DESIGN DOCUMENT PART 2

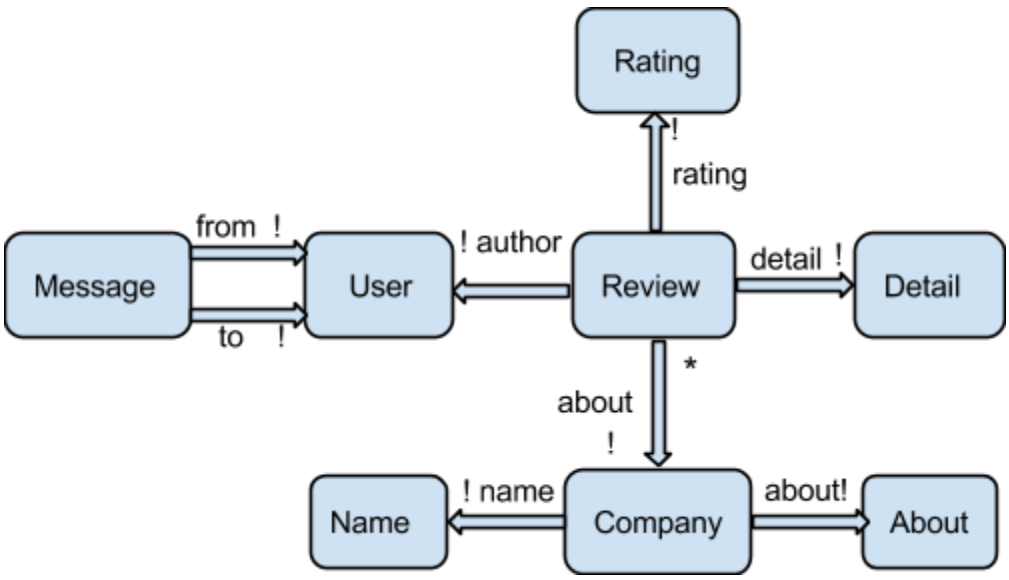
Feature Implementation Authors:
Zulsar: Users & messages
Kapaya: Companies
Yee Ling: Reviews

COMMENT for Testing: Our tests could be found in http://tim-kkatongo.rhcloud.com/tests
Our test files can be found in the public/qunit directory

Changes from Part 1:

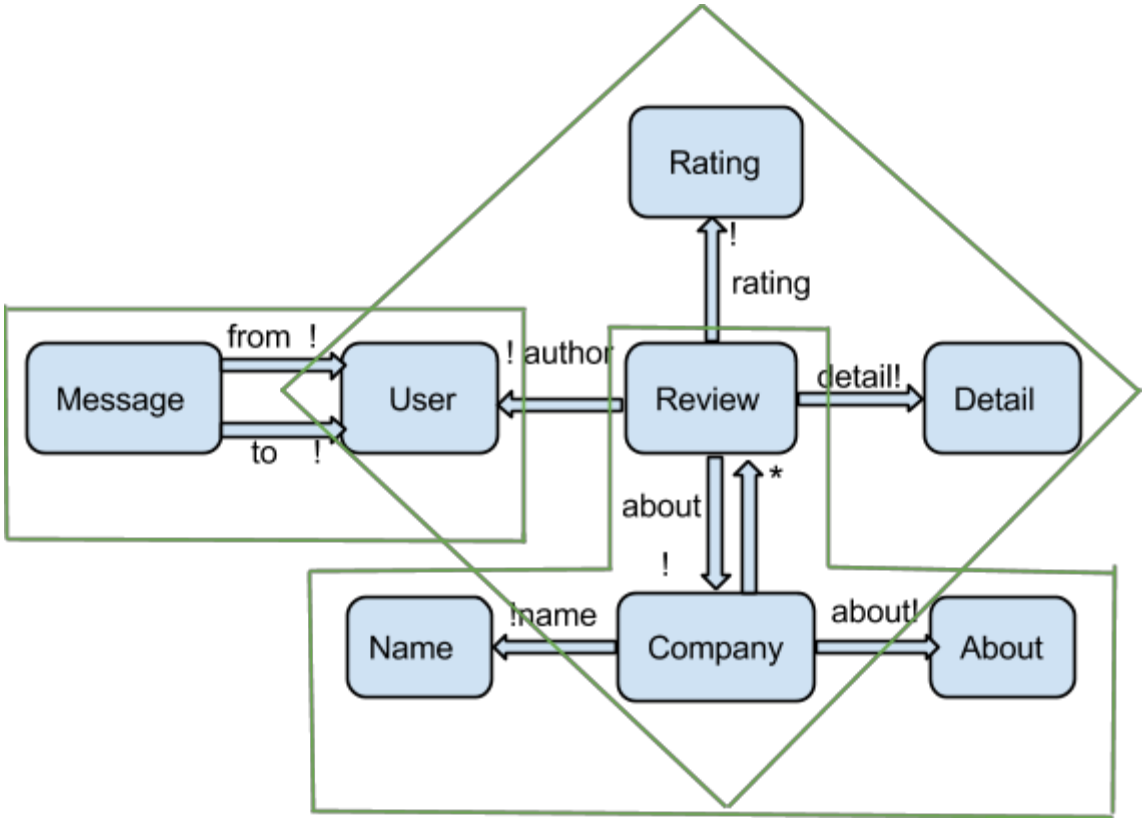
- **Concepts from Part 1 that were not implemented for our current MVP, but will be implemented for Part3:**
  - Question
  - Answer
  - Voting (upvote and downvote)
  - **Q&A implementation will be done in part 3
- Our concept of **beaver call** is implemented in the form of the internal messaging system
- Our concept of the **toolbox** is currently implemented in the form of 'details' under each review
- Our idea of keeping TIM in a Suit exclusive to the MIT community through validation that users' emails have the form '@mit.edu' will be implemented in Part 3
- We will change from using express-sessions for authentication to using Passport.js for Part 3
- For Part 2, we disallow the option to delete a user, we will implement this in Part 3

Data Model



A user cannot send themselves a message.
A user can post at most one review per company.

Transformed Model



Collections:
Users: {username: String, password: String}
Messages: {from: Object_id(Users), to: Object_id(Users), content: String}
Companies:{name: String , about: String, ratings: Number, reviews: Object_id(Reviews)}
Reviews:{user: Object_id(Users), company: Object_id(Companies), rating: Number, details: String}

Data design choices + justification:

- The most updated items are Reviews, Messages, Companies, Users.
  (We haven't implemented some of our features in this phase yet.)
- Therefore we have a collection for each of them that would make it easier to access to update them individually.
- We have two separate collections Companies and Reviews because a Review contains ratings and details, which can be updated separately from Company. Companies has a list of Review, which is updated whenever a Review is created / deleted to make searching for particular companies much easier. Otherwise, whenever we have to get

the reviews, we would have to query by the company. Now, the company would keep track of its own reviews and it would be much easier to get the reviews.

- For the Review object we decided to put all information such as interview/application details into one Detail object instead of having many attributes with all sub-details. Therefore, when we want to search for all details to display in the user interface, we would not have to search for multiple collections. Furthermore, it provides a nice organization.
- To support our messaging feature, we added the Message Object with from, to and content attributes and instead of Users keeping track of their own messages in two attributes with a list of messages, querying by Message Object helps finding particular user's messages. Otherwise, we would have to update the user, everytime we add/edit/delete a new message. Similarly, User do not track his/her own reviews.
- We didn't choose a deeply nested collection structure because we think that it is easier to access different collections to update a particular item rather than access deeply nested items. For example, we could have implemented Companies and Reviews as one collection with a company having a list of review item and each review item contains the rating and details. To update the rating or details in this nested structure, we would have to access the company, iterate through its list of reviews to find the right one and update the rating or details within.

Design Challenges

1. When to calculate the ratings for a company?

Options:

(a) Calculate when the ratings is requested
   i. Pros: No potential concurrency issues if two users try to post a review at the same time
   ii. Cons: If there are many reviews, could take a noticeable amount of time to calculate and return the ratings

(b) Store a company's ratings in the database, and update whenever a review is added/edited/deleted
   i. Pros: Saves time as we don't need to recalculate for every query that involves the company rating.
   ii. Cons: Concurrency issue is possible, especially since we are using MongoDB which doesn't guarantee that only one person can edit the database at one time.

Final choice:

(a), to calculate whenever we need the ratings. This is because we think that concurrency would be a bigger issue than time for calculation.

2. Whether to populate the reviews (through (path: user and company), when returning a JSON object from company query? * a company stores a list of reviews

Options:

(a) Populate the reviews

i. Pros: No need for extra calls to API to find the relevant reviews, especially since we would query a company to render all the company's info and all its related reviews on a single HTML page
ii. Cons: Potentially a large and complicated/nested JSON object will have to be returned

(b) Don't populate them, just return as a list of review ids.
i. Pros: Don't need to send a potentially large JSON object
ii. Cons: If we wanted to render any of the reviews, we would have to make extra calls to the API with the review ID. Given that in the long run, a company may have many reviews, this choice could lead to a noticeable lag in rendering the page.

Final Choice:
If a user makes a company query for a single company , the API returns a JSON object with the reviews all populated. If a user makes a company query to return all the company, the API will return a JSON object without populating the reviews. We chose this solution because for Part 3, we would probably use a single company query to render a full page with company information + reviews. We would probably use the full company query to render an index page with all the companies currently in the database.

3. Whether or not to allow a logged in user to create a new user?
Options:

(a) Yes, allow logged in user to create a new user.
i. Pros: Easier to implement / more relaxed rules
ii. Cons: You allow duplicate users

(b) No, don't allow a logged in user to create a new user.
i. Pros: Simpler authentication method, as new users are immediately authenticated.
ii. Cons: Stricter rules on users, more work to implement

Final choice:
(b), do not allow a logged in user to create a new user. This is because when a new user is created, we automatically authenticate the new user using their new information. So, if we were to allow a logged in user to create a new user, then there would be problem of two authentications at that point in time, which would complicate the concept of an authenticated user (especially during POSTing when we use a user's authenticated id).

4. Getting individual users review via users/:userId/reviews or reviews/:userid?
Options:

(a) users/:userId/reviews
i. Pros: A user can get their own reviews easily.
ii. Cons: It may be difficult for a user to search for other users' reviews

(b) reviews/:userid
i. Pros: Any user can obtain any other user's reviews easily.
ii. Cons: Not specific to a user, and it makes less sense than users/:userId/reviews

Final choice:
(a) users/:userId/reviews: Because in TIM in a Suit, we would let the user have the option to see only the review that they have created. The other option would be showing a user all the reviews for a particular company, which disallows the user to search for reviews made by another specific user. Also, users/:userId/reviews, is more intuitive and makes it hard for a different person to get the reviews of a different person unless they know the :userId and are logged in as the other user.

Choices from Design Challenges in Part 1:

- *Allowing anonymity, but implementing an internal messaging system*
- *A middle ground solution to setup an internal user messaging service that notifies a user via email whenever they receive a new message / email*
- *Using a built-in email validator and check that it is the form of @mit.edu*
- *Only MIT students are allowed on TIM in a Suit*
- *5-Star Scale*

API documentation

For all Request Actions, we assume that a user is logged in, otherwise, the user will be redirected to the login. Currently, since we don't have any views, we're rendering any HTML page in our methods, instead, we're returning JSON objects with appropriate messages. **Note the example response is in *italics.*

Methods for Users:

GET /users/logout
Request parameters: empty
Request body: empty
Response:

- upon successful logout: a status 200 with JSON object sent back to the client with "Succesfully logged out!" message and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "message": "Succesfully logged out!",
   "success": true
}
```

GET /users/:userId
Request parameters:
{ userId: "544320577a3be99b1a229459" }
Request body: empty
Response:

- upon successful completion: a status 200 with JSON object sent back to the client, with formatted User object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
   "message": {“username” : “Zulaa”},
   "success": true
}
```


GET /users/:userId/reviews

Request parameters:

{ userId: “544320577a3be99b1a229459” }

Request body: empty

Response:

- upon successful retrieval of the reviews, a status 200 with JSON object sent back to the client, with formatted User object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
   "message": [
      {
         "_id": "5447162519ea4c73c1734a2b",
         "user": "Zulaa",
         "company": "Facebook",
         "rating": 5,
         "details": "awesome!"
      }
   ],
   "success": true
}
```


GET /users/:userId/reviews/:reviewId

Request parameters:

{ userId: “544320577a3be99b1a229459”, reviewId:“544320707a3be99b1a22945a” }

Request body: empty

Response:

- if successfully retrieved user’s review from the database: a status 200 with JSON object sent back to the client, with formatted Review object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
   "message": {
```

```
        "_id": "5447169519ea4c73c1734a2c",
        "user": "Katongo",
        "company": "Microsoft",
        "rating": 4,
        "details": "ok!"
    },
    "success": true
}
```


POST /users/
Request parameters: empty
Request body:
{ username: “Zulaa”, password: “11” }
Response:

- upon successful creation of a new user: a status 200 with JSON object sent back to the client, with formatted User object and success: true, otherwise rejects and sends JSON object with an error message with success: false.


```
{
    "message": "Home Page",
    "success": true
}
```


POST /users/login
Request parameters: empty
Request body:
{ username: “Zulaa”, password: “11” }
Response:

- upon successful login, a status 200 with JSON object sent back to the client, with appropriate message and success: true, otherwise rejects and sends JSON object with error message with success: false.


```
{
    "message": "Succesfully logged in!",
    "success": true
}
```

Methods for Messages:
GET /messages
Request parameters: empty
Request body: empty
Response:

- upon succesful retrieval of the messages, a status 200 with JSON object sent back to the client, with formatted Message Objects and success: true, assumes a user is

logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "message": {
      ["_id": "5447169519ea4c73c1734a2c",
      "from": "Zulaa",
      "to": "Kapaya",
      “content”: “Hi Kapaya :)”,
       "_id": "8447169519ea4c73c1784a28",
      "from": "Zulaa",
      "to": "Elaine",
      “content”: “Hi Yee Ling :)”]
   },
   "success": true
}
```

GET /messages/:messageId

Request parameters:

{ messageId: “544320577a3be99b1a229459” }

Request body: empty

Response:

- upon successful retrieval of a message: a status 200 with JSON object sent back to the client, with the formatted Message Object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "message": {
      "_id": "5447169519ea4c73c1734a2c",
      "from": "Zulaa",
      "to": "Kapaya",
      “content”: “Hi Kapaya :)”
   },
   "success": true
}
```

POST /messages

Request parameters: empty

Request body:

{ to: “Zulaa”, content: “Hi :)” }

Response:

- upon successful creation of a new message, a status 200 with JSON object sent back to the client, with a list of formatted Message Objects and success: true, assumes a

user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "message": {
      "_id": "5447169519ea4c73c1734a2c",
      "from": "Zulaa",
      "to": "Kapaya",
      “content”: “Hi Kapaya :)”
   },
   "success": true
}
```

DELETE /messages/:messageId
Request parameters:
{ messageId: “544320577a3be99b1a229459” }
Request body: empty
Response:

- upon successful completion, sends a JSON object back to the client: with the formatted Message Object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "error": "Message does not exist.",
   "success": true
}
```

Methods for Companies:

GET '/companies'
Request body: empty
Request parameters: empty
Response:

- if server succeeds in getting all companies, sends a JSON object back to the client: with a list of formatted Company Objects (which is all the companies available, each company object has 4 fields: name, about, rating, reviews) and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "success": true,
   "message": [
      {
         "_id": "5446fdcb352249dbbed88463",
```

```
        "name": "Akamai",
        "about": "Started by Tom Leighton",
        "rating": 0,
        "reviews": []
      }
   ]
}
```


POST '/companies'
Request parameters: empty
Request body (all required):
{ name: “Palantir”, about: “Awesome!” }
Response:

- if server succeeds in creating a new company, sends a JSON object back to the client: with a list of formatted company object (company object has 4 fields: name, about, rating, reviews) and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
   "success": true,
   "message": {
      "_id": "5447149619ea4c73c1734a27",
      "name": "Microsoft",
      "about": "about",
      "rating": 0,
      "reviews": []
   }
}
```


GET '/companies/:id'
Request parameters:
{ id: “544320577a3be99b1a229459” }
Request body: empty
Response:

- if server succeeds in getting the company, sends a JSON object back to the client: with the formatted company object (company object has 4 fields: name, about, rating, reviews) and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
   "success": true,
   "message": {
      "_id": "5446fdcb352249dbbed88463",
      "name": "Facebook",
```

```
    "about": "Started by Mark Zuckerberg",
    "rating": 5,
    "reviews": []
  }
}
```

Methods for Reviews:

GET '/reviews'

Request body: empty

Request parameters: empty

Response:

- if server succeeds in getting all the reviews, a status 200 with a JSON object back to the client with a list of formatted review objects and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
  "success": true,
  "message": [
    {
      "_id": "5447162519ea4c73c1734a2b",
      "user": "elaine",
      "company": "Facebook",
      "rating": 5,
      "details": "awesome!"
    },
    {
      "_id": "5447169519ea4c73c1734a2c",
      "user": "elaine",
      "company": "Microsoft",
      "rating": 4,
      "details": "ok!"
    }
  ]
}
```


POST '/reviews'

Request body (all required):

{ company: “Facebook”, rating: “5”, details: “The interviewer was really nice :).” }

Request parameters: empty

Response:

- if server succeeds in creating a new review, a status 200 with a JSON object back to the client with the formatted review object and success: true, assumes a user is logged in

with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "success": true,
   "message": {
      "_id": "5447153419ea4c73c1734a2a",
      "user": "elaine",
      "company": "Facebook",
      "rating": 5,
      "details": "awesome!"
   }
}
```

PUT "/reviews/:id"

Request parameters:

{ id: “544320577a3be99b1a229459” }

Request body:

{ rating: 5, details: “The interview was ...” }

Response:

- if server succeeds in getting the review, a status 200 with a JSON object back to the client with the formatted review object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
   "success": true,
   "message": {
       "_id": "5447153419ea4c73c1734a2a",
       "user": "elaine",
       "company": "Facebook",
       "rating": 4,
       "details": "cool!"
   }
}
```

DELETE '/reviews/:id'

Request parameters:

{ id: “544320577a3be99b1a229459” }

Request body: empty

Response:

- if server succeeds in deleting the review, a status 200 with a JSON object back to the client with the review id and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.

```
{
```

```
   "success": true,
   "message": "5447153419ea4c73c1734a2a"
}
```


GET '/reviews/:id'

Request parameters:

{ id: “544320577a3be99b1a229459” }

Request body: empty

Response:

- if server succeeds in getting the review, a status 200 with a JSON object back to the client with the review object and success: true, assumes a user is logged in with an active session, otherwise rejects and sends JSON object with an error message with success: false.


```
{
   "success": true,
   "message": {
       "_id": "5447162519ea4c73c1734a2b",
       "user": "elaine",
       "company": "Facebook",
       "rating": 5,
       "details": "awesome!"
   }
}
```