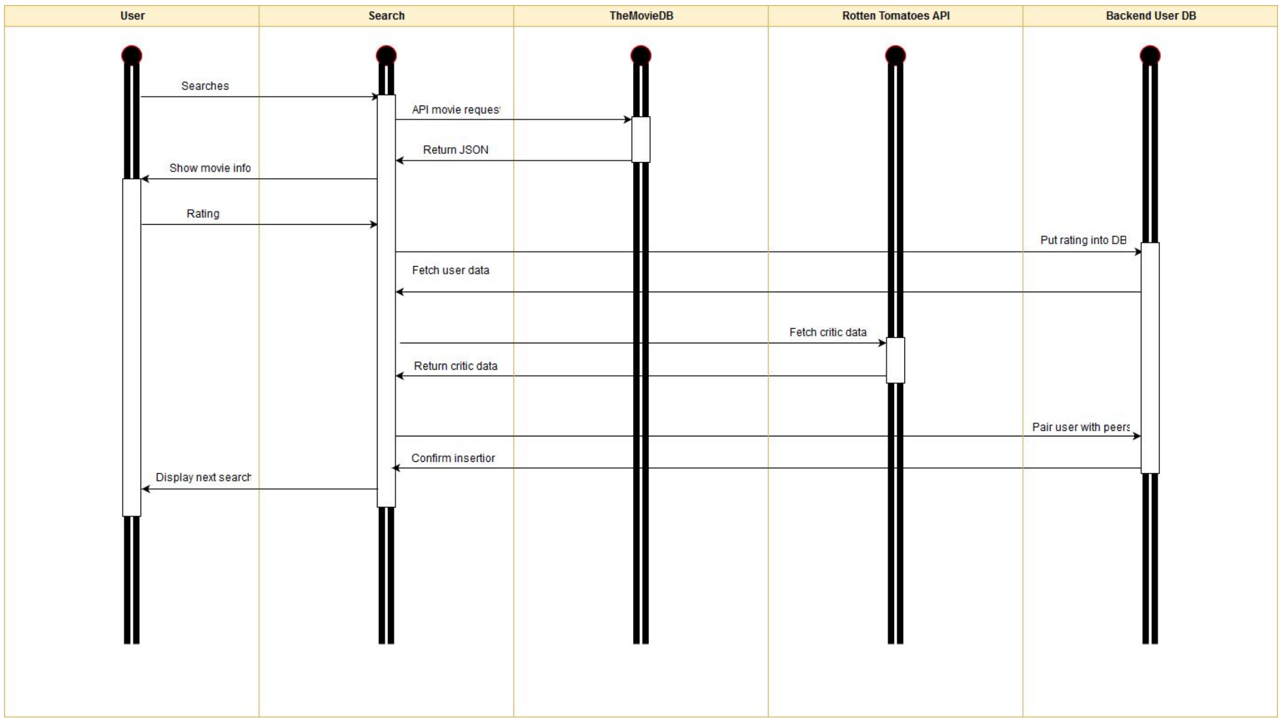
Part 1 (picture in Git repo):



Part 2 (happy path):

1. User searches using the search page
2. Search sends API movie request to the movie database
3. Movie database returns JSON to the search page
4. Search page shows the movie information to the user
5. User enters the movie rating onto the search page
6. Search page enters movie rating into our database
7. Our database sends the user data to the search page
8. Search page fetches critic data from Rotten Tomatoes
9. Rotten Tomatoes returns the critic data to the search page
10. Search page sends the critic data to our database and pairs user with peers
11. Our database sends confirmation to search page
12. Search page displays next search to users

Possible Exceptions:

If the movie the user searches for doesn't exist:

1. Movie DB sends error to search: “Movie does not exist”

If critic data doesn't exist:

1. Rotten Tomatoes sends error to search: “Movie critic data not available”

Part 3 (architecture choices):
Back end: Python and Django
Front end: Responsive HTML5, Angular.js
Database: MySQL

We chose Django and Python because our team members are most familiar with Python, which is the basis for Django. We chose Angular.js because it works well with Django, and because an important aspect of our program is seamless transition between different pages. We chose MySQL for our database because our project calls for a relational database, and again because it operates well alongside the Django framework.

Our team did not choose Node.js; from previous experience with Node, we believe that learning it would add too much upfront cost for applying it. Also, Node.js is best for apps with lots of asynchronous interactions between the user and the server, whereas our app represents few interactions between the user and server with several complicated algorithms running in the backend.