**CS673S16 Software Engineering** 

**Team 2 - MeetCute**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Daniel [Shih](mailto:Shih,shih0623@bu.edu) | Team Leader | *Huai Chun Shih* | Oct. 24, 2016 |
| Rebecca Bell | Env. & Integ. Lead | *Rebecca Bell* | Oct. 24, 2016 |
| Gabriel Leake | Requirement Leader | *Gabriel Leake* | Oct. 24, 2016 |
| Gabriel Leake | Security Leader | *Gabriel Leake* | Oct. 24, 2016 |
| Ian Klarman | QA Leader | *Ian Klarman* | Oct. 24, 2016 |
| Adiba Nisa | Design Leader | *Adiba Nisa* | Oct. 24, 2016 |
| Gautam Bhat | Configuration Lead | *Gautam Bhat* | Oct. 24, 2016 |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| ０．１ | Gabriel Leake | 11-6-2016 | Added intro, updated software architecture |
|  |  |  |  |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_buttcq9i221r)

[Design Patterns](#_x18fj36s1121)

[Key Algorithms](#_mtfbusfb0eq3)

[Classes and Methods](#_7ucksmkf6rzx)

[References](#_15tmymhipvdv)

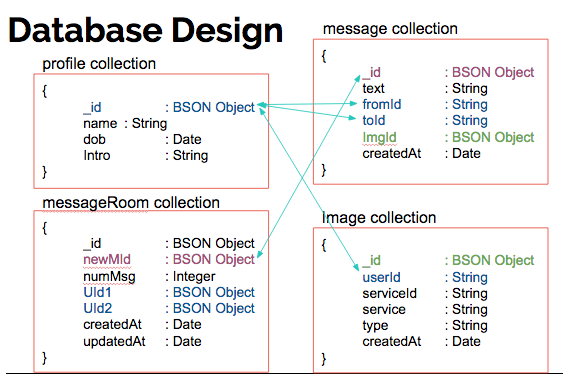
[Glossary](#_8n34lvocupub)

# Introduction

This document will lay out the architecture, design patterns, algorithms, and classes/methods of our Meteor dating application, which is called MeetCute. A primary goal of MeetCute is to provide a performant system.

# Software Architecture

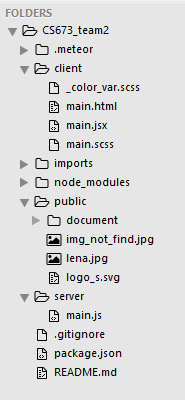
In this section, you will describe the decomposition of your software system, which include each component (which may be in terms of package or folder) and the relationship between components. You shall have a diagram to show the whole architecture, and class diagram for each component. The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too. Database design should also be described if used.



Controller = control.js

Each jsx file in the ui/components folder is a Component - i.e. it extends the Component class. These will all be imported so that their html can be displayed by a particular page.

The jsx files in the pages folder import all of the “components” - i.e. Discover.jsx imports profile\_card.jsx.



In most meteor applications there is a client and server folder - in order to separate the code that needs to run on each.

mports/

startup/

client/

index.js # import client startup through a single index entry point

routes.js # set up all routes in the app

useraccounts-configuration.js # configure login templates

server/

fixtures.js # fill the DB with example data on startup

index.js # import server startup through a single index entry point

api/

<feature>/ # a unit of domain logic,

# that're Discover, message, profile, landing\_page

server/

publications.js # all list-related publications

publications.tests.js # tests for the list publications

<feature>.js # definition of the Lists collection

<feature>.tests.js # tests for the behavior of that collection

methods.js # methods related to lists

methods.tests.js # tests for those methods

ui/

components/ # all reusable components in the application

# can be split by domain if there are many

layouts/ # wrapper components for behaviour and visuals

pages/ # entry points for rendering used by the router

client/

main.js # client entry point, imports all client code

server/

main.js # server entry point, imports all server code

ref. https://guide.meteor.com/structure.html

# Design Patterns

The Meteor.js framework is flexible enough to allow the software architecture to implement the patterns they choose. We are using a Model - View - Controller pattern. The views are comprised of nested Components that define the content of the pages, however much the information that does not require logic bypasses the controller and is retrieved directly from the database. In this way our architecture takes on aspects of the Model-View-ViewModel pattern. Events are handled in the View, however any Events resulting in a model update are handled by the Controller.

# Key Algorithms

#### This function plays an important role in our message system.

#### **\_clientSendMessage(toUserId, text)**

API for sending message to another user At this point the \_clientSendMessage works on Message.jsx because the Message.jsx is subscribing to the messagRoom publication

##### **Parameters:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| toUserId | string | The unique ID of the recipient of the message |
| text | string | The message to be sent to the recipient |

Source:  
[api/control/control.js](http://gautambhat.github.io/meetcute-docs/api_control_control.js.html), [line 17](http://gautambhat.github.io/meetcute-docs/api_control_control.js.html#line17)Example:  
\_clientGetLatestMsg("fivE7HyBekduoKe67")

**Code review:**  
First create a new message in Messages collection.   
  
Second check if the messageRoom between the two users exist or not.  
If the message room already exist, it update the message room’s attribute, “updatedAt” that show the time of the last msg, and “NumOfNewMsg” that show the unmber of unread messages.  
Else, if the message room is not exist yet. It create a new message room in MessageRooms collection.  
If anything failed, it roll-back and delete all changes made and show error.

# Classes and Methods

Description of classes and methods used can be found [here](http://gautambhat.github.io/meetcute-docs/). Documentation was generated using JSDoc.

# References

* MeetCute Doc <http://gautambhat.github.io/meetcute-docs/>
* Meteor Doc <https://docs.meteor.com/>
* File Structure <https://guide.meteor.com/structure.html>

# 