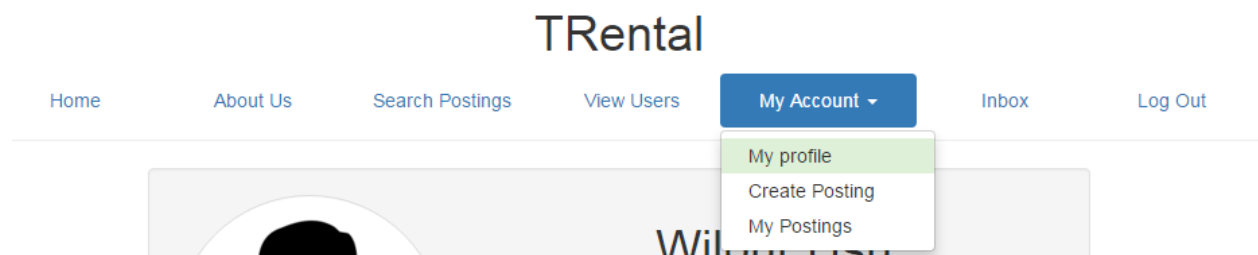
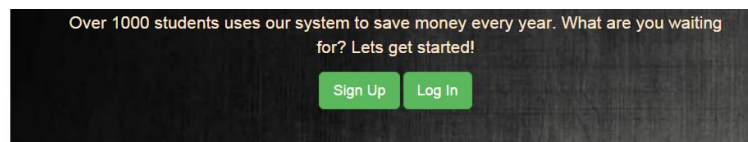


Figure 1 Navigation bar

### Before login

**Home:** In the home page, user can see brief descriptions of our website and buttons that links to the login and signup page.



### What do we *provide*?

Efficiency	Variety	Satisfiability
With TRental, you can find books you want to rent quickly, for low costs.	TRental contains a database of textbooks provided by students all over the world. We	With TRental's book rating system, users guaranteed to find a textbook that is not o

Figure 2 Home page

**About Us:** In this page, user can see brief descriptions about us and our goals

## Read more, buy less

We are a group of web developers who seek a textbook sharing platform that every student can benefit from. Students often purchase textbooks at high prices, but end up leaving them unused on bookshelves for ages. On the other hand, there are people who wish to use a book only for a short period of time and not have to pay full price to buy it. Our idea is to create a site for these users; book holders who are not going to touch their book for a while can lend them out to the others who need it temporarily and also make money in the process.



Figure 3 About us

**Login Page:** A page where user can logged in. The inputs are send to the server and will display error message accordingly if there is any. On failure, the page is reloaded. On success, users are redirected to their profile page. This page presents two methods for logging in. Using a local account or a 3<sup>rd</sup>-party Google account.

### Log In

LoginG+ Login with Google

©2015 TRental

Figure 4 Log in page

**Signup Page:** User creates a local account on this page, and are asked to input their username, email, password, and confirm password. The email must be unique among all local accounts. Validation are checked on both server and client side. Upon failure, the page is reloaded, or modified to display error message. On success, users are redirected to their profile page.

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

### After login:

When user is first logged in, they are redirected to the profile page. The navigation bar presents these tabs, home, about us, search postings, view users, my account, inbox, and logout. In particular, the “my account” tab is a drop down menu, there are more tabs in this menu. Depending on whether a user is admin or not, there may be different tabs in this menu.

**View User page:** On this page, users can see a table of all the users in the system and perform searching on it. Red row means that user is an admin, green row means that user is the logged in user. Clicking on a user will link to that user’s profile page.

<a href="#">Home</a>	<a href="#">About Us</a>	<a href="#">Search Postings</a>	<a href="#">View Users</a>	<a href="#">My Account ▾</a>	<a href="#">Inbox</a>	<a href="#">Log Out</a>
----------------------	--------------------------	---------------------------------	----------------------------	------------------------------	-----------------------	-------------------------

Show 

10 ▾

 entries

Search:

Username ▴ ▾	Email ▴ ▾	Account Type ▴ ▾	User Type ▴ ▾
<a href="#">admin</a>	admin@abc.com	local	admin
<a href="#">Amy</a>	amy@abc.com	local	user
<a href="#">Jen</a>	jen@abc.com	local	user
<a href="#">Wilbur Hsu</a>	phoneinshoe@gmail.com	google	user
Username	Email	Account Type	User Type

Showing 1 to 4 of 4 entries

Previous

1

Next


**Profile page:** User can see their own profile page and other user’s profile page. On this page, user sees a specific user’s profile and some additional buttons depending on the state of the user. For instance, if the logged in user is viewing other user’s profile, he would be able to see an “inbox” button, which allows message communications. This button would be missing if the user is viewing his own profile, since it makes no sense to message oneself. Other buttons such as “edit profile”, “remove user” will also appear in different situations depending on the type of the currently logged in user. For information about what each type of user can and cannot do, see the functionalities section.

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)



**Edit profile page:** User are presented an input form for modifying their personal information such as name and password. Depending on whether the account is locally created or logged in with a Google account, some fields may be unmodifiable. See functionalities section for details.

### Edit Profile



Upload a different photo...

[Choose File](#) No file chosen

**Personal info**


**Phone number:**

N/A

**Description**

no description

[Update Info](#) [Cancel](#)

 You cannot modify password and name of a Google account here. Do so on Google.

**Search Postings:** On this page, user sees a table of all textbook postings in the system and perform search queries on it. Red postings means the book is currently unavailable (book rented out already), green ones mean available. User can click on any posting to see a detail profile of the posting.

Show  entries

Search:

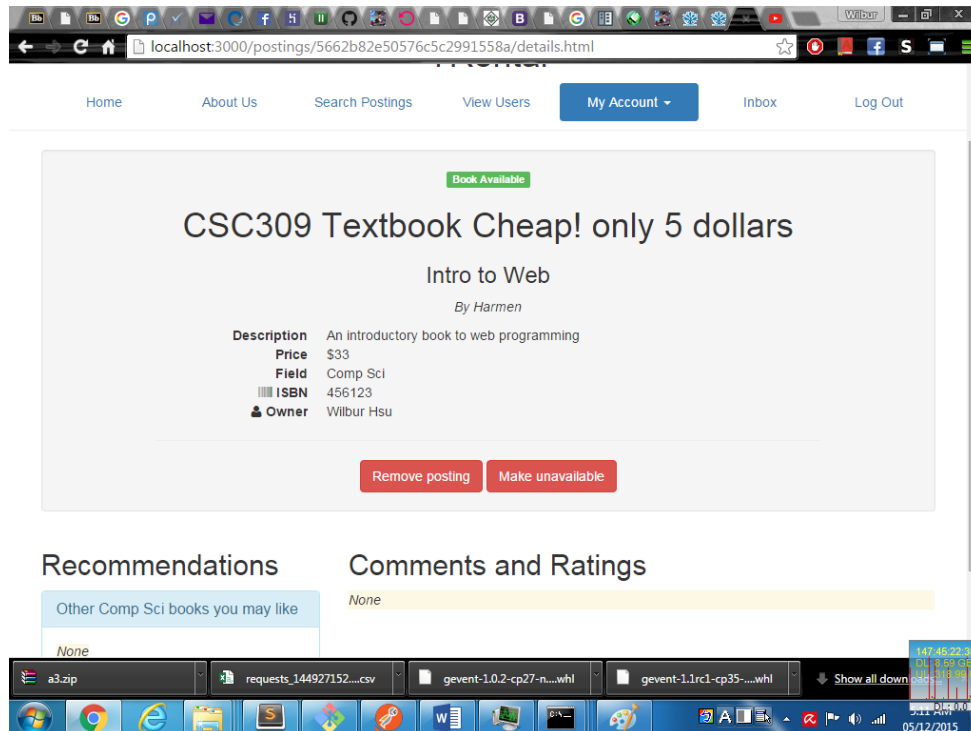
Posting Title	Book Title	Rent Price	ISBN	Owner
CSC309 Textbook Cheap! only 5 dollars	Intro to Web	\$33	456123	Wilbur Hsu
Posting Title	Book Title	Rent Price	ISBN	Owner

Showing 1 to 1 of 1 entries

[Previous](#) [1](#) [Next](#)

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

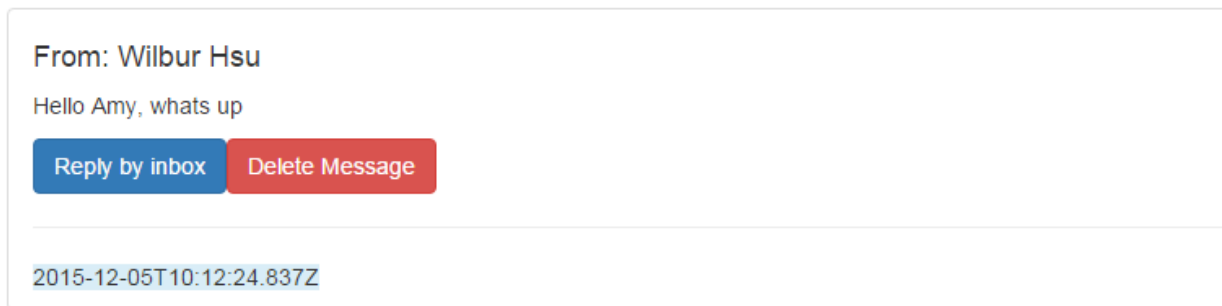
**Posting detail:** User reach this page upon clicking on a posting. On this page, user can see details about the selected posting, including the textbook information and owner details. This page also provides a recommendation section that gives links to other similar postings, and a rating section where user can create ratings or view ratings and comments from previous book borrowers. Depending on the currently logged in user type and status, there may also be additional buttons such as “remove posting” and “contact owner”. For instance, if the currently logged in user is an admin or the owner of the selected posting, he would be able to see a “remove posting” button. Additionally, the owner of the posting will not be presented with the rating input form as he cannot leave a rating for his own posting. For details about the privileges and what each type of users can do and not do, see the functionalities section.



**My posting page:** On this page, user sees a table of all the postings he has created so far. This page is the same as the search postings page, except that it limits the postings to display to the ones that belongs to the currently logged in user. Admin cannot access to this page since admin cannot make postings and would have 0 postings to display.

**Create posting page:** users can create a posting on this page. Only non-admin users can access this page since admin cannot create postings. Users are asked to input posting information such as posting title, book name and book authors. Each field must be non-empty and are validated on server side. On failure, this page is reloaded and displays error message. On success, user are redirected to their profile page.

**Inbox page:** On this page, user can see a list of all messages sent to them. User can view messages and reply to the sender.



## Functionalities:

- 1) Accounts
  - a) User can login to our website using Google. This type of account is called Google account.
  - b) User can sign up and create an account locally in our website, and login using that account. This type of account is called local account.
  - c) The first user account in the system is automatically an admin (the only admin). This account could be either local account or Google account.
- 2) Profiling
  - a) A user account has basic information such as avatar, name, phone number, and description. Every user can update their own profile. Email is not changeable.
  - b) Local accounts can modify passwords and name. Google accounts cannot. If a user wish to change his name and password of an account logged in using Google, they should do so on Google. Our website just grab those information from Google and is consistent with Google.
  - c) Every user can use a search user feature to view any other user's profile but cannot modify anyone's profile other than their own. This applies to admin as well.
  - d) Admin can delete a user in the system. Deleting a user will delete all book postings, ratings, and inbox messages associated with that user.
- 3) Social Network
  - a) Any users can communicate with any other users through a message system. Every user has an inbox containing messages.
  - b) A user can message another user by going to that user's profile and click the "inbox" button
  - c) Users can only view their own inbox
- 4) Postings
  - a) Non-admin users can create a posting indicating the textbook they want to rent. A posting represents one textbook and contains information such as book title, price and the textbook's field (of study).
  - b) At any given time a posting is either available or unavailable. The owner will be able to modify this flag on his postings to reflect the current status of the textbook.
  - c) A user can view all postings he has created and remove any one of them.
  - d) A user can search for postings based on fields such as book name, price, and authors. In the search result, unavailable postings will appear red. Available postings will appear green.
  - e) Admin can remove a posting.
  - f) A posting contains a rating sections. In this section, user can leave a rating, which contains a score and comment, about that posting. A user cannot rate a posting he created.

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

- g) In the profile page of a posting, user can view other recommended postings with similar book type.

## REST APIs

\*Some POST and PUT methods reads fields from HTTP request body. See the code for the name of these fields.

GET <http://localhost:3000/api/users>

- Returns a list of all users in the system in JSON format

GET <http://localhost:3000/api/users/:id>

- Returns a particular user with specified id in the system in JSON format

POST <http://localhost:3000/api/users>

- Create a user
- Information about the user is passed as fields in HTTP request body in x-www-urlencoded format
- Return the user created in JSON format

PUT <http://localhost:3000/api/users/:id>

- Updates a particular user with specified id
- Information about the user is passed as fields in HTTP request body in x-www-urlencoded format
- Return the modified user created in JSON format

DELETE <http://localhost:3000/api/users/:id>

- Delete a particular user with specified id

GET <http://localhost:3000/api/postings>

- Returns a list of all postings in the system in JSON format

GET <http://localhost:3000/api/postings/:id>

- Returns a particular posting with specified id in the system in JSON format

POST <http://localhost:3000/api/postings>

- Create a posting in the system
- Information about the posting is passed as fields in HTTP request body in x-www-urlencoded format
- Return the posting created in JSON format

PUT <http://localhost:3000/api/postings/:id>

- Updates a particular posting with specified id
- Information about the postings is passed as fields in HTTP request body in x-www-urlencoded format
- Return the modified posting created in JSON format



By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

DELETE <http://localhost:3000/api/postings/:id>

- Delete a particular posting with specified id

GET <http://localhost:3000/api/postings/:pid/ratings>

- Returns a list of all ratings of a particular posting with the specified pid in the system in JSON format

GET <http://localhost:3000/api/ratings/>

- Returns all ratings in the system in JSON format

POST <http://localhost:3000/api/postings/:pid/ratings>

- Create a rating for the posting with the specified pid in the system
- Information about the rating is passed as fields in HTTP request body in x-www-urlencoded format
- Return the rating created in JSON format

DELETE <http://localhost:3000/api/postings/:pid/ratings/:rid>

- Delete the rating with the specified rid from the posting with the specified pid

GET <http://localhost:3000/api/messages>

- Returns a list of all messages in the system in JSON format

GET <http://localhost:3000/api/messages/:id>

- Returns a particular message with specified id in the system in JSON format

POST <http://localhost:3000/api/messages>

- Create a message in the system
- Information about the message is passed as fields in HTTP request body in x-www-urlencoded format
- Return the message created in JSON format

DELETE <http://localhost:3000/api/messages/:id>

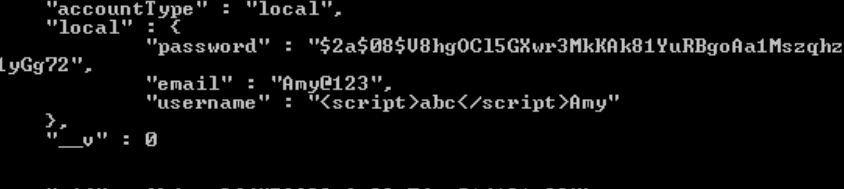
- Delete a particular message with specified id

## Web Security

The two security aspects that we specifically focused on are XSS prevention and authentication. For authentication, we used a node module called “bcrypt” to provide a one-way-encryption for the passwords. User passwords are hashed and salted before storing into our database, as shown in the picture. This way, if our database files are leaked, user account will not be exposed. When user login and enters their password, we simply use the same hash function to hash their password input and compare with the entries stored in database. If it is a match, then authentication is successful. This way, decryption is not needed and the one-

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

way-encryption provides stable security



The screenshot shows a MongoDB command prompt window with the following JSON document displayed:

```

{
  "accountType": "local",
  "local": {
    "password": "$2a$08$U8hgOC15GKwr3MkKAK81YuRBgoAa1MsqzhzcBwdmuxw7sWT1yGg72",
    "email": "Amy@123",
    "username": "<script>abc</script>Amy"
  },
  "_v": 0
},
{
  "_id": ObjectId("56622cfc99e7dec014131e92"),
  "phone": "N/A",
  "description": "no description",
  "imgPath": "/avatar/default.png",
  "userType": "user",
  "accountType": "local",
  "local": {
    "password": "$2a$08$nx1jAChPh.yjexxPQ9Km2ne2Evc54Z3UzpzivT.QdKRfWxML6n0P.O",
    "email": "Cindy@Cindy.com",
    "username": "Cindy"
  },
  "_v": 0
}

```

The document contains two user records. The second record, for user 'Cindy', is highlighted with a red rectangle. The password field in this record is highlighted with a red circle.

We understand that users will not always enter valid data in input forms, therefore we validate form data on server side and prevents XSS attacks. We used a module called “sanitizer” to sanitize and detect invalid data if any and display corresponding error messages. For example, using sanitizer, users cannot enter malicious inputs such as “<script>evilscrip here</script>” anymore on forms.

The diagram illustrates a security vulnerability in a web application's sign-up process. On the left, a user submits a sign-up form with the following data:

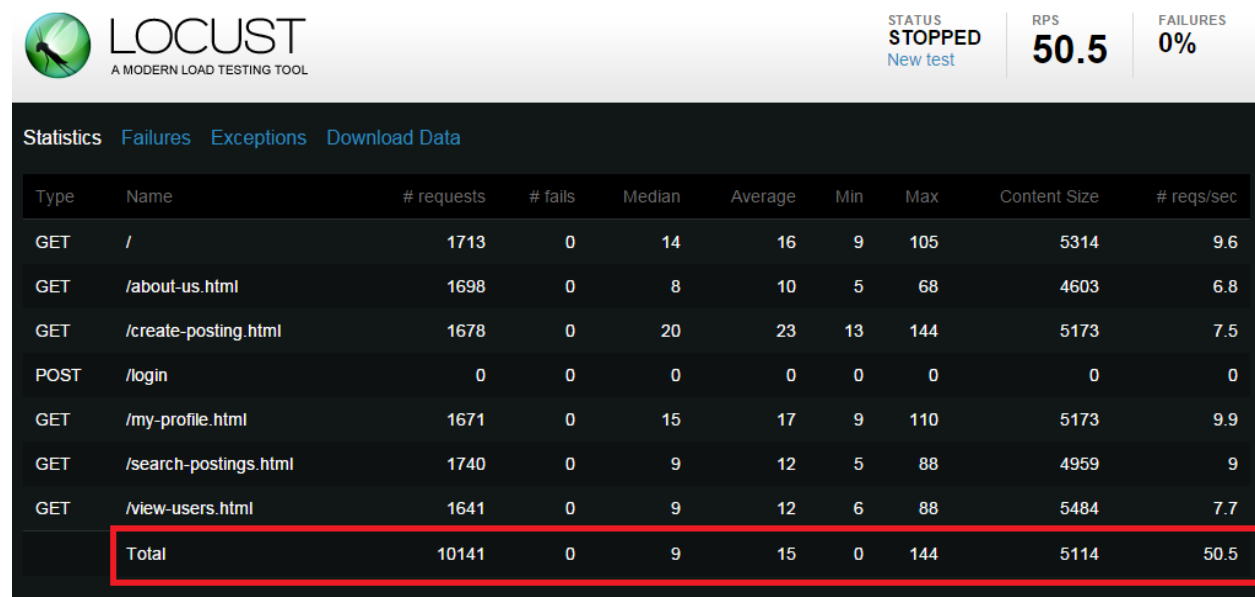
- Username: `<script>abc</script>Amy`
- Email: `amy@abc.com`
- Password 1: `...`
- Password 2: `...`

A red arrow indicates the flow of the process to the right, where the application's response is shown. The response indicates that the input is invalid, specifically highlighting the first password field with a red border and a red message box stating "Invalid input".

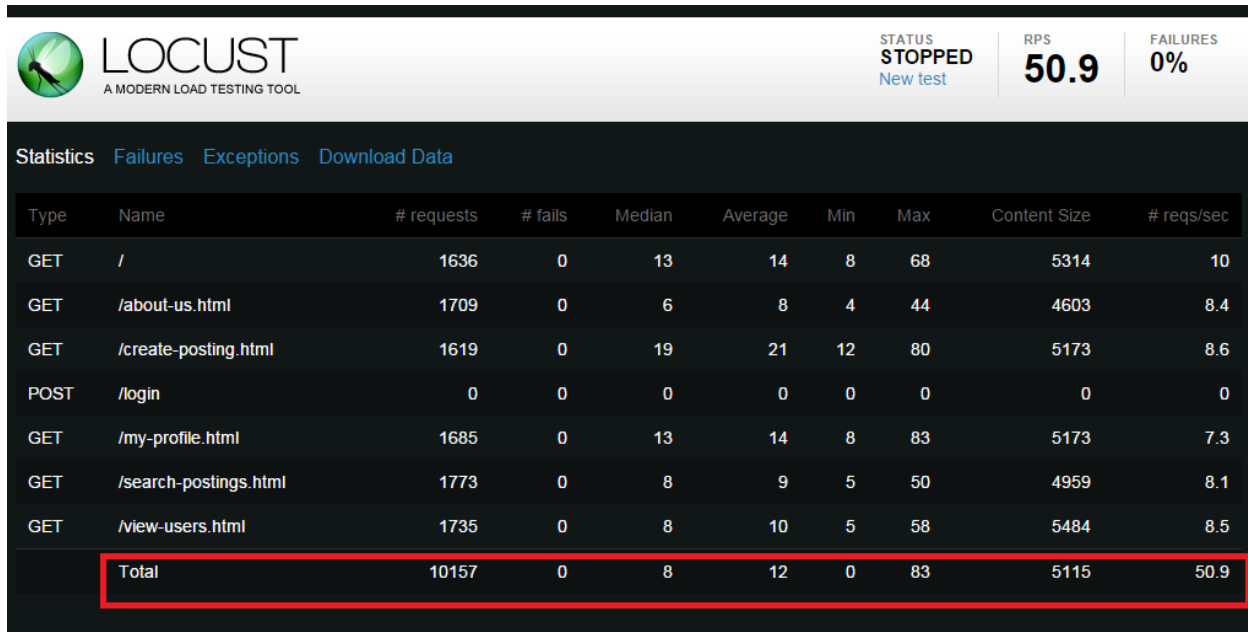
## Performance

By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)

We use mainly two techniques to optimize our website performance. We try to spawn multiple threads for handling server request and make use of the multiple cores architecture in modern machine. We achieve this by using the “cluster” module, and spawn as many worker threads as possible. The other technique we use is caching. Specifically, we cache the static files, CSS, Image, and JavaScript etc. We have also tried to compress the server response as gzip to minimize the size of HTTP response, but it seems that our files are too small to be compressed and using compression actually gives more overheads than performance improvement. Therefore we decided not to use it. Below shows the comparative result before and after optimization. We test our performance using Locustio, a load testing tool. In this test, we chose some webpages, and have 100 users accessing them repeatedly. Each users sends a request to each of those pages a few times every seconds, we waited until there the server received about 10100 requests in total before we stop the simulation. The first picture is the performance result before optimization. The second one shows the result after optimization.



By: c4hsuwei (Wei-Bo Hsu), g5yimii (YiMing Han), g4shumja (Jacky Shum), g2leslau (Leslie Lau)



The image shows the Locust web interface. At the top, there's a header with the Locust logo (a green circle with a black locust) and the text "LOCUST A MODERN LOAD TESTING TOOL". To the right of the header, there are three status indicators: "STATUS STOPPED New test", "RPS 50.9", and "FAILURES 0%". Below the header, there are four tabs: "Statistics", "Failures", "Exceptions", and "Download Data". The "Statistics" tab is selected. Below the tabs is a table with the following columns: Type, Name, # requests, # fails, Median, Average, Min, Max, Content Size, and # reqs/sec. The table contains seven rows of data for different endpoints, and a final row for the total statistics. The total row is highlighted with a red border.

Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	1636	0	13	14	8	68	5314	10
GET	/about-us.html	1709	0	6	8	4	44	4603	8.4
GET	/create-posting.html	1619	0	19	21	12	80	5173	8.6
POST	/login	0	0	0	0	0	0	0	0
GET	/my-profile.html	1685	0	13	14	8	83	5173	7.3
GET	/search-postings.html	1773	0	8	9	5	50	4959	8.1
GET	/view-users.html	1735	0	8	10	5	58	5484	8.5
Total		10157	0	8	12	0	83	5115	50.9

Notice the last row, which indicates the total statistics from loading the different webpages. As can be seen, after optimization, we have increase the average response time by 3 ms, from 15 ms to 12 ms. This is a great performance improvement.

## Testing

We are using the modules “mocha”, “superagent” and “assert” to test our website. We tested some basic APIs, and route endpoints. More descriptions in test.js

## Video

For video and demonstration, watch the video here: <https://www.youtube.com/watch?v=YnnQrrcLZtw>

## GitHub Repo

<https://github.com/yiminghan/CSC309A5>