SOCIAL TRENDS

Software Requirements Specification V2.0

MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

Anil BEKTAS 1824010
Tekin ERTEKIN 1745959
Gozde GOKMEN 1678960
Bashir KAZIMI 1824523

Contents

1. Introduction

This document contains software requirements data about Social Trends and prepared in accordance with IEEE STD 830-1998 IEEE Recommended Practice for Software Requirements Specification. [1]

Main purpose of this document besides being the very first planning stage, is pointing out the requirements of the system both software-wise and hardware-wise. For this purpose we are widely using UML and its standards, tables and diagrams to depict our graphics. [2] Besides what is mentioned also, a schedule is provided applicable to our upcoming months.

1.1 Problem Definition

As a result of increasing number of Internet users in today's world, number of social-media users are also increasing. People try to solve their communication problems using social-media tools. Our project is about developing a new social-media tool that meets the needs of people's communicating in a better way. Our first aim is to establish a social-media environment that enables us more interaction. On the other hand, we want to bring in a native social-media web to our country.

The biggest disadvantage of the current social media sites is they aren't filtering data tediously. The one example that breaks down this rule might be Twitter since it depicts worldwide trends but is it still sufficient? Not yet, until we discovered to show much more variety of trends filtered by only string based like video and image.

At last we conclude that this free roaming unfiltered data is making the interactions of active users in more unclear. With our system there will be a clear interaction between the users.

Figure 1 shows the growth of the Internet users around the globe.

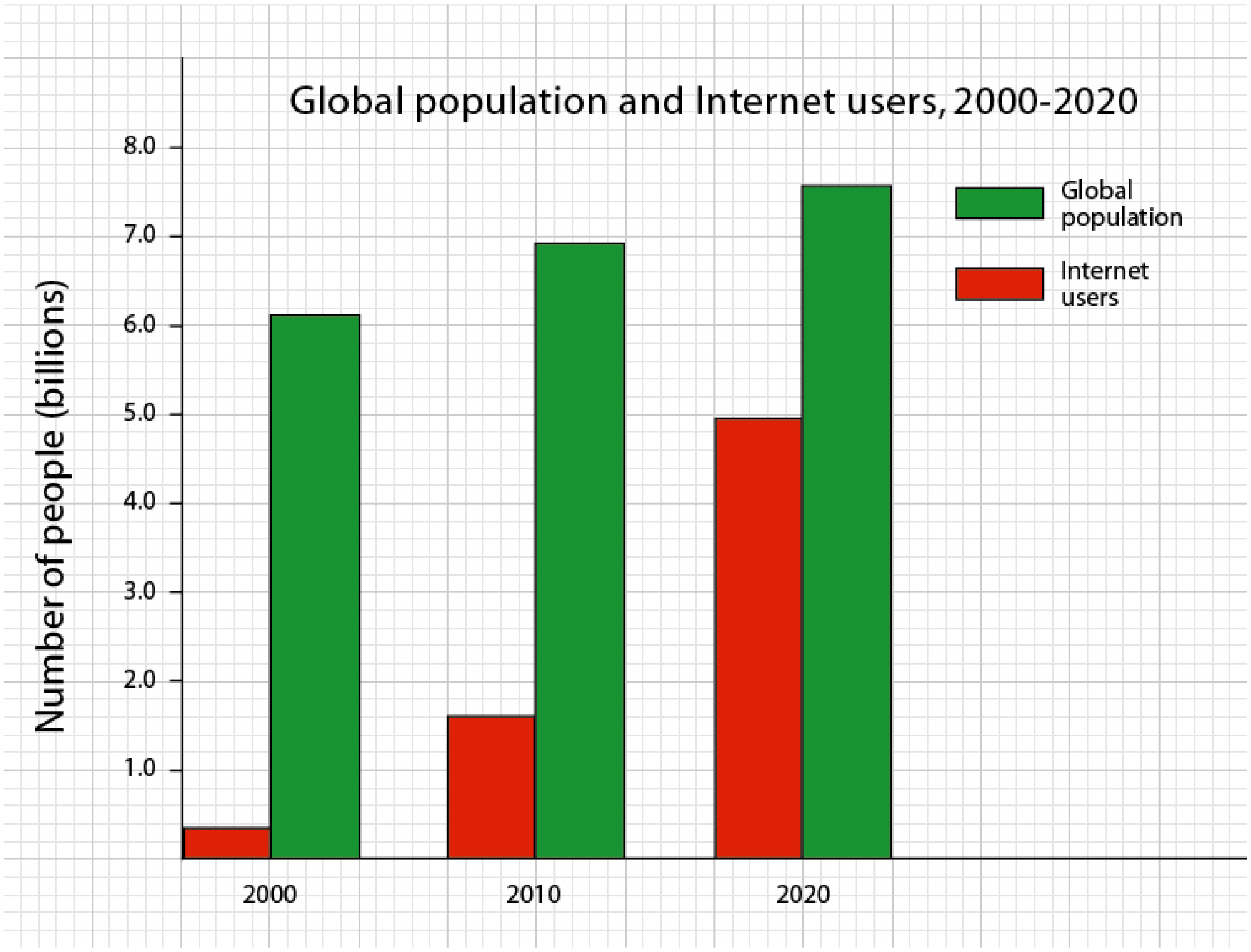


Figure 1: Internet Usage Globally

Social-media usage graphic:

Figure 2 shows the *age-Internet usage* graph

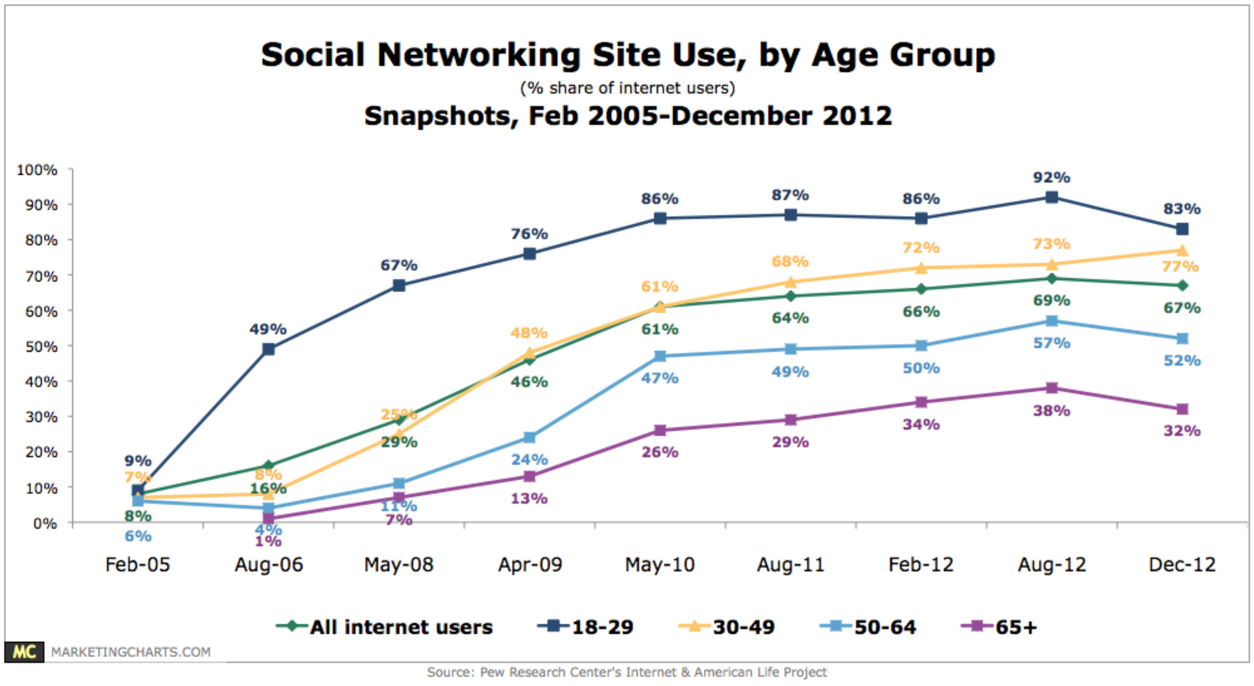


Figure 2: Social Media Statistics

1.2. Purpose

a) Our aim in preparing this SRS is to generate a document to embody our ideas with their very first necessities. This document foresees most necessities of the project while enlightening the point why this project exists and what it serves for.

b) All the students, teachers, investors who are willing to discover new funding shores, anybody who is interested in our project are encouraged to read this SRS.

1.3 Scope

a) Software part of the project will consist of three main parts: database, main program and user-interface.

b) For now, we do not need any other software program except for the mobile part of the project.

c) User-interface: Connection between the users and the system will be provided by user-interface. Besides, there can be small user-interface applications for system managers. JQuery will be used for that.

Main program: This is the part where system provides necessary information for user-interface with the database part. Also, there will be some methods for writing the information that comes from user-interface to database. For the sake of the system in the security manner, no database connections will be established in this part. Necessary information from the methods which are inside the database part. (Node.js)

Database: In this part, the information that is received from main program will be saved and necessary information will be gotten. Data interchanging can only be done from this part. In the other two parts, there will no interchanging data process.

d) System requirements will be changed according to number of the current users.

1.4. Definitions, Acronyms, and Abbreviations

Table 1 contains the definition of the terms, acronyms and abbreviations used throughout this document.

Name	Operability
User	A random person who is member of the system
System	Web-site that is defined on the web-server
Manager	People that have the authority to manage the system
Server	Server computer that is connected to internet 7/24 and has high internet connection speed

Table 1: Acronyms

1.5 Overview

This document, the Software Requirements Specification (SRS), identifies the software requirements in the form of a task and system object model. The model presented within this document is an implicit statement of the requirements. It exhibits the boundaries and capabilities of the system to be built.

This document has 8 major sections:

Executive Summary, gives a brief summary of the document contents and their purpose.

Section 1: (Introduction) provides an overview of the entire SRS document and the product being specified.

Section 2: Describe the general factors that affect the product and its requirements.

Section 3: Should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements.

Section 4: Describes information domain for the software.

Section 5: Presents a description of the behavior of the software.

Section 6: Contains team structure and estimated plan of the project.

Section 7: Briefly explains what we are going to do in this project.

Section 8: Includes the supporting information makes the SRS easier to use.

1.6. References

[1] IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

[2] For UML sources: http://www.tutorialspoint.com/uml/index.htm

2. Overall Description

2.1 Product Perspective

Our system does not work together with any other system. Our system works as an independent system itself.

2.1.1 System Interfaces

In this interface, the managers who manage the system will get information about the ongoing of the system. Information about the number of the current users, daily posts, daily messages, daily sent videos and images could be retrieved through this interface. Only authorized people could log in to this interface with their name and password.

Figure 3; the diagram depicts the interface rather system-wise. On following topics for instance user interface and some others are issued more in detail and separately.

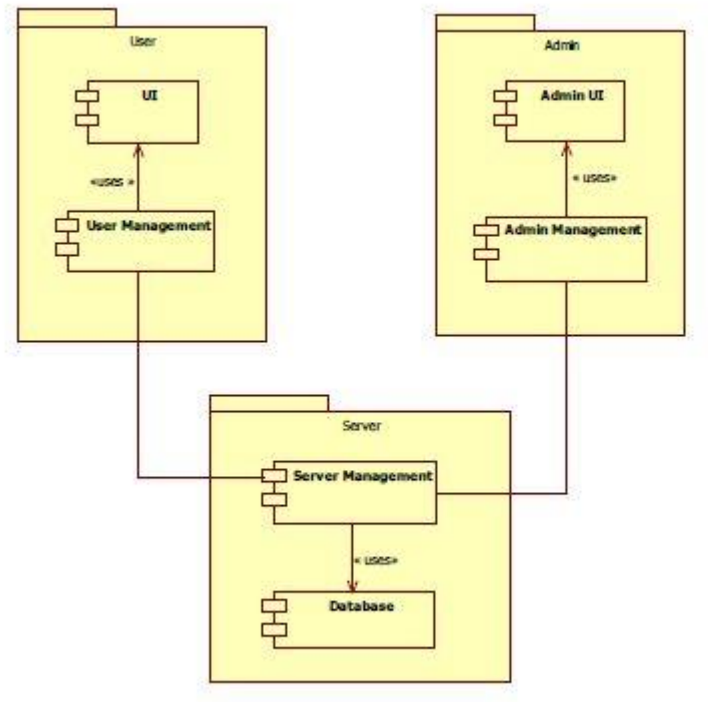


Figure 3: General Interface Structure

2.1.2 User Interfaces

This interface will be the one which will be open to the outside world. Unlike other social networks; as explained before, this system will be composed of panels. Examples to these panels are:

1. Post Panel

2. Message Panel

3. Chat Panel

4. Notification Panel

5. Trend Panels (Text, Images, Video etc.)

6. Main Menu Panel (Horizontal and Vertical)

7. Friends, Pages, Group suggestion Panels

8. Profile Panel

These profiles could be closed by the users if they want to. In addition, the user could do move, maximize, minimize and other operations like this to the panels. Thus the user members of our system could make their own design and bring the system to a suitable shape for themselves.

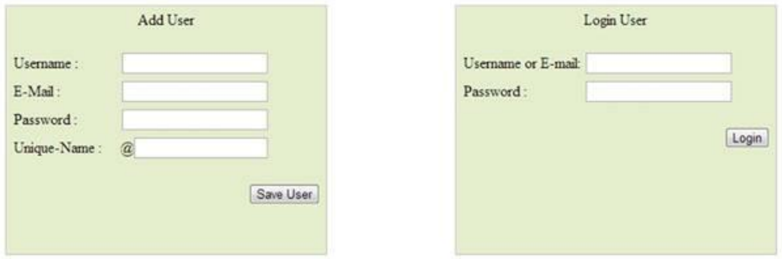


Figure 4: Sign in Sign up Page

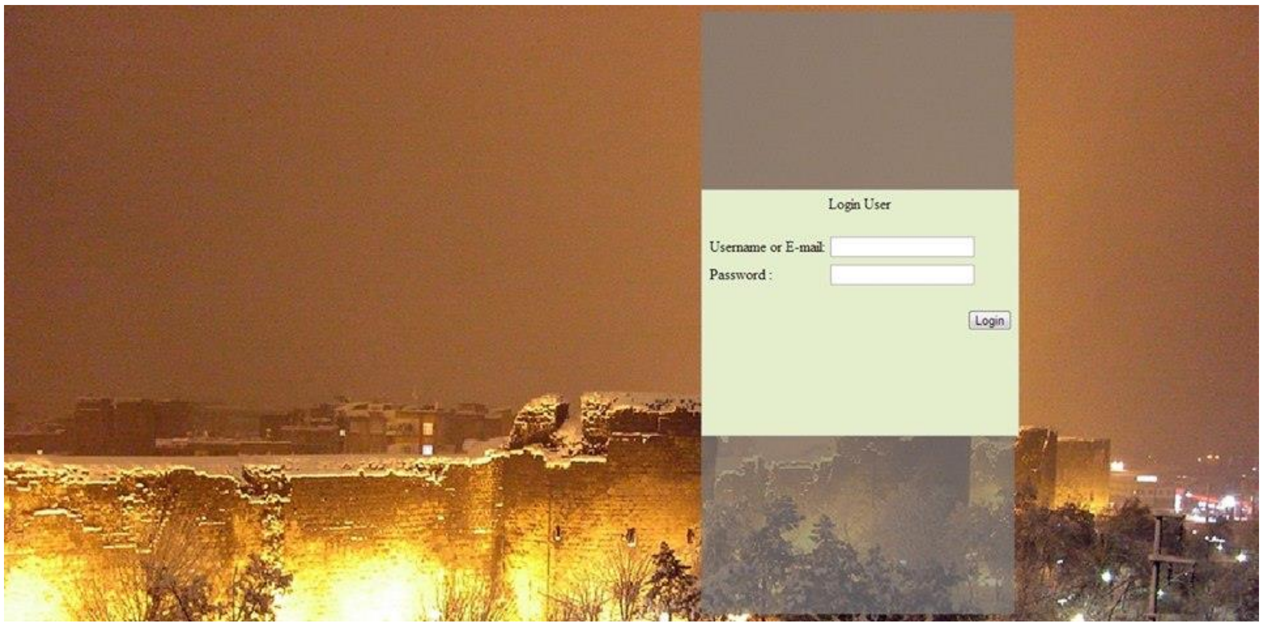


Figure 5 Sign in Page with Background

In figure 4 and 5 above, there are much simpler log in pages and far away from aesthetic view, they are meant to be for symbolic purposes only but in the figure 6 below, there is a well-structured and partly "Bootstrapped" home page is depicted which has more appealing interface. At the lefts and right side near page margins there are panels where users sees trending topics and at right side there is a scrollable (like-wise all other panels) that suggest friends when JQuery detects the scrollbar hits the bottom. The panels are drag gable so the system gives to users the perception of customization and ownership.

The panels other than the mentioned ones are user and posts panels. At the post panel a user is eligible to share some sort of data types (links, images, text etc.) to be filtered by our system later on. We are planning to establish new links at right side inside the user panel to change settings and upload a brand new profile picture. At the main panel when a user clicks on a post, the background dims and browser puts the content into the middle of the page.

There isn't the chat and message interface implemented yet at the picture below.

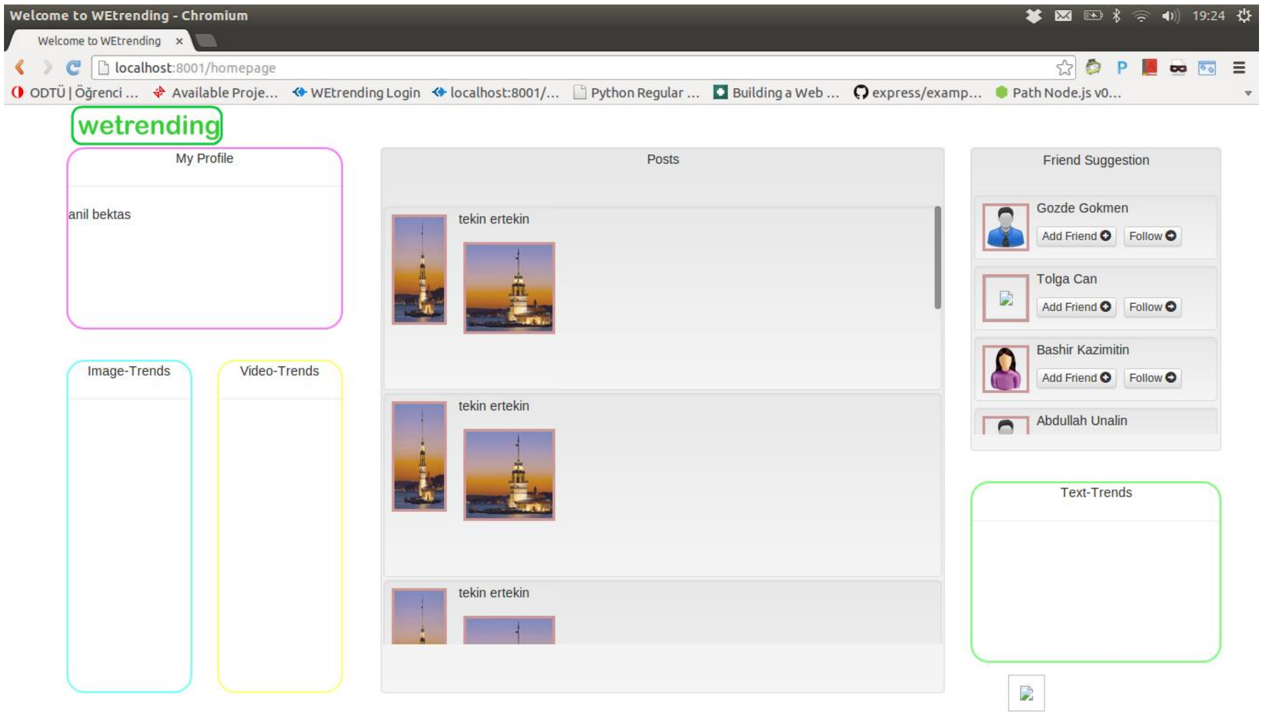


Figure 6 Homepage Panels

2.1.3 Hardware Interfaces

Hardware part of our system will be dealt with by the server. The changes here could be controlled by the managers through the system interface. By now the hardware instance we are contemplating to run on our programs has the properties explained in table 2 below:

Property	Spec
Disk Space	4096 MB
Traffic/month	81920 MB
Control Panel	cPanel
FTP Accessibility	Limitless
Database	MySQL
Row Count	Limitless
Location	Turkey
Speed	100 Mbit
Subdomains	2

Table 2: Hardware properties of the system

The major hardware requirements of the users are:

- A mobile phone, tablet or a computer connected to the internet.
- The devices should have latest browsers since we are working with new cutting edge libraries. Most browsers' first versions doesn't handle anymore this kind of systems but user shouldn't have concerned since browsers generally keep them updated.
- The system above is capable of working thousands of operations per second but user is responsible of his or her connection location since lag increases the more a user is far away from the servers. In the upcoming years if system grows successfully we will rent more servers around globe in other continents.
- The users' devices has enough working power either fed by battery or alternative current.

2.1.4 Software Interfaces

Our system will be built on Linux. The basic programming languages we will need are listed like: JavaScript, Node.js, JQuery, Bootstrap and MySQL.

We have chosen Node.JS for its speed and it is really easing the process of a coder implementing JavaScript coding at server like what happens mostly at client side so developers don't get messed with other like Python or Java at server side. Other reason is it that it is notably speedier than other current frameworks based on Java or Python. Everybody recognizes Java for its bad performance when compared to C or C++ but Node.JS is even faster than Python and Ruby and so on. Figure 7 depicts the current situation.

Other reason why we chose Node.js is that it has a large and constantly growing registry with thousands of packages where a developer easily fetch and pull new package for targeted use. For instance in case we need a file operation like read or write, besides the native framework's supporting libraries we can pull fancy packages to make typing easier or covering some deficiencies in the native library. Also there are many empty spot in native library provoke this kind of libraries which is the main reason. The registry is handled by NPM. A new package is easily be installed with command "npm install fs-extra".

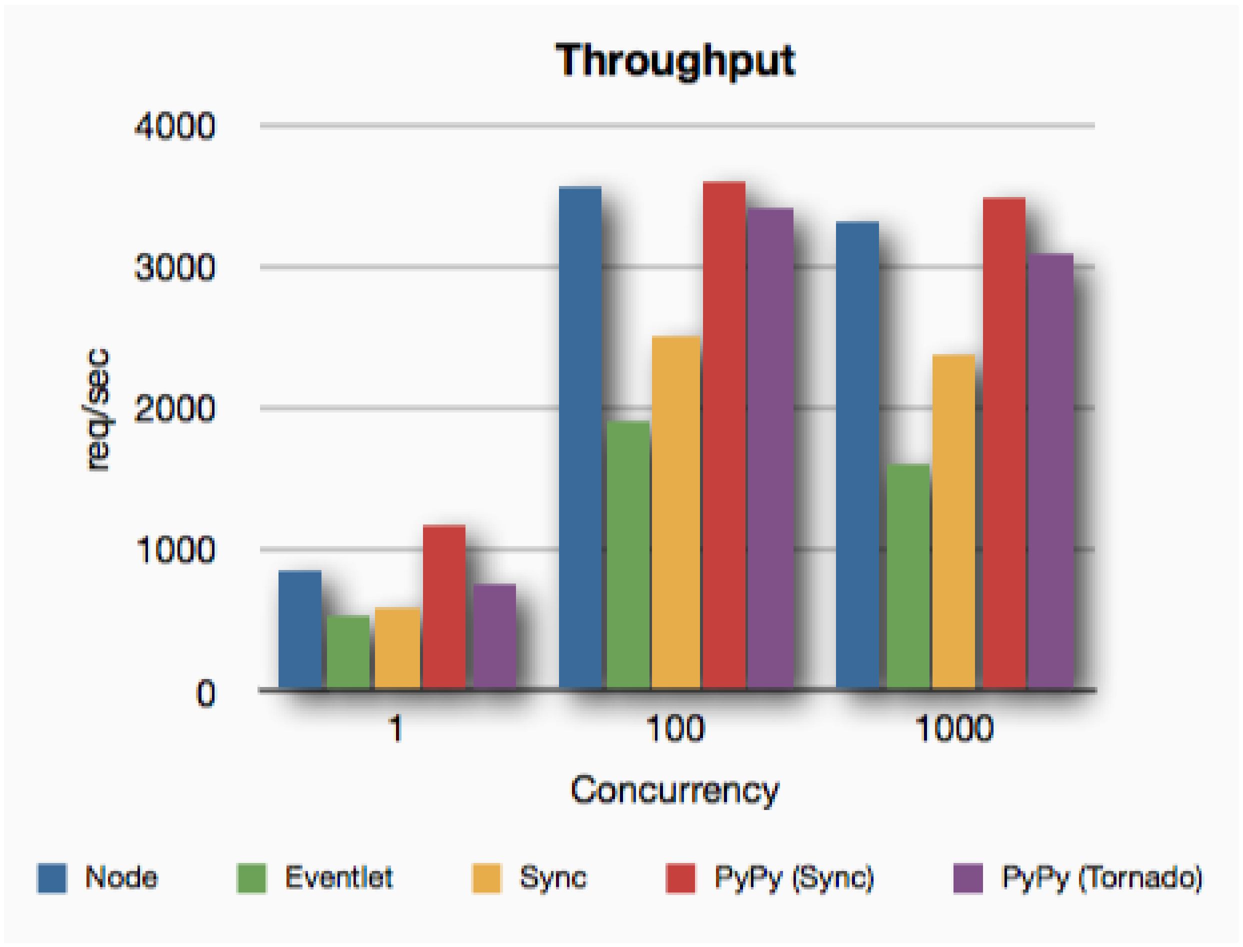


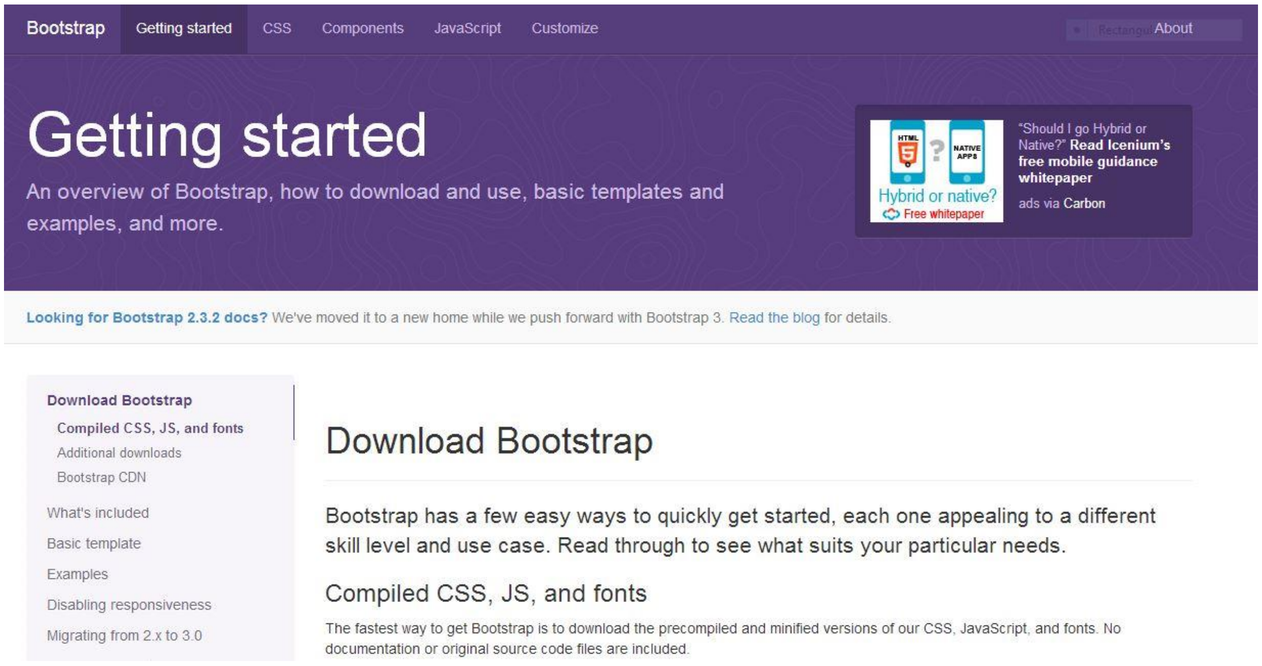
Figure 7 Throughputs [1]

The reason we chose MySQL is it is old enough to be stable unlike its brand new rivals like Mongo DB. There are many disappointed people who are constantly giving feedbacks of how they restarted their whole business logic from scratch. Mongo is new so it does not have as much legacy as MySQL. Other reasons why we chose it over others are it has wide support and easy to maintain.

On the client side we are going with JQuery and Bootstrap. There are many detractors of JQuery out in the market, may be due to its poor coding environment. We concluded that we can find support easily since it has very much legacy. For now regardless of speed its large libraries are providing tools what we do really want.

Bootstrap is still one of the most lightweight and charming frameworks around the globe that is easily integrated to a system. Moreover it is quite popular. It also has responsive design which easily makes it to adapt variety of devices with different screen sizes. For more information please look at http://getbootstrap.com/.

Figure 8 below shows a snippet from the bootstrap webpage.



Getting started

An overview of Bootstrap, how to download and use, basic templates and examples, and more.

"Should I go Hybrid or Native?" **Read Icenium's free mobile guidance whitepaper**

ads via Carbon

Looking for Bootstrap 2.3.2 docs? We've moved it to a new home while we push forward with Bootstrap 3. Read the blog for details.

Download Bootstrap

Bootstrap has a few easy ways to quickly get started, each one appealing to a different skill level and use case. Read through to see what suits your particular needs.

Compiled CSS, JS, and fonts

The fastest way to get Bootstrap is to download the precompiled and minified versions of our CSS, JavaScript, and fonts. No documentation or original source code files are included.

Figure 8 A Snippet from Bootstrap Web Page

2.1.5 Communication Interfaces

The connection between our system and users will be provided by the server. Our program is like none of the spread web sites using PHP and constantly checking a user input but driven by events. At this point sockets are notably remembered since all work load when communicating server with client is on their shoulders.

Hardware-wise the Ethernet and wireless protocols are the major ones for communicating and either one is required. The connection between a PC and our hosting site is ensured by also large, enterprise-size Ethernet switch hubs.

In the future besides the TCP protocol we are contemplating to use UDP to fasten the process but since UDP has a connection-less property we would like to make sure that we are using this tiny enhancement for lesser important operations.

In conclusion, this part is mostly meant for informative data, a user doesn't need to worry about these details since in our modern world they are all supplied and ready to use.

2.1.6 Memory

The need for RAM and Hard Disk for our system will be changing according to the number of users. As the number of users increase, we will meet and talk to the server provider and do the necessary increasing.

2.1.7. Operations

a) Users in our system could send posts, messages, like and dislike posts, make comments and share posts.

b) Our system will periodically (once every half an hour or once an hour) scan the images, texts and videos that the users have shared and find out the most shared ones among them. The retrieved data will again be presented to the users separately in the form of trends, just like the text trends in twitter.

c) When necessary, we could do some necessary changes on the images and other data in our system. For example, bringing the profile pictures to suitable dimensions.

d) Our system database will be backed up by our server provider after a certain period. This back-up will be used as needed.

2.1.8. Site adaptation requirements

a) Our system will have some built-in users for trial accounts and will automatically be added to the friends' lists of the new user. Users could log in to the system with just a click using the trial account and surf the system without signing up.

b) Necessary software need to be installed on the server.

2.2. Product Functions

There will be two types of interfaces in our system. The first one will be for the users. Users could only access the interface. Managers will have a separate interface. Managers could have access to main program and the database at the same time with privileged rights

2.3 Constraints

2.3.1. Constraints over User Actions

Users who harm and misuse our system (like frequently sending messages, adding friends, sending posts etc.) will be detected by the managers and will be fined (like banned to add friends, send messages, etc.). In case these behaviors continue, they may be kicked out of the system permanently. This punishment could be done automatically in the future when the number of users increase.

System security will be maintained by the server provider. During the possible attacks to system like (DDOS attack, Ping of Death Attack, Sync Attack, UDP attack etc.) it is server provider's job to stop them.

On the other hand the developers are responsible for sanitizing the user input regardless of anything. For now we are using mostly against anti-flood attacks like when an input becomes bigger than

1 MB which normally only contains some kB of data it is not processed anymore. The other point we are protecting is e-mails. When a user is entered an email which is not globally recognized it does not any more evaluated.

In addition, our system will have some policies that will make sure our users don't get harmed (like racism, hatred, targeting, and insults). We are planning to detect users who use and share posts like these by the complaints of other users.

2.3.2. Constraints over Hardware

The system will be checked by the managers and if needed, they will talk to the server provider and take precautions. For example, the lack of RAM and hard disk would be detected and the server provider would be informed and the problems are solved.

Auto scaling is a solution if only provided by the provider in such database critic systems. In future years if we conclude that our hardware has its limits we may turn over a new page and use HADOOP systems.

The other constraints are hardware has its own physical limit of emitting the data over its cables and processing the data. The system provider is offering 100 Mbit of speed on their lines.

2.4. Assumptions and dependencies

Our system will not be working unless its server is not working. In the future, we might find a backup server for situations like these. But for now, we don't have such an activity, which means we are dependent on the server we are working on. In addition, the database that contains our system should be continuously working. Therefore, the database of our system is also dependent on its provider. Apart from these, in case the system collapses, a small software (that could get other software and the service started) could be written in order to get the system working again.

3. Specific Requirements

The server side is handled by Node.js. Our little node.js scripts are going to be responsible for writing in/out to database and updating the GUI using templating languages likes JADE. The GUI are going to be mostly designed using gorgeous Twitter Bootstrap and the interaction is thought to be handled by JQuery like movable panels or drag-drop operations. The database, as most free software use, is MySQL. There exist many ORM (Object Relational Mapper) for Node.js like Sequelize or its own MySQL library.

There might exist components requiring PHP but again the authentication is going to be handled by Node.js as well as most of the other work like sending notifications. The tests exists for this kind of software architecture but as its late use, we postpone our observations to a later time, frankly, after achieving an alive local web site.

3.1. Interface Requirements

Users complete all their action over a graphical user interface. If we want to instantiate this relation with an example, when a user click on one of his/her friend’s post, the like button changes its displaying state. Afterwards the GUI process the received POST or GET data to one level behind to Node.JS application server. Node.JS sanitizes the data for abused usage (for instance adding html tags to change the state of the page or thousands of repetitive actions in row) and processes the data such that the database can understand and interpret. It is called transaction. What ORM stands for is Object Relational Mapping is kind-of a package. In this case we are using Node.JS-belonging MySQL package.

At last database turns the transaction and information into static data to be read or written afterwards. It manipulates its regarding tables to store the information in a convenient way. Performance of the queries are up to coders mostly. The saved data can be accessed from server with a request one way or another.

Figure 9 below depicts what happens when one clicks a button.

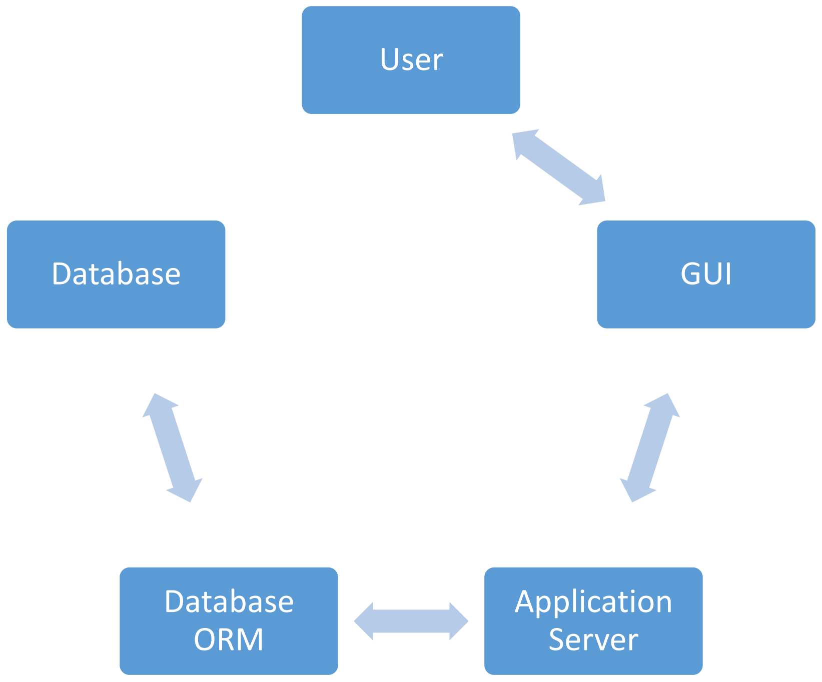


Figure 9 What Happens After Clicking

3.2. Functional Requirements

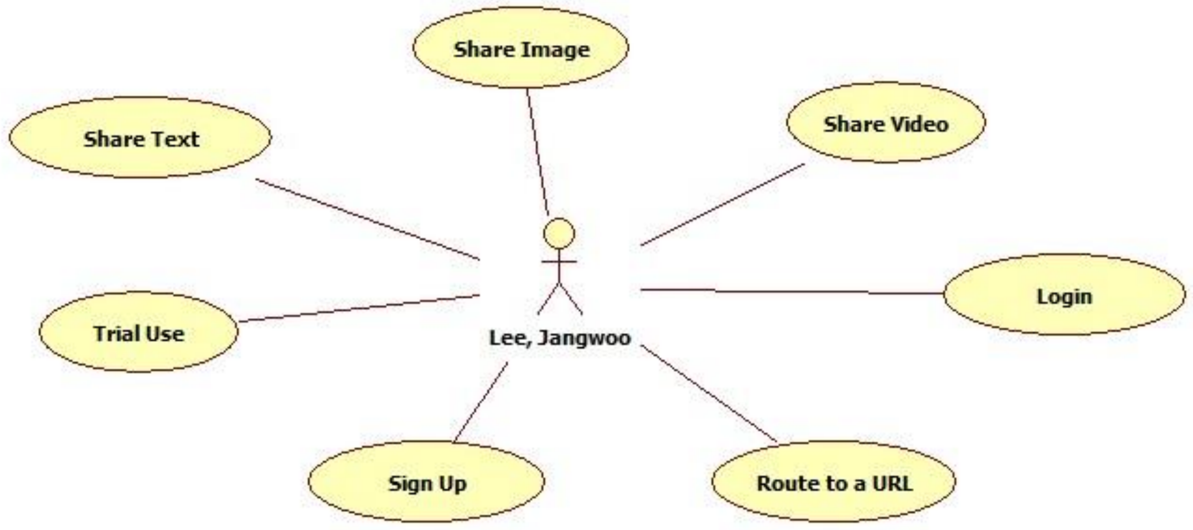
3.2. Functional Requirements

Data flow is sampled at the figure above. The flow of data is based on with blocking of user and the new trends, from GUI to server program, from there to database and vice versa. With assist of this structure the user's input is recorded into system. Moreover, the output what the user desires emitted through the same flow with a different direction. The following use case diagrams are depicting both user and admin use cases.

Below a sample user Lee, is able to

- Share image in main panel
- Share video in main panel
- Share text in main panel
- Not sign up but go for trial account
- Sign up and start using the system
- Route back and forth in URLs
- Login after registered

Figure 10 Sample User Use Case 1



Below there are other operation which a user can action over the web site. A sample user Lee is able to

- Change his settings (password, name, surname, marital status etc.)
- Upload profile pictures
- Undo dislike or like
- Remove the content shared before

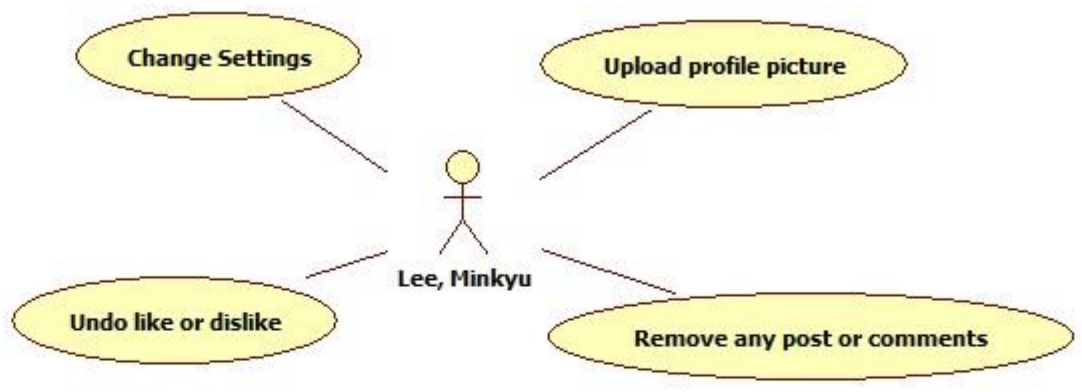


Figure 11 Sample User Use Case 2

Below other miscellaneous actions that a user is able to perform. A sample user Lim is able to

- ❖ Like a post
- ❖ Comment on a post
- ❖ Dislike a post
- ❖ Report to system administrators a violent user
- ❖ Chat instantly
- ❖ Use message feature to send files or make a simple chat

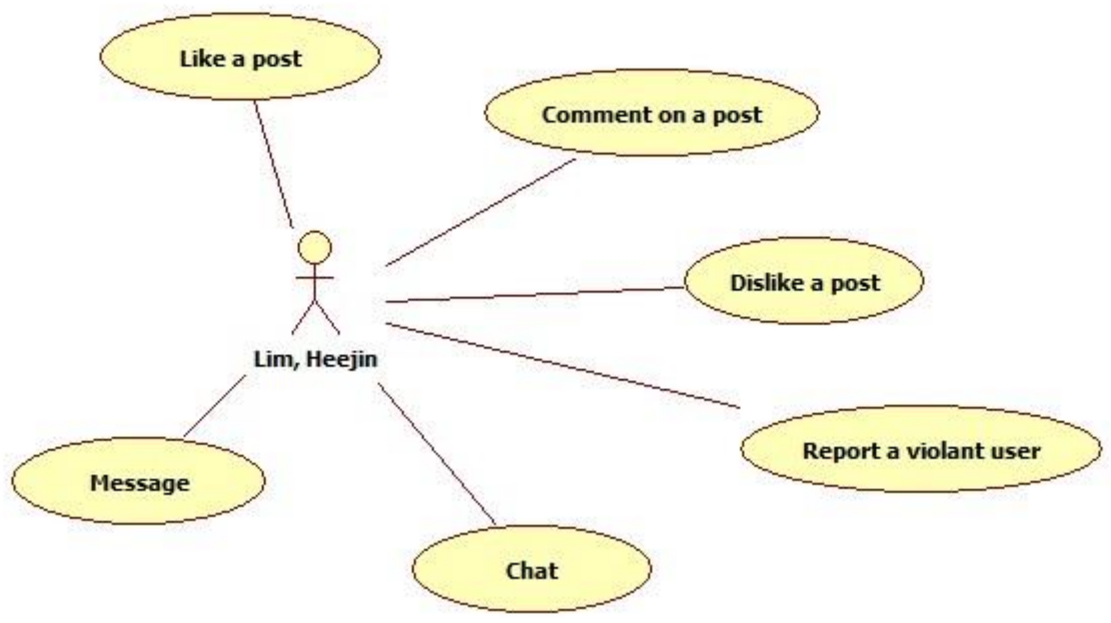


Figure 12 Sample User Use Case 3

Below what a system manager is able to perform is depicted. Though this diagram doesn't includes the types of system administrators and their permission in more bird's eye viewpoint they are able to perform

- ❖ Alter database tables
- ❖ Drop unnecessary tables
- ❖ Oversee a reported user and take necessary actions
- ❖ Change the server's code with the crew
- ❖ Update the design elements
- ❖ Create a brand new URL and inside a full-fledged web page

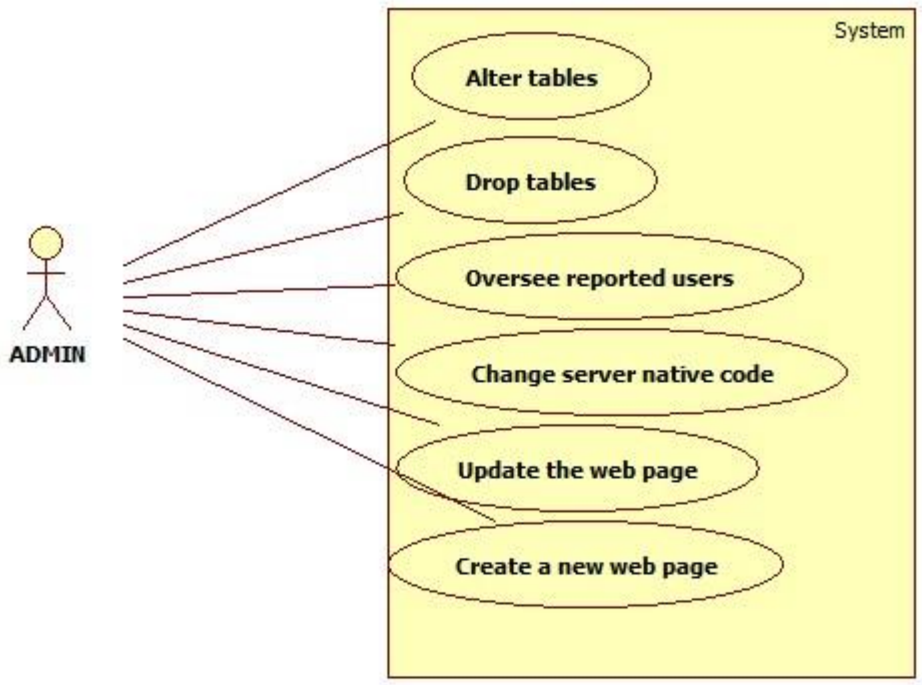


Figure 13 Sample admin Use Case

3.3. Non-Functional Requirements

3.3.1. Performance Requirements

Per 10 000 users of our system we need one administrator. This count may decrease with along the improvement curve's acceleration because in future, our little scripts will be operating duties like banning, blocking or punishing like any administrator does. Moreover we expect that per 10 000 user there need 1 GB RAM and per one year usage of 10 GB hard disk is needed. It depends on bandwidth and posting frequency at the same time.

3.3.2. Design constraints

The RAM, processors and hard disks are going to increase parallel with along the posting count. For the sustainability of the system following a high demand of sign up, we have to increase our RAM and static storage quota.

For reporting IEEE standards are used in both SRS and SDD meanwhile on software side while coding we are only using JavaScript and HTML. Thanks to Node.Js at this point.

3.3.3 Database Requirements

Since we have preserved a user's last state without forgetting their passwords, names, posts, what they liked and where they commented, we should implement database relational mappers over a database or a physical storage. For now, if we use the billing and usage schemas of Heroku a Node.JS hosting site and say we rent 512 GB for $750 dollar per month and also a user is sharing 600 KB of data per day.

512 gigabytes = 536 870 912 kilobytes

(536870912) / 600 = ~894 000 users if the servers up time is just only 1 day.

894 784 / 30 = 29 826 users

In one month we have a limit of 29 000 users roughly and it never increases after this point. So proportionally we have to enlarge and scale our database systems. For now at first 10 months we are expecting at most 30 000 already so 512 gigabytes will be enough. Note that we have limitless row count. We are sharing the reference inline here: https://devcenter.heroku.com/articles/heroku-postgres-plans

3.3.4. System Attributes

This section specifies some system attributes that should exist to keep system alive and reliable. These properties are reliability, usability, security and integrability.

3.3.4.1 Reliability

The reliability defines how much the system is biased to failures both in terms of hardware and software. At software side, we are trying to stay away from some deficiencies like design defects, wear-out software components. Also in case of any failures the system should have been coded in a neat structure since the complexity makes harder to maintain. The more complex the system gets the more we pay effort on developing reliability. The metrics of complexity of systems are initially based on LOC (Lines of Code) but since there isn't any standard way to calculate it we aren't sharing any more approaches here.

The paragraph above summarizes how we maintain reliability within certain metrics. In conclusion if not considered carefully, software reliability can be the reliability bottleneck of the whole system and it is no easy task.

3.3.4.2. Usability

This topic defines the usability of the system. This system uses mostly the similar GUI what people are get used to from Facebook or Twitter. The web site uses many GUI interfaces from rendering picture to video links displaying not only the link but thumbnails. Our web system is barely using pure texts other than text trends but widely implementing movable and drag able, user-friendly panels.

3.3.4.3. Security

This attributes encompasses some measures in the software life-cycle to prevent code from some gaps and system from vulnerabilities. We are sanitizing all user inputs backed up via text boxes or other related html elements. In order to not breach the bridge between user and us to secure their data we will be constantly supervising and stay touched with hosting firm.

3.3.4.4. Integrability

Since the system is open and subject to new components and changes we must ensure system is easily integrable to new software parts. We should be easily linking and binding together the whole system with their add-ons. A software engineer should own required skills and so do the software developers. The code development are staying away from spaghetti coding and implying neat commented coding style.

For instance later than we shuttled the system we are contemplating to add “News Trend” for filtering popular news. In theory it should be as easy as adding a new panel and using some current database functions without a big hassle.

4. Data Model and Description

4.1. Data Description

The data objects which are planned to be used:

- User
- Post
    - Picture List
    - Video List
    - Text
    - Commentary list
    - Dislike List

- Message
- Group
- Other package related variables and temporary variables (not included below)

4.1.1. Data Objects

Figure 14 below depicts a diagram for the data objects we use.

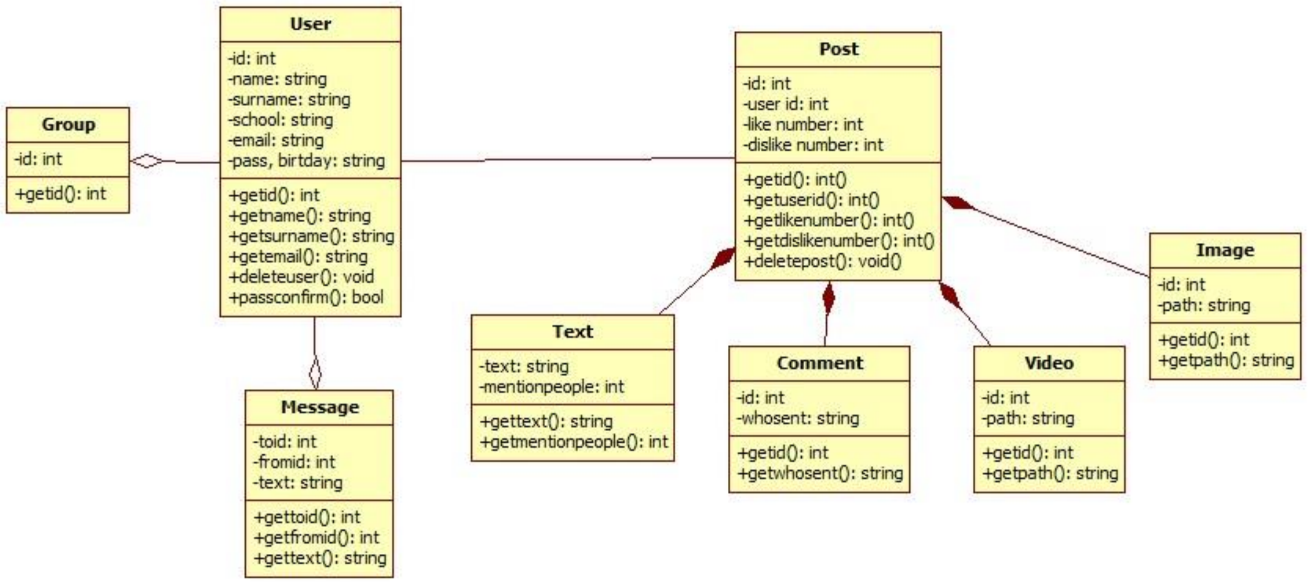


Figure 14 Data Objects

4.1.2 Data Dictionary

Table 3 below depicts some data elements related to objects. These attributes are used widely and frequently through the system. The whole data type is “var” in canonical JavaScript. The temporary variables are dismissed also some of the contents are subject to change.

Name	Alias	Description
group_id	ID of a group	Unique global ID

user_id	ID of a user	Unique global ID
user_name	User name	Name of the user
user_surname	User surname	Surname of the user
user_school	User school	Educational information of user
user_email	User email	Email of user (unique)
user_password	User password	Password of user
post_id	ID of a post	Unique global ID
post_user_id	ID of the user who made a post	Unique global ID
post_like_number	Like number	Post total like count
post_dislike_number	Dislike number	Post total dislike count
image_id	Image id	Unique global ID
image_path	Path of image	Path of the respective content
message_toid	Sender id	The receiver end of the message
message_fromid	Receiver id	The sender end of the message
message_text	Message text	Context
text_text	Inside of text	Context
text_mentionpeople	Mentioned users	Which people has mentioned
comment_id	Id of comment	Unique global ID
comment_whosent	Sender of comment	The user ID involved in action
video_id	ID of video	Unique global ID
video_path	Path of video	Path of the respective content
image_id	ID image	Unique global ID
image_path	Path of image	Path of the respective content
express	Express Framework	Variable for using Express package content
http	HTTP Framework	Variable for using Http package content
path	Path Framework	Variable for using Path package content
socket.io	Socket.io Framework	Variable for using Socket.io package content

check	Check Framework	Variable for using Check package content
mkdirp	Mkdirp Framework	Variable for using Mkdirp package content
fs	Fs Framework	Variable for using Fs package content
stochator	Stochator Framework	Variable for using Stochator package content

Table 3: variables related to the objects

5 Behavioral Model and Description

5.1 Description for software behavior

1. Adding Friends

A user X that has an id adds the other user Y that has a different id. User interface part transmits this request to main program. Main program makes database save this request. And database saves it in a list. Later, the user that has a friendship request sees it and confirms it. User-interface sends information that Y approved X's request to main program. Main program makes database part remove this request from the Y' request list. Also, main program makes database add Y to X's friend list. In the same way, main program makes database add X to Y's friend list. And later on, database part does processes about data interchanging.

2. Comments

A user X comments on user Y's post that has an id Z. . . User-interface part transmits this post to main program. Main program makes database save this post. And database saves it in a list. There is no need to have an approval this time. Also, main program makes database add X' comment to Y's Z post. At the end, this post will be added to database.

5.2 State Transition Diagrams

Figure 15 depicts the State Transition Diagram about adding friend:

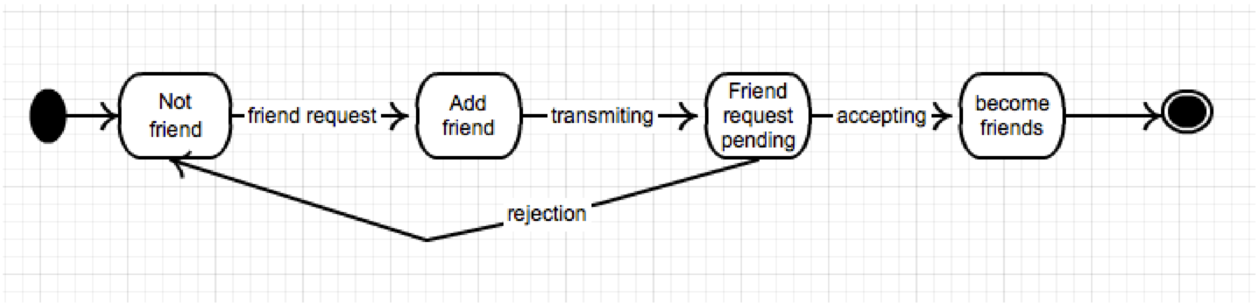


Figure 15: Adding Friend

State Transition Diagram about comments:

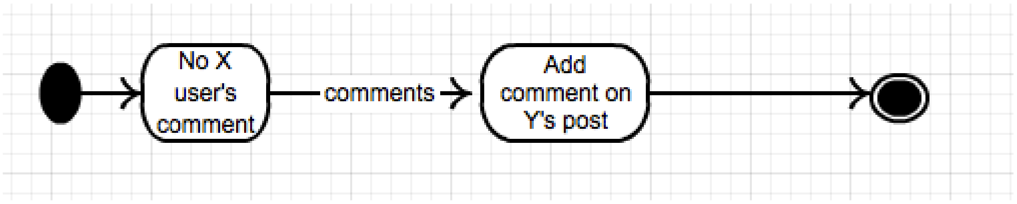


Figure 16: Comments

6. Planning

6.1 Team Structure

There are four members in the team each of whom has different tasks assigned to them. The following table; table 4, shows each member's tasks.

Name	Task
Anil Bektas	Application server, backend
Tekin Ertekin	Founder, client side enhancements
Bashir Kazimi	Business Logic
Gozde Gokmen	Research on miscellaneous subjects

Table 4: Team Members and Tasks

6.2 Estimation (Basic Schedule)

We are planning to accomplish the tasks needed to complete the system in a certain period of time and in a certain order. The following figure shows an estimated schedule for concluding our project at the end of the first term. The more detailed figures are depicted in SDD report

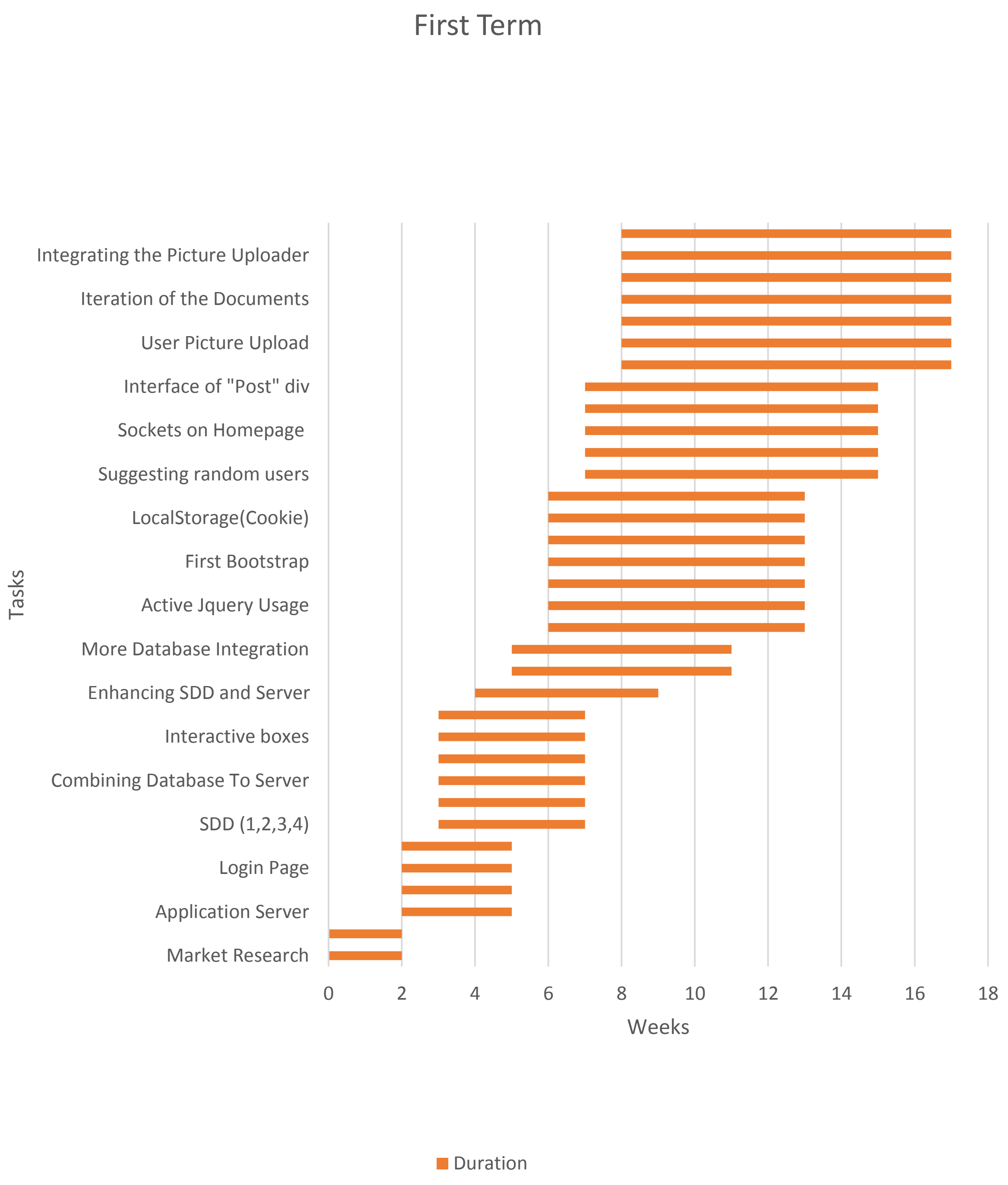


Figure 17 First Term Tasks

7. Conclusion

In summary, we are creating a social network. In doing this, we are using some useful features from some of the existing social networks and also we are adding some new features of our own to make social networking easier, more fun and more useful for the users. For example, we are using instant messaging, sharing posts, like buttons, comments and other basic features that Facebook uses, we are using dislike button and general comment features that YouTube uses, we are using text trends that twitter uses, we are using multiple media posts that so.cl uses. The new features we add would be image and video trends, automatic trial accounts, movable panels and some other new features which will make things easier for the users. This document; SRS (Software Requirement Specification), gives a brief description of the system we are creating, and also it serves as a guideline for us to iterate while creating the application.

8. References

[1]

GRIFFITHS, K. (2012). Retrieved from Blog Kgriffs: http://blog.kgriffs.com/2012/10/23/python-vs-node-vs-pypy.html

9. Change Log

Changes to SRS after revision are listed as follows:

- ✓ Cover page got renewed.
- ✓ Introduction part got elaborated on more.
- ✓ Figures and tables were named, numbered and referred to throughout the text.
- ✓ The whole document was proofread and grammatical and spelling mistakes were fixed.
- ✓ In section 1.4, a table was used to show the definitions, acronyms and abbreviations in a more clear way.
- ✓ For section 1, another sub-section; 1.6 was added to point out the references.
- ✓ In each section from 2.1.1 through 2.1.4, a figure was added for a more clear illustration of the features.
- ✓ In section 3.1, the previous figure was renewed to explain the system feature in a more clear way.
- ✓ In section 3.2, three use case illustrations for user and one use case illustration for manager was added to elaborate more on the functional requirements of the system.
- ✓ In section 3, two sub-sections were added as 3.3.3 Database Requirements and 3.3.4 System Attributes and each were explained in details.

- ✓ In section 6.2, the schedule for first term task was enhanced and changed from a table into a more explanatory graph.
- ✓ The section 8 from previous SRS document which enlisted the table of contents was removed from the end of the document and brought to the beginning of the new SRS document.
- ✓ Reference section was added as section 8.