SAPIENZA
UNIVERSITÀ DI ROMA

Progetto ed implementazione di un Sistema social basato sulla condivisione di idee di viaggio

Facoltà di
Corso di laurea in
Ingegneria dell'Informazione, Informatica e Statistica

Candidato
Massimiliano Cestra
1564408

Relatore
Massimo Mecella

A/A 2015/2016

Alla mia famiglia

INDICE

Chapter 1

1.1 Realtà di interesse

Da un articolo della internet marketing inc. è emerso che il 52 % delle persone prendono ispirazione per i loro viaggi da foto postate da amici su facebook e il 76% di coloro che effettuano un viaggio sono soliti postare la loro esperienza online.Tra questi ultimi i più attivi sui social sono sicuramente i cosidetti « nomadi digitali » o « influencer » che hanno avuto l abilità di associare il piacere del viaggio al proprio lavoro . Da queste considerazioni è nata l'idea di sviluppare un applicazione sui viaggi . Essa è stata sviluppata tenendo a mente l'idea social che sempre più sta dilagando nella società . In questo modo gli utenti hanno la possibilità di condividere la stessa passione e curiosità per i viaggi .

La piccola comunità che si vuole creare, si vuole riferire a viaggiatori esperti e principianti, in modo da poter condividere opinioni, idee, luoghi e foto di ogni angolo del mondo . L'applicazione chiamata « TripTrack » si divide in due pagine principali : il profilo personale di ogni utente e la home page .

1.2 Gestione dei dati

L'applicazione che prende il nome di "TripTrack" si occupa di gestire nella base di dati diverse entità quali: Utenti,Posts, Luoghi e Immagini.
L'entità utente è caratterizzata da:

1. nome
2. email univoca, non potranno quindi essere associate email uguali per istanze di utenti diversi

3. password che sarà salvata sul database in modo criptato attraverso un password digest creato dalla libreria BCrypt, una stringa di dimensioni variabili che identifica in modo univoco la password inserita
4. tre ulteriori digests (risultato dell'applicazione dell'algoritmo hash a uno specifico testo/documento) utilizzati per salvare le credenziali dell'utente nel browser per un futuro accesso per attivare l'account dell'utente al momento della registrazione, e infine per il reset della password
5. un ulteriore digest utilizzato per salvare il token generato dall' oauth con facebook.
6. Un attributo booleano che identifica se l'account dell' utente è stato attivato attraverso la verifica dell'account.
7. Due attributi booleani per identificare il ruolo dell'utente.

Uno User può creare dei post nell'applicazione e un post è associato ad uno e un solo User.

I posts, invece, sono caratterizzati da due importanti attributi:

1. il contenuto di tipo testo
2. un eventuale stringa che rappresenta l'url di una picture

Inoltre,d'interesse nell'applicazione è il prelievo di tutti i posti di Facebook dove l'utente si è taggato o è stato taggato con associate a quel luogo, se presenti, le relative foto.

L'entità Foto, è quindi caratterizzata dalla risorsa dell'immagine(url), in modo da mostrarla all'utente .

L’entità Luogo, è caratterizzata da :

1. due floats che rappresentano la latitudine e longitudine dello specifico posto
2. un nome

E’ di interesse nell ‘applicazione la relazione amicizia tra due utenti.
Uno user può inviare una richiesta di amicizia ad altri utenti e quest’ultimi possono decidere se accettarla o meno.
Esistono poi particolari tipi di utenti come gli Admin ed i SuperUser.
L’admin ha il permesso di eliminare altri user e posts.
I Super User invece sono particolari utenti che hanno raggiunto un numero elevato di amicizie e questi ultimi possono essere solo seguiti da utenti normali.
Uno User infatti può seguire più super user, se non è presente una relazione d’amicizia, ma non può essere seguito da questi ultimi. Inoltre un Super User non può inviare una richiesta d’amicizia se è già seguito.
Queste 2 classi di utente sono gestite in modo semplice e chiaro attraverso l’utilizzo di 2 attributi booleani nell’entità User.
Infine un utente può mettere i likes a più post e un post può ottenere un like da più utenti.

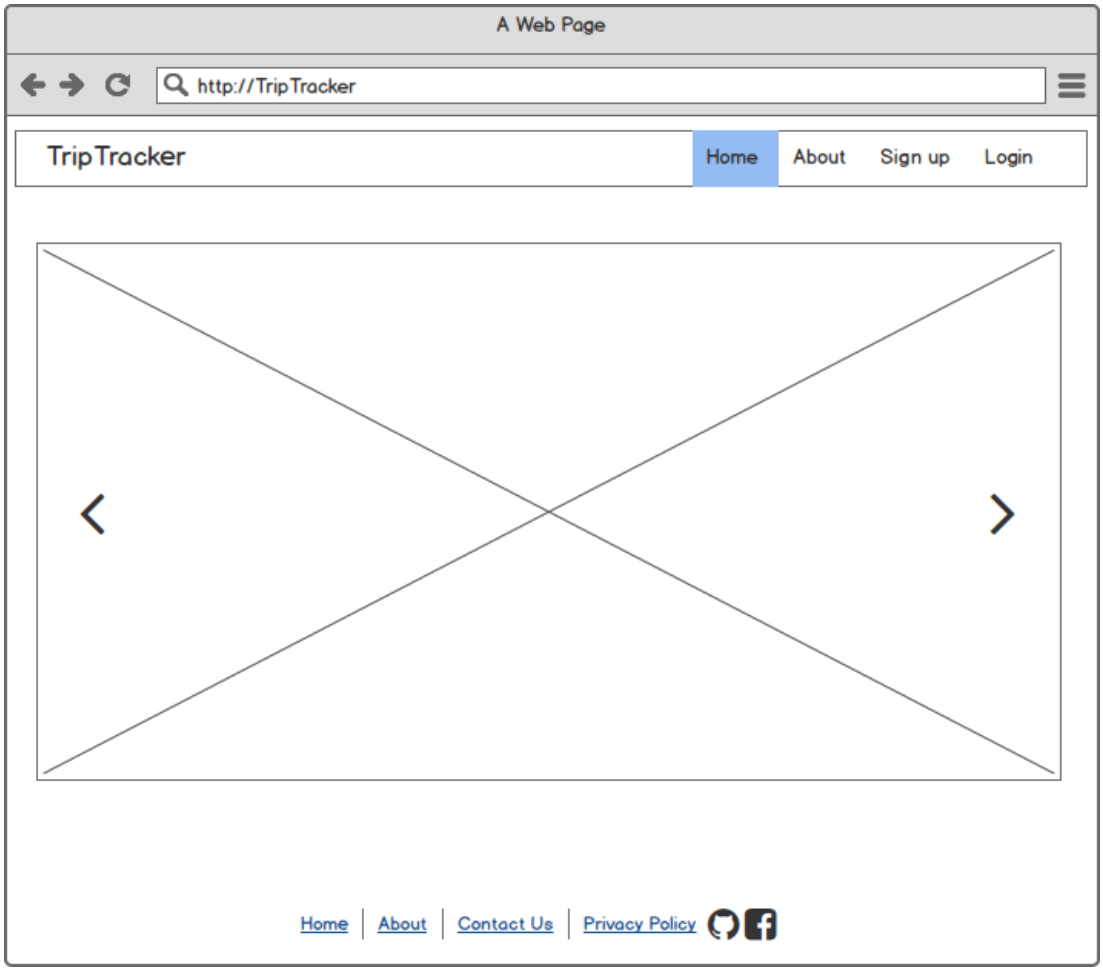
1.3 Funzionalità

Al fine di avere una visione globale di come l’applicazione dovrà risultare, e sempre seguendo le linee guida dell’approccio Agile, sono state messe a punto le funzionalità da sviluppare, progettando per ognuna di esse l’interfaccia grafica che ne dovesse risultare.
Come detto nella sezione 1.2 l’applicazione è stata strutturata verticalmente cosi che ogni utente possa progredire la sua posizione diventando un Super User. I ruoli gestiti sono tre: User, Admin, Super user. Al fine di mantenere un filo logico nell’illustrazione delle funzionalità sviluppate verranno trattate dapprima le attività

che ogni utente base è possibilitato a svolgere e successivamente si menzioneranno le funzionalità extra disponibili per ogni super user e admin.

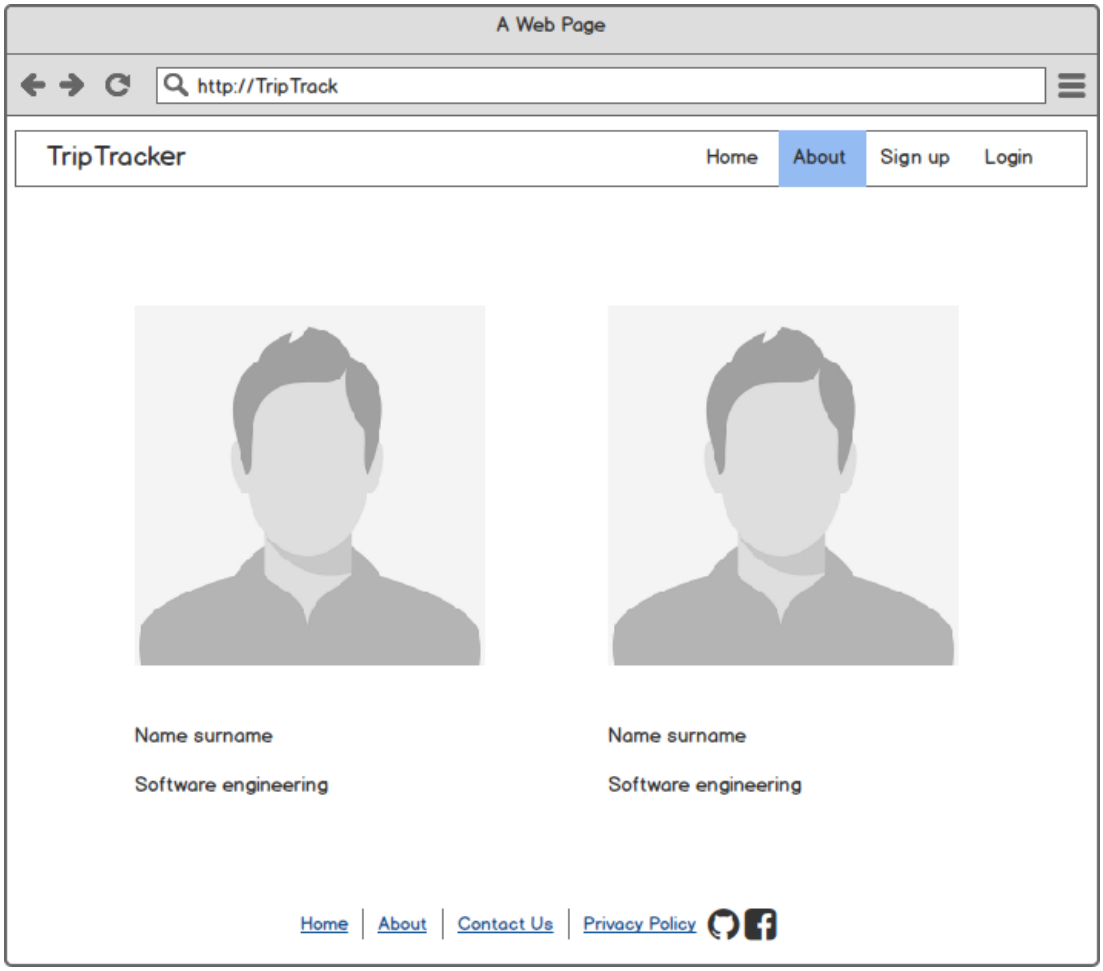
Visitata la home page dell'applicazione, questa si occupa di verificare se l'utente ha ancora una sessione valida per cui risulta essere loggato o meno. Considerato il caso base in cui l'utente non sia ancora iscritto all'applicazione la pagina iniziale risulta essere quella di Figura 1.1, dove la navbar nell'header da quattro opzioni all'ospite: Home (Figura 1.1), About, Signup e Log In.

Figure 1.1



La pagina “About” è stata ideata per dare informazioni aggiuntive su coloro che hanno sviluppato il sistema, in modo tale che qualsiasi utente possa contattare il team in caso necessiti di supporto, (Figura 1.2).

Figure 1.2



Nel caso in cui l’ospite si trovi al primo approccio con l’applicazione, potrà registrarsi all’interno di essa attraverso l’azione di sign up, questa rappresenta la prima vera e propria feature del sistema definita in Figura 1.3 (user-story) , la cui progettazione grafica risulta visibile nella Figura 1.4.

Figure 1.3

TRIPTRACK **US 01**

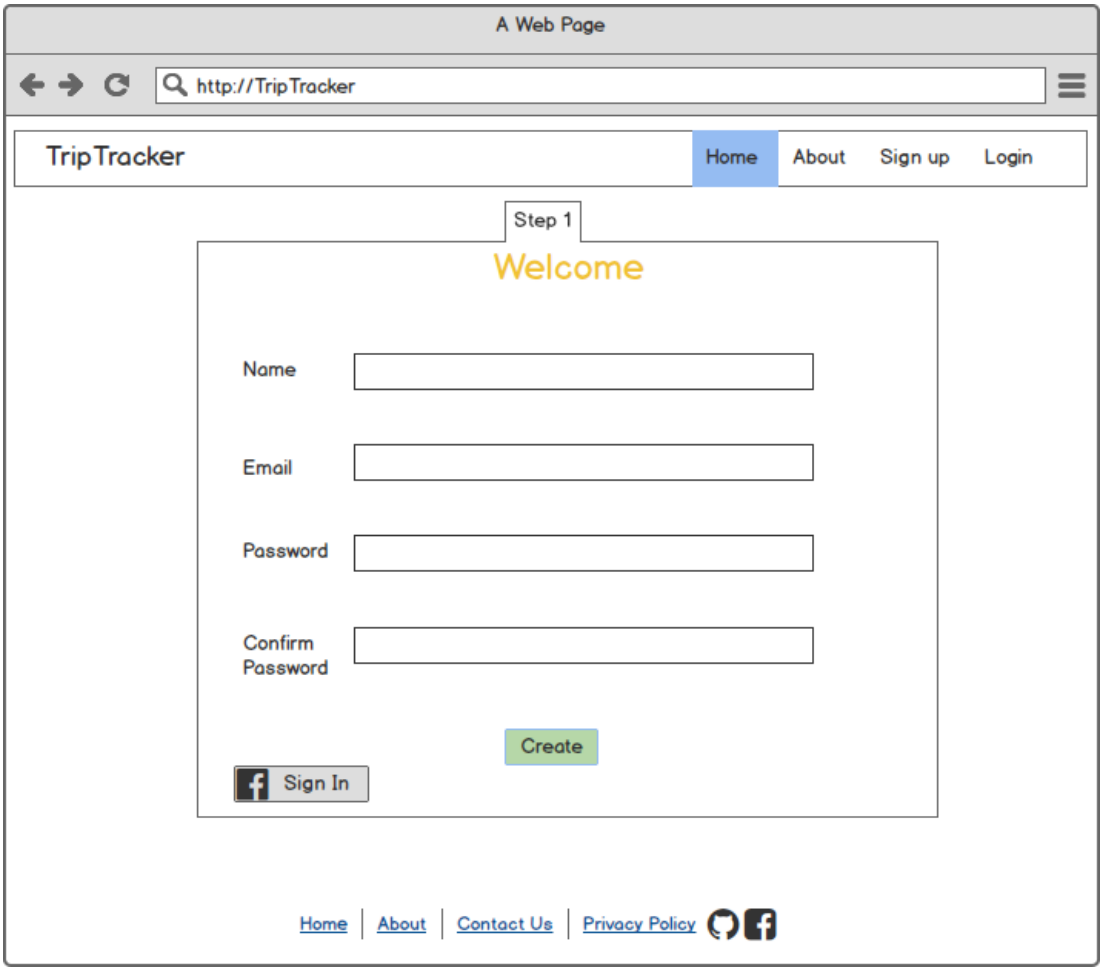
As a User

Wants to sign up in the application

So that I can enter in the TripTrack community to use the application's functionalities

PRIORITY: High

Figure 1.4



Una volta che l'utente si è registrato, riceverà una conferma tramite mail , in modo da verificare la validità di quest'ultima. Il contenuto dell'email permette all'utente di accedere direttamente al sistema e di confermare la sua identità (Figure 1.5).

Figure 1.5

TRIPTRACK **US 02**

As a Owner of the application

Wants to send an email to each user who signed up in TripTrack

So that I gather to each user his successfully sign up

PRIORITY: High

Completata la fase di registrazione, l'utente sarà abilitato ad entrare nell'applicazione per poterne utilizzare i servizi. Cliccando direttamente nel link annesso alla mail sarà reindirizzato sul suo nuovo profilo utente, equivalentemente una volta terminata questa fase, il profilo utente è accessibile semplicemente svolgendo il login nel sistema. Il login può essere effettuato secondo due modalità : tramite l'applicazione stessa, tramite un servizio esterno. Il classico login richiede all'utente di inserire username e password in modo da validare la sua identità (Figure 1.6, Figure 1.7).

Figure 1.6

TRIPTRACK **US 03**

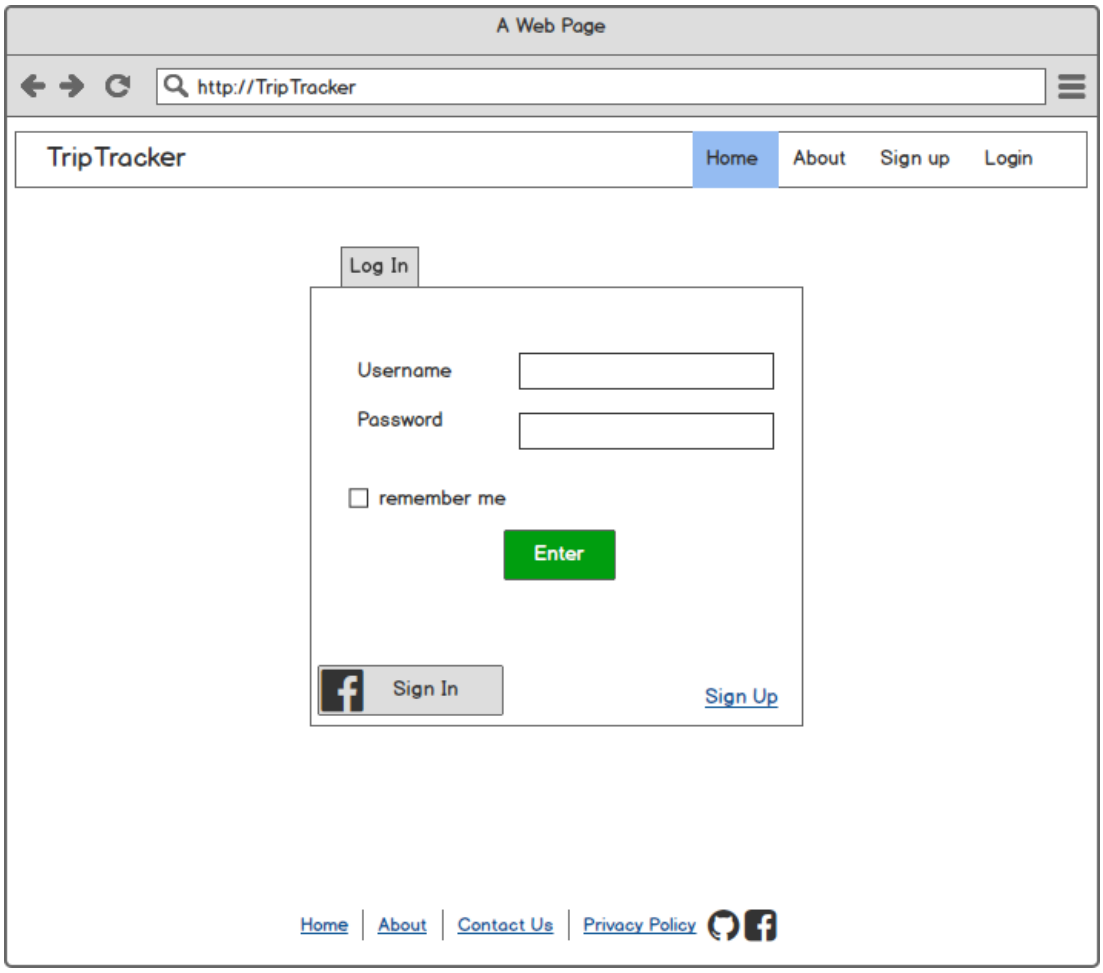
As a User

Wants to log in into the application

So that I can use different services offered by the application

PRIORITY: High

Figure 1.7



Se invece l'utente desidera effettuare l'accesso tramite un servizio esterno, TripTrack si interfaccia con il social network Facebook in modo che l'utente si possa registrare e loggare nel modo più veloce possibile (User Story: Figura 1.8)

Figure 1.8

TRIPTRACK	**US 04**
As a User	
Wants to log in with Facebook	
So that I can directly jump the sign up part to enter in the application	
PRIORITY: High	

Il layout del profilo utente è stato progettato per essere il più intuitivo possibile con :

- mappa generata dalla comunicazione con il sistema google maps (vedi capitolo 2)
- foto dell' utente affiancata da informazioni base quali nome e cognome
- statistiche sul numero di amici/followed/followers(se super_user)
- posts pubblicati

Anche la navbar, una volta che ci si è loggati nell'applicazione "TripTrack" ha subito delle modifiche, in questo modo l'utente potrà accedere più velocemente alla lista degli amici, alla pagina di ricerca utenti e alla gestione delle richieste di amicizia ricevute.

Partendo proprio dalla navbar possiamo analizzare un pagina alle volta , in modo da descrivere le varie funzionalità disponibili in quella sezione.

Ogni utente, visitando la home page (Mockup: Figura 1.9), avrà una visione generale del suo profilo, in quanto potrà non solo aggiungere un nuovo post con eventuali

foto(User story: Figura 1.10, Figura 1.11) ma avrà la possibilità di vedere tutti i posts più recenti pubblicati dagli amici (User story: Figura 1.14).

Figure 1.9

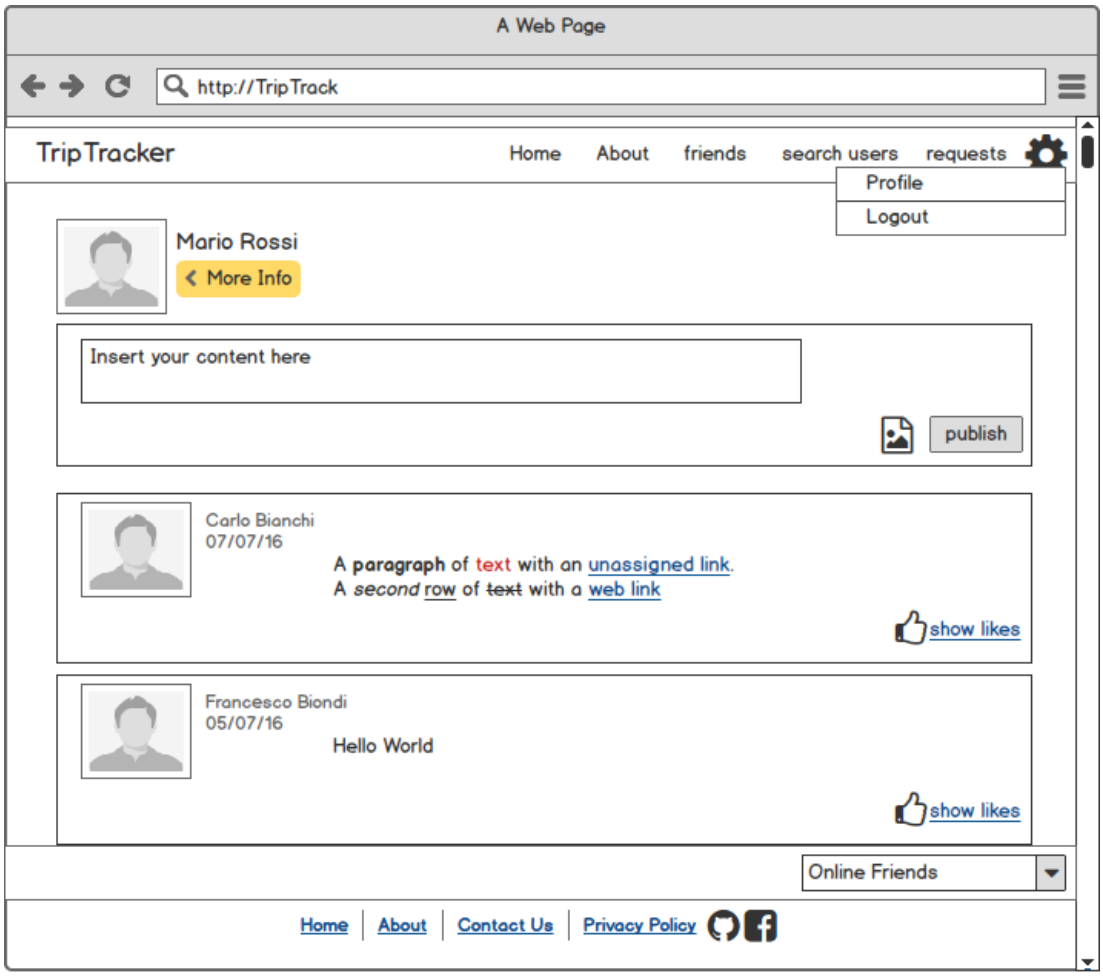


Figure1.10

TRIPTRACK **US 05**

As a User

Wants to add a micropost

So that I can publish an opinion for everything I want

PRIORITY: High

Figure 1.11

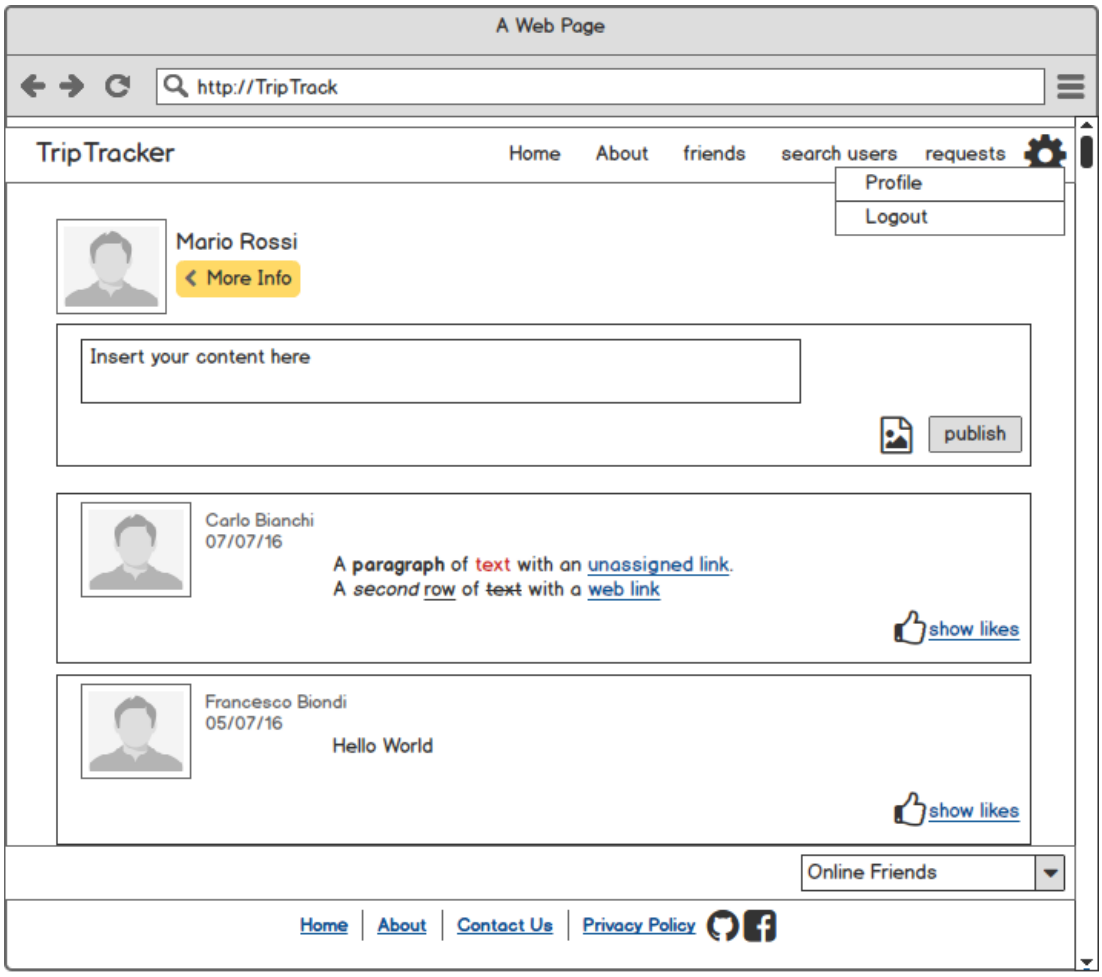


Figure 1.12

TRIPTRACK **US 05**

As a User

Wants to add a micropost

So that I can publish an opinion for everything I want

PRIORITY: High

Figure 1.13

TRIPTRACK **US 05**

As a User

Wants to add a micropost

So that I can publish an opinion for everything I want

PRIORITY: High

Figure 1.14

TRIPTRACK **US 05**

As a User

Wants to add a micropost

So that I can publish an opinion for everything I want

PRIORITY: High

Ogni utente può dare la sua opinione riguardo un post o una foto attraverso l'aggiunta di un like, riferiti al post (Figure 1.15).L'utente può inoltre vedere tutti i likes specifici per un determinato post (Figure 1.16, Figure 1.17)

Figure 1.17

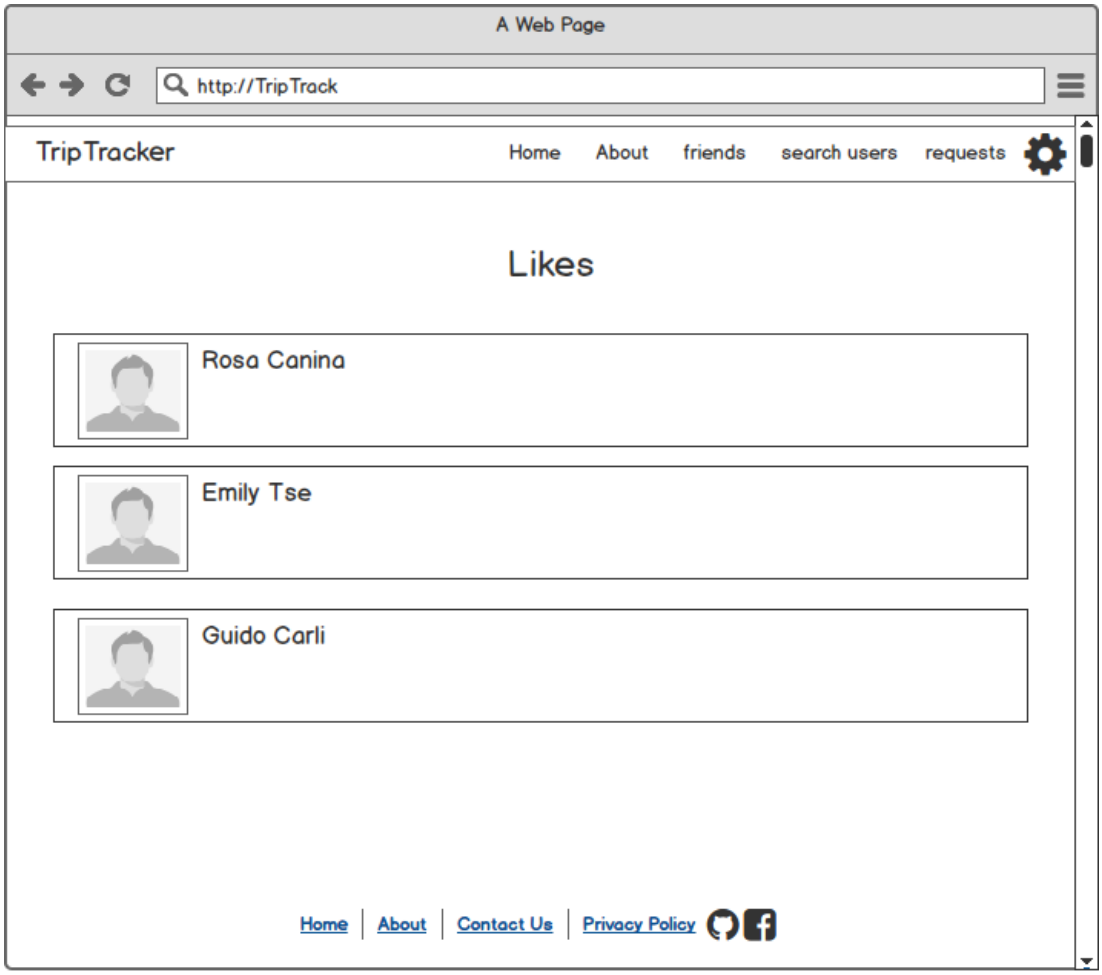


Figure 1.15

TRIPTRACK	**US 08**
As a User	
Wants to add a like to a specific post	
PRIORITY: Low	

Figure 1.16

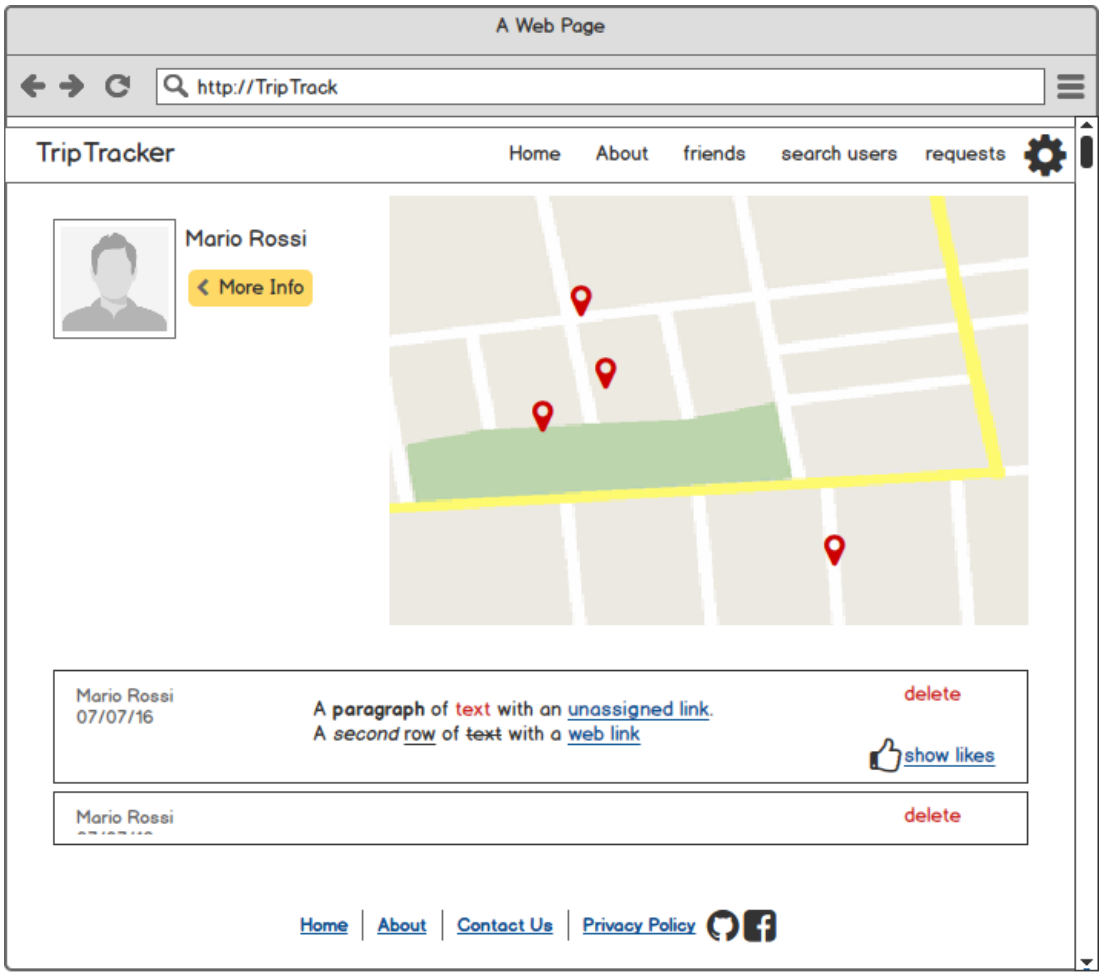
TRIPTRACK	**US 09**
As a User	
Wants to see all the likes associated to a specific post	
PRIORITY: Low	

Un post può essere cancellato dalla stessa persona che lo ha scritto (Figure 1.18). Tutte queste azioni sono visibili in Figure 1.19.

Figure 1.18

TRIPTRACK	**US 10**
As a User **Wants to** delete a specific post that I published **So that** anyone can't read it anymore **PRIORITY:** Medium	

Figure 1.19



C’è inoltre una pagina per esplorare la lista degli utenti che utilizzano l’applicazione (Figure 1.20), in questo modo ogni utente può inviare una richiesta di amicizia ad un altro (Figure 1.21) o seguire un super utente (Figure 1.22, Figure 1.23)

Figure 1.20

TRIPTRACK **US 11**

As a User

Wants to search user inside the application

PRIORITY: High

Figure 1.21

TRIPTRACK **US 12**

As a User

Wants to send a friendship request

So that I can improve the number of friends

PRIORITY: High

Figure 1.23

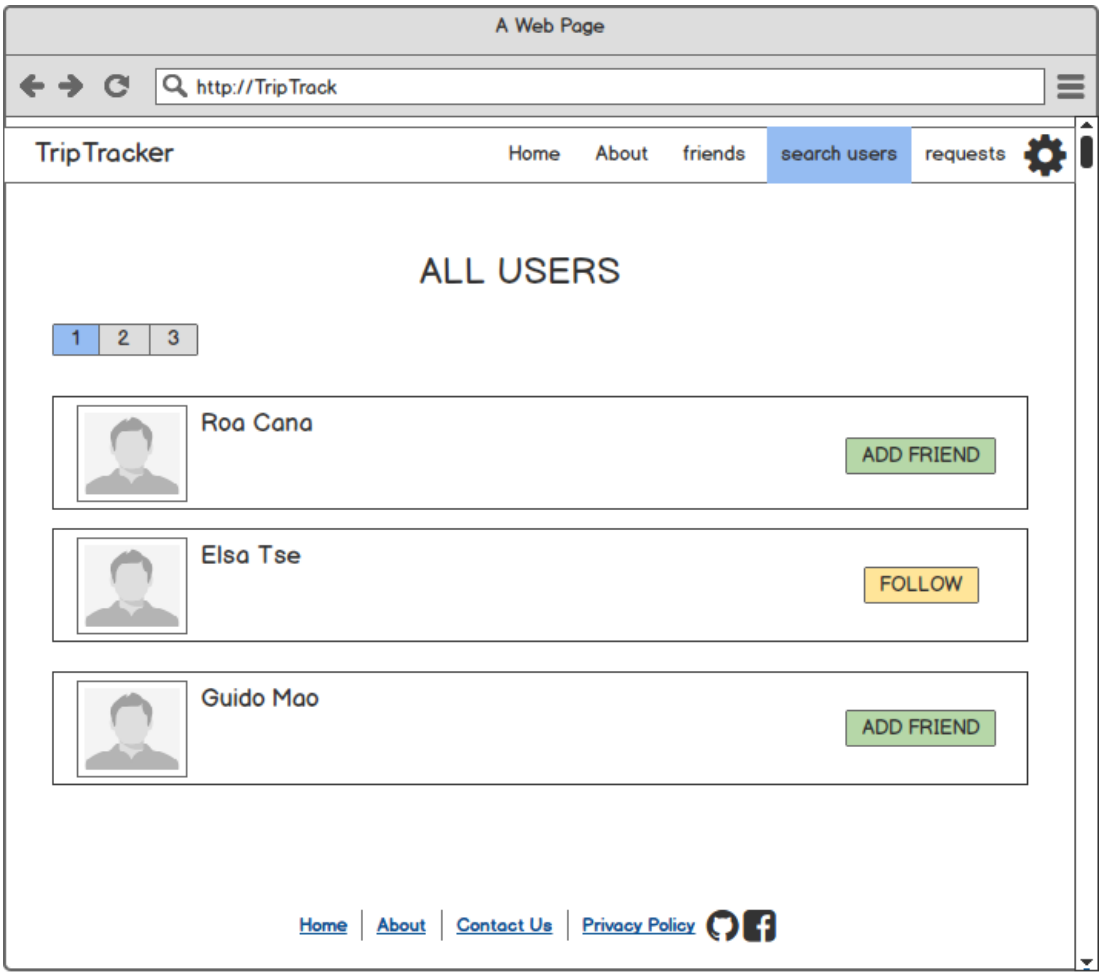


Figure 1.22

TRIPTRACK	**US 13**
As a User **Wants to** follow a super user PRIORITY: High	

La pagina per mostrare gli amici (Figure 1.24), invece, dà la possibilità di cancellare un amico (Figure 1.25)

Figure 1.24

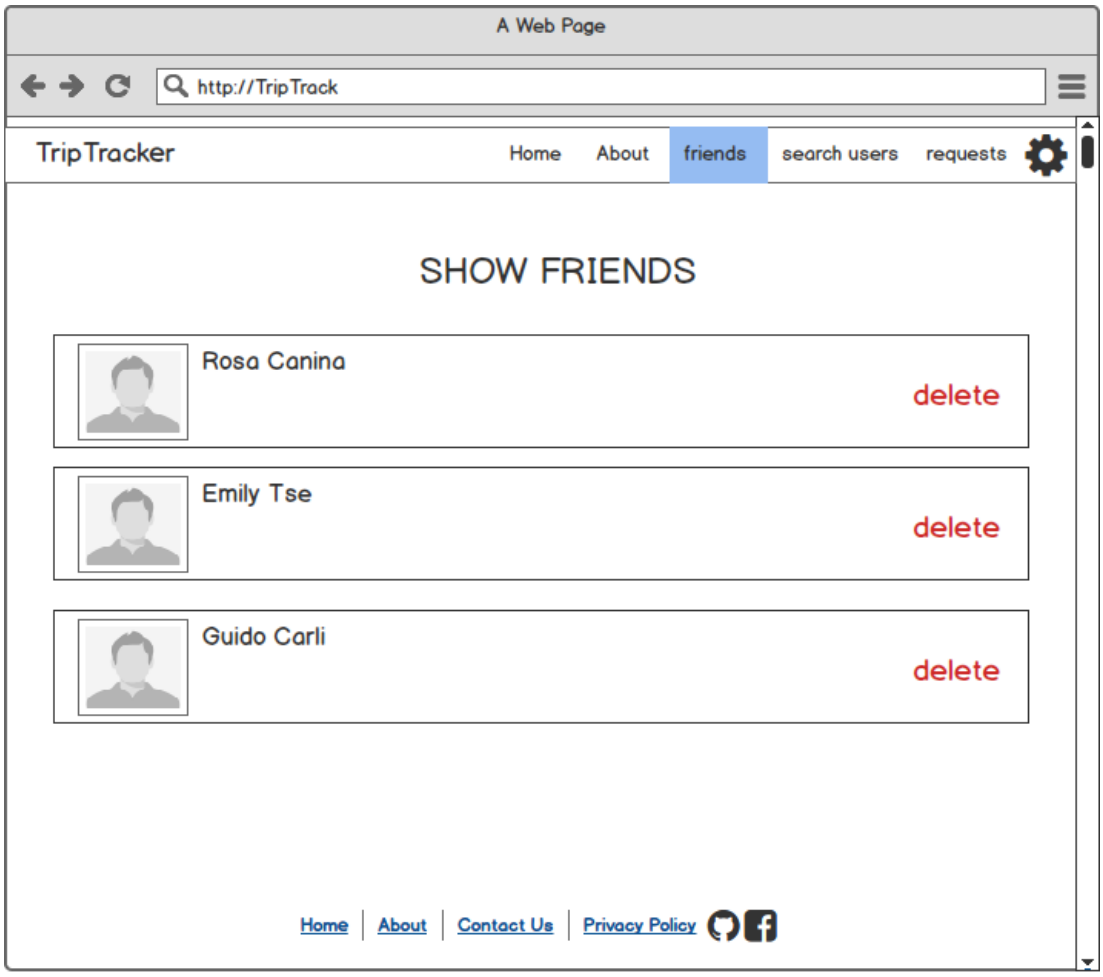


Figure 1.25

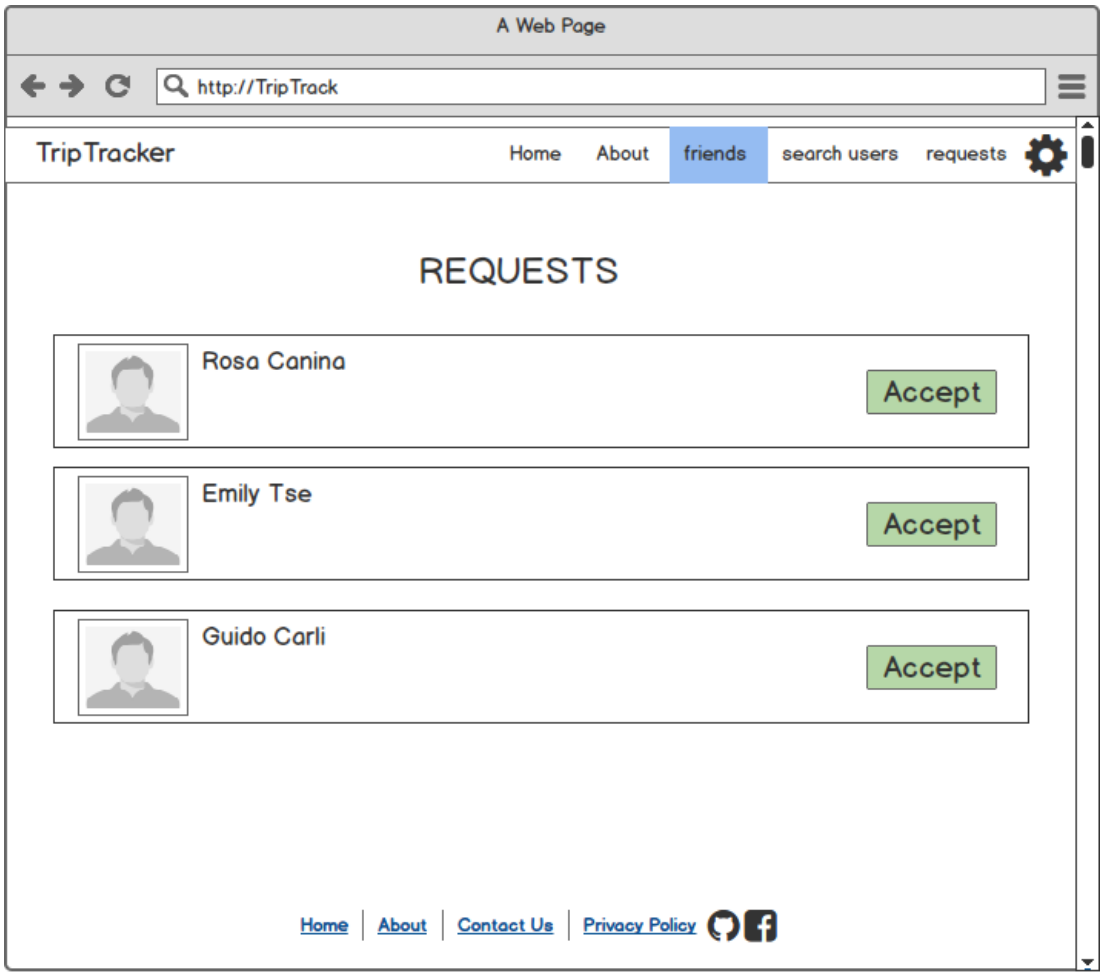
TRIPTRACK	**US 14**
As a User **Wants to** see the list of friends **PRIORITY**: High	

Ogni utente può anche accettare o meno un richiesta di amicizia e verificare essa nella pagina “friendship request” (Figure 1.26, Figure 1.27)

Figure 1.26

TRIPTRACK	**US 15**
As a User **Wants to** see all the friendship requests **PRIORITY**: High	

Figure 1.27



L'ultima parte che sarà descritta riguarda i places, ogni utente loggato nell'applicazione attraverso facebook, può salvare i luoghi visitati, presi da facebook, in TripTrack e mostrarli sulla mappa presente nel profilo (Figure 1.28, Figure 1.29). Ogni luogo può avere inoltre foto diverse, che l'utente può vedere al click su un determinato posto nella mappa del profilo (Figure 1.30, Figure 1.31)

Figure 1.28

TRIPTRACK	**US 16**
As a User **Wants to** have a personal map inside my profile **PRIORITY**: High	

Figure 1.29

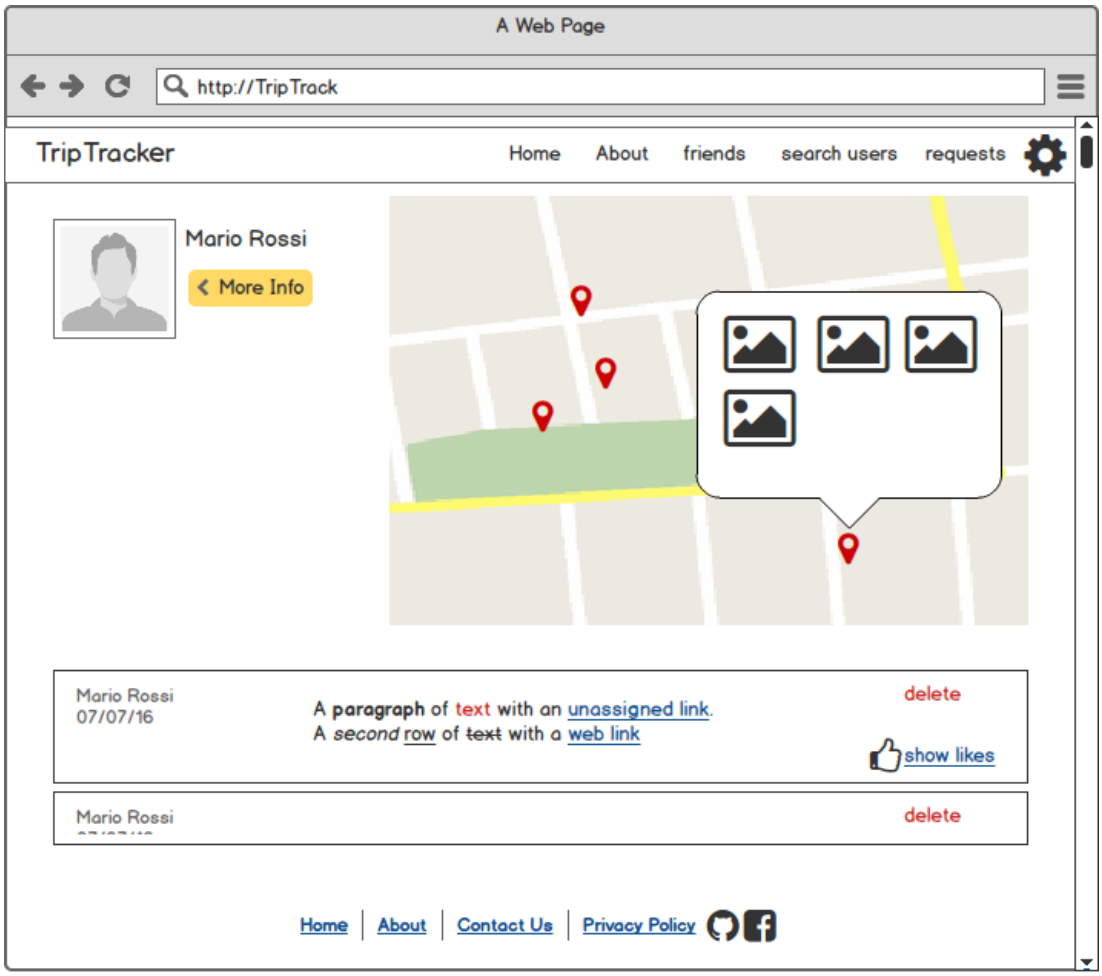
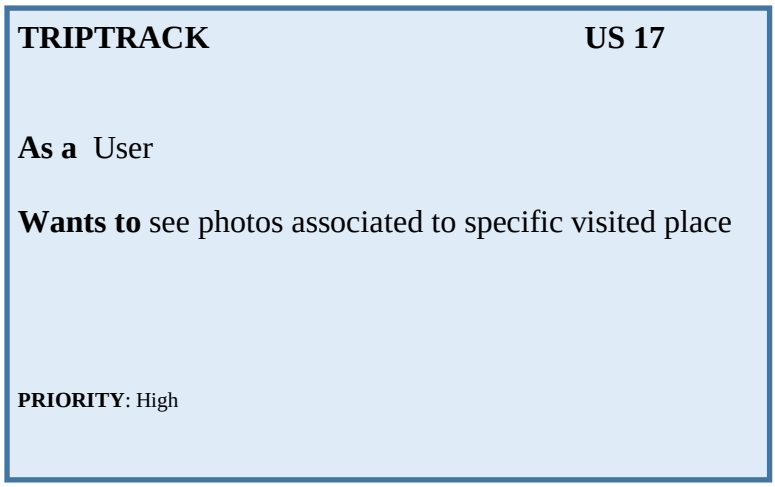
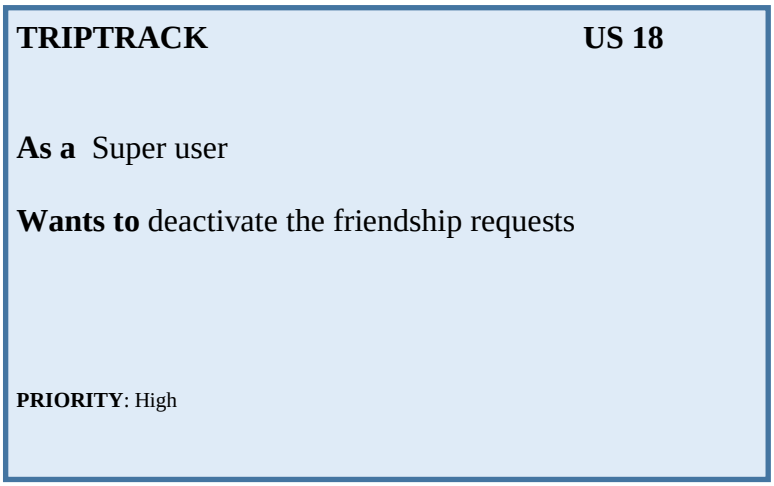


Figure 1.31



Come già detto un utente può avere diversi ruoli: admin o super user. Generalmente un super user può essere seguito da altri utenti (Figure 1.32)

Figure 1.32



L’admin, invece ha due importanti funzionalità: cancellare qualsiasi utente lui voglia e cancellare un qualsiasi post di altri utenti (Figure 1.33, Figure 1.34, Figure 1.35)

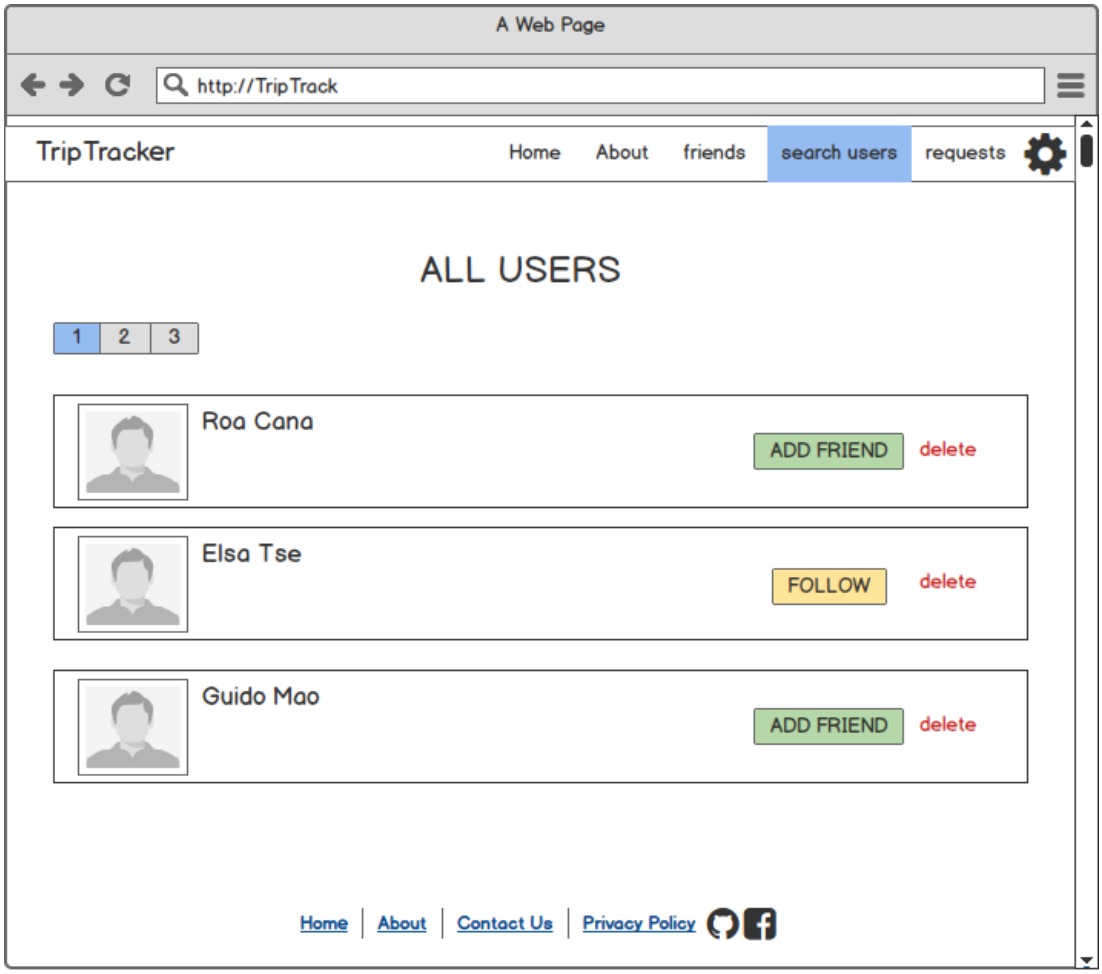
Figure 1.33

TRIPTRACK	**US 19**
As an Admin **Wants to** delete each user I want **PRIORITY**: High	

Figure 1.34

TRIPTRACK	**US 20**
As an Admin **Wants to** delete each post of other users **PRIORITY**: High	

Figure 1.35



Un utente, infine, può effettuare il logout dall'applicazione, modificare il profilo e resettare una password (Figure 1.36, Figure 1.37, Figure 1.38)

Figure 1.36

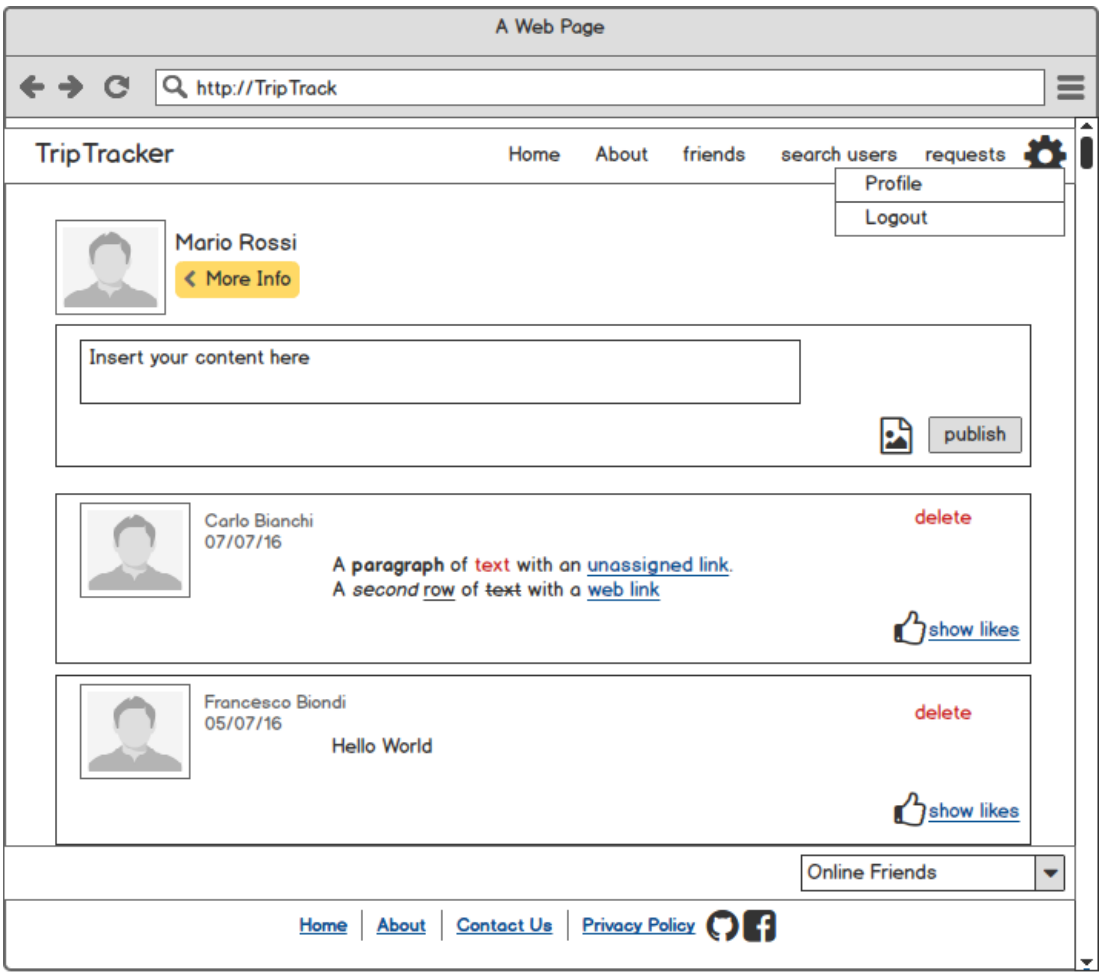


Figure 1.37

TRIPTRACK	**US 21**
As a User **Wants to** log out from the application **PRIORITY**: High	

Figure 1.38

TRIPTRACK **US 22**

As a User

Wants to edit his profile

PRIORITY: High

Figure 1.39

TRIPTRACK **US 23**

As a User

Wants to reset his password

PRIORITY: High

Chapter 2

Tecnologie e metodologie utilizzate

In questo capitolo delineeremo i tempi di sviluppo e definiremo in modo chiaro le attività che ogni persona del team dovrà svolgere.
Verranno inoltre presentati quei servizi extra (al di fuori del framework rails e il linguaggio ruby), utilizzati nello sviluppo

2.1 Servizi esterni

Sono stati utilizzati quattro servizi esterni

- Facebook
- Google Maps
- SendGrid
- SimpleStorageService(S3)-Amazon

Google Maps Api e S3 di Amazon sono stati trattati in dettaglio dalla mia collega Giulia Giugno.

2.1.1 Facebook

Facebook è stato utilizzato per permettere all'utente di autenticarsi tramite un social network esterno, è stata installata la gemma "devise"per utilizzare Oauth 2.0. Il protocollo Oauth è stato ideato da Blaine Cook nel 2006 per garantire l'accesso protetto alle informazioni di un utente, senza però in alcun modo visualizzare informazioni delicate quali la sua password. Nelle figure sottostanti
(Figura 2.1)(Figura:2.2) verrà mostrato come è stato configurato Devise per l'oauth con facebook.

```
config.sign_out_via = :delete

# ==> OmniAuth
# Add a new OmniAuth provider. Check the wiki for more information on setting
# up on your models and hooks.
# config.omniauth :github, 'APP_ID', 'APP_SECRET', scope: 'user,public_repo'
################################################################################################
config.omniauth :facebook , ENV['FACEBOOK_KEY'] , ENV['FACEBOOK_SECRET'],
           callback_url: ENV['MY_URI']+'/users/auth/facebook/callback',
           scope: 'email,user_location,user_hometown,user_tagged_places,user_photos,user_friends,publish_actions,user_posts
```

Oauth: Figura 2.1

```
devise_for :users, :controllers => { :omniauth_callbacks => "users/omniauth_callbacks" }
```

Oauth: Figura 2.2

Nella (Figura 2.3) invece , mostriamo la parte della view che gestisce la registrazione dell'utente , evidenziando il link per il login con facebook.

```
<%= form_for(@user,url:signup_path) do |f| %>
  <%= render 'shared/error_messages', object: f.object %>

  <%= f.label :name %>
  <%= f.text_field :name, class: 'form-control' %>

  <%= f.label :email %>
  <%= f.email_field :email, class: 'form-control' %>

  <%= f.label :password %>
  <%= f.password_field :password, class: 'form-control' %>

  <%= f.label :password_confirmation, "Confirmation" %>
  <%= f.password_field :password_confirmation, class: 'form-control' %>

  <%= f.submit "Create my account", class: "btn btn-primary" %>

  <p>or <%= link_to "Sign in with Facebook", user_facebook_omniauth_authorize_path %></p>
<% end %>
```

Oauth: Figura 2.3

Una volta che l'utente ha effettuato il login con facebook viene chiamata una callback url che è gestita dall omniauth controller e in particolare dall' azione "facebook" (Figura 2.4)

```
def facebook


  auth=request.env["omniauth.auth"]
  @user = User.from_omniauth(auth)

  if @user.persisted?  #persisted return true if the user exists and hasn't been destroyed
    digest = User.digest(auth.credentials.token)
    @user.facebook_token=auth.credentials.token

    if @user.provider and @user.uid #allora ci sarà anche la mail
      @user.update_columns facebook_digest: digest

    else #ha solo la mail
      @user.update_columns(provider: auth.provider , uid: auth.uid ,
                           activated: true , facebook_digest: digest )
    end
    log_in @user
    face_api
    redirect_to @user
  else
    if @user.valid?
      log_in @user
      face_api
      redirect_to @user
    else
      render 'users/new'
    end
  end
end
```

Oauth: Figura 2.4

L'azione appena mostrata chiama un metodo del Model User che interagisce con il database per la registrazione dell'utente.(Figura 2.5)

```
def User.from_omniauth(auth)
  digest=User.digest(auth.facebook_token)
  user=find_by(provider: auth.provider, uid: auth.uid)
  if user.nil?
    user=find_by(email: auth.info.email)
    if user.nil?
      user=User.new(email: auth.info.email, password: Devise.friendly_token[0,20],
                    name: auth.info.name, facebook_token: auth.credentials.token,
                    facebook_digest: digest , activated: true)
      user.save
      user
    else
      user
    end
  else
    user
  end
end
```

Oauth:Figura 2.5

Inoltre è stata installa anche gemma “HTTParty” per poter interfacciarsi con il social network attraverso le Graph Api. Queste sono state utilizzate per richiedere a Facebook informazioni sui “feed”, “friends_tag_places” e “place” in modo da salvare nel database interno all’applicazione i luoghi e le annesse foto che ogni utente ha postato su facebook, per mostrarle sulla mappa presente nel profilo.
Come possiamo vedere sia nella precedente figura(Figura 2.5) che in quella sottostante(Figura 2.6) dopo aver effettuato il login con facebook viene chiamato il metodo privato dell’ OmniauthController che prepara 1’hash contenente tutto il necessario per le tre chiamate alle graph api di facebook.

```
private

  def face_api
    face_uri="https://graph.facebook.com/v2.8"

    options_place = { query: {fields: "place", access_token: @user.facebook_token} }
    options_friends_tag_places= { query: {fields: "tagged_places,friends", access_token: @user.facebook_token} }
    options_feed= { query: {fields: "place,attachments", access_token: @user.facebook_token} }

    param_face= {place: { option: options_place , url:  face_uri+'/me/photos' } ,
                 friends_tag_places: { option: options_friends_tag_places , url: face_uri+'/me' },
                 feed: { option: options_feed ,url: face_uri+'/me/feed' } }

    getFaceGraph param_face
  end
```

Graph_Api:Figura 2.6

Il metodo privato “getFaceGraph”è colui che si occupa, grazie all’ utilizzo della libreria HTTParty di chiamare le api di facebook e utilizzando un ulteriore metodo privato denominato load_or_create di interagire con il database, come mostrato nelle figure sottostanti.(viene mostrato solo la gestione del model place per semplicità)
(Figura 2.7)(Figura 2.8)

```ruby
def getFaceGraph(param_face)
  param_face.each do |key,value|
    response=self.class.get value[:url] , value[:option]
    puts key
    #prova e vedi
    if response.success?

      if key == :place
        #puts response["data"].to_s
        array= response["data"]
        array.each do |i|
          i.each do |key1,value1|
            if key1 == "place"
              location=i[key1]["location"]
              name=location["city"]
              lat=location["latitude"]
              lng=location["longitude"]
              hash={:user_id => @user.id,:lat =>lat , :lng => lng, :name=> name}
              load_or_create(Place,hash)
            end
          end
        end
      end
      .
      .
      .
```

Graph_Api:Figura 2.7

```ruby
def load_or_create( name_class, hash )
  obj=name_class.find_by(hash)

  if obj.nil?

    obj=name_class.new(hash)

    if obj.valid?

      obj.save
      obj

    else
      nil
    end

  else
    obj
  end

end
```

Graph_Api:Figura 2.8

2.1.2 SendGrid

Per quanto riguarda la registrazione dell' utente senza l utilizzo di Facebook, è stato utilizzato SendGrid, per l' invio della mail di conferma in modo da poter verificare l'account dell' utente.

In (Figura 2.9) possiamo notare la configurazione per impostare SendGrid in produzione.

```
Rails.application.configure do
  # Settings specified here will take precedence over those in config/application.rb.

  # Code is not reloaded between requests.
  config.cache_classes = true

  config.action_mailer.raise_delivery_errors = true
  config.action_mailer.delivery_method = :smtp
  host = 'triptrack.herokuapp.com'
  config.action_mailer.default_url_options = { host: host }
  ActionMailer::Base.smtp_settings = {
    :address        => 'smtp.sendgrid.net',
    :port           => '587',
    :authentication => :plain,
    :user_name      => ENV['SENDGRID_USERNAME'],
    :password       => ENV['SENDGRID_PASSWORD'],
    :domain         => 'heroku.com',
    :enable_starttls_auto => true
  }
```

SendGrid:Figura 2.9

All' atto della creazione dell' utente viene chiamato il metodo d'istanza "send_activation_email" dello User model.(Figura 2.10)(Figura 2.11)

```ruby
def create
  @user = User.new(user_params)
  if @user.save
    @user.send_activation_email
    flash[:info] = "Please check your email to activate your account."
    redirect_to root_url
  else
    render 'new'
  end
end
```

SendGrid:Figura 2.10

```ruby
# Sends activation email.
def send_activation_email
  UserMailer.account_activation(self).deliver_now
end

def send_password_reset_email
  UserMailer.password_reset(self).deliver_now
end
```

SendGrid:Figura 2.11

Infine mostriamo lo UserMailer (Figura 2.12)che è colui che impacchetta la mail andando a riempire il body con la vista sottostante(account_activation.html.erb) (Figura 2.13)

```ruby
class UserMailer < ApplicationMailer

  def account_activation(user)
    @user = user
    mail to: user.email, subject: "Account activation"
  end

  def password_reset(user)
    @user = user
    mail to: user.email, subject: "Password reset"
  end
end
```

SendGrid:Figura 2.12

```
<h1>Sample App</h1>

<p>Hi <%= @user.name %>,</p>

<p>
Welcome to the Sample App! Click on the link below to activate your account:
</p>

<%= link_to "Activate", edit_account_activation_url(@user.activation_token,
                                                    email: @user.email) %>
```

SendGrid:Figura 2.13

2.2 Notifica real time

E’ stato inoltre sviluppato un servizio di notifica real time, per quanto riguarda la aggiunta o la rimozione di un amicizia tra due utenti..

2.2.1 Redis

Come message-broker è stato utilizzato Redis. Redis è un data-structure store che viene utilizzato come db non relazionale, cache o come nel nostro caso message broker. All’ atto dell’ operazione di add o delete viene pushato nella coda un messaggio di notifica. Questa coda è sottoposta a un interrogazione asincrona da parte del miniserver in node.js per ogni utente.

Per semplicità verrà mostrata solamente la gestione dell’aggiunta di un amicizia in quanto la gestione della rimozione è analoga.(Figura 2.14)

```
def create
  @friend = User.find(params[:friend_id])
  if @friend.super_user?
    redirect_to(root_url)
    return
  end
  current_user.require_friend(@friend)
  msg = { resource: current_user.name+" has just sent you a request friend",
          friend_id: params[:friend_id],
          user_id: current_user.id
        }
  $redis.publish 'rt-change', msg.to_json
  redirect_to request.referrer

end
```

Redis:Figura 2.14

2.2.2 Node.js e Websocket

Come detto è stato utilizzato un miniserver in Node.js per verificare il contenuto del message broker . Node.js è un runtime Javascript costruito sul motore JavaScript V8 di Chrome. Node.js usa un modello I/O non bloccante e ad eventi, che lo rendono un framework leggero ed efficiente. La scelta di integrare un miniserver in node.js è stata valutata sotto queste ipotesi:

- semplicità di integrazione con i websockets
- necessità di operazioni asincrone e veloci sia per la lettura dal message-broker sia per per l' interazione con i websockets.

Il WebSocket è una tecnologia web che fornisce canali di comunicazione full-duplex attraverso una singola connessione TCP. L'API del WebSocket è stata standardizzata dal W3C e il protocollo WebSocket è stato standardizzato dall'IETF.
Il protocollo WebSocket permette maggiore interazione tra un browser e un server, facilitando la realizzazione di applicazioni che forniscono contenuti e giochi in tempo reale.Per la realizzazione del protocollo websocket è stata utilizzata la libreria Socket.io.

Ogni volta che un utente effettua un login , dal lato client viene instaurata una connessione con il websocket definito dal miniserver. A questo punto e per ogni utente connesso, il miniserver controlla che il message-broker non abbia nuove entry
E in contemporanea, lato client, si è in attesa di ricevere una notifica dal miniserver,sempre in modalità asincrona.(Figura 2.15)(Figura 2.16)

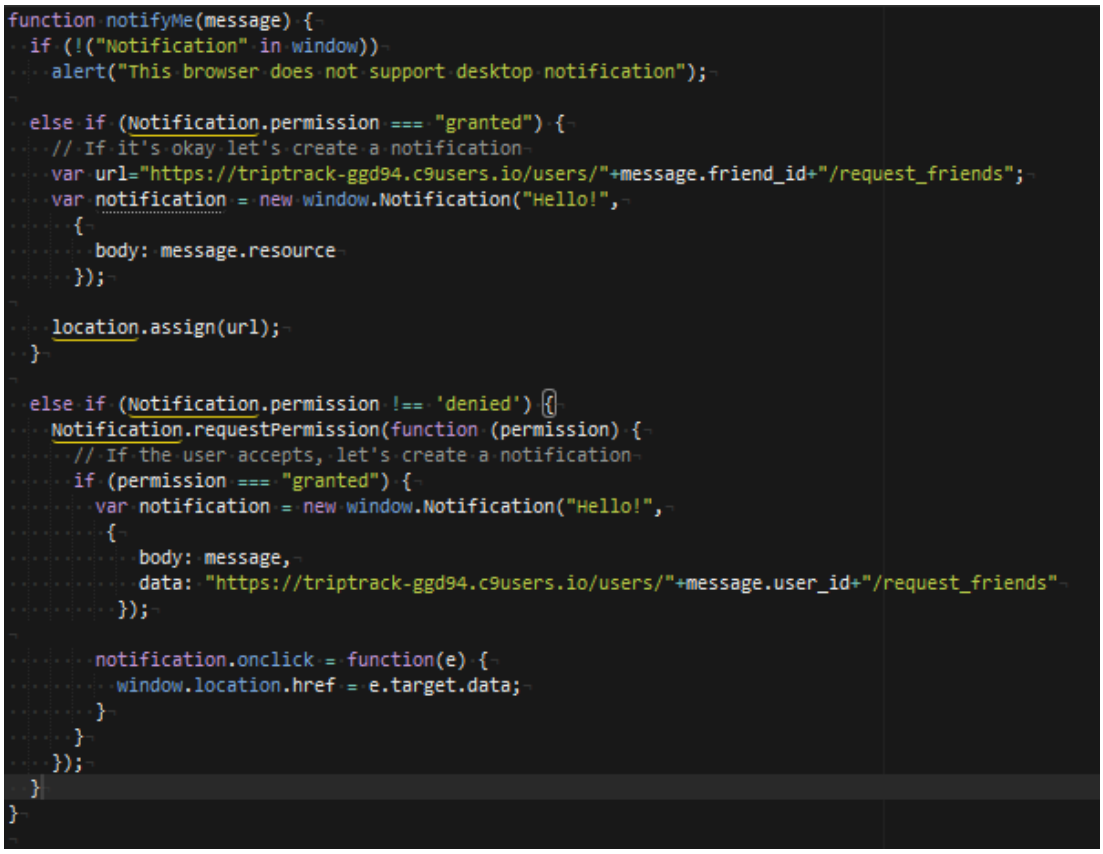
```
// module dependencies
var http = require("http"),
    sio = require("socket.io");
// create http server
var server = http.createServer().listen(8081, process.env.IP);
// create socket server
var io = sio.listen(server);
var redis = require('redis').createClient();
redis.subscribe('rt-change');
io.on('connection', function(socket){
  console.log("connect");
  redis.on('message', function(channel, message){
    socket.emit('rt-change', JSON.parse(message));
  });
});
```

Node e WebSocket: Figura 2.15

```
function listen(id){
  socket = io.connect("https://triptrack-ggd94.c9users.io:8081",{secure:true});
  socket.on("rt-change", function(message){
    if( id == message.friend_id)
      notifyMe(message)
  });
}
```

Node e WebSocket: Figura 2.16

Non appena il message-broker riceve un nuovo messaggio dal controller rails, come visto precedentemente , il miniserver inoltra quel messaggio al websocket del diretto interessato, in ascolto sul client. Ricevuto il messaggio, per la visualizzazione realtime sono state utilizzate le web notification compatibili con HTML5 ed i moderni browser escluso Microsoft Edge.(Figura 2.17)



```
function notifyMe(message) {
  if (!("Notification" in window))
    alert("This browser does not support desktop notification");

  else if (Notification.permission === "granted") {
    // If it's okay let's create a notification
    var url="https://triptrack-ggd94.c9users.io/users/"+message.friend_id+"/request_friends";
    var notification = new window.Notification("Hello!",
      {
        body: message.resource
      });

    location.assign(url);
  }

  else if (Notification.permission !== 'denied') {
    Notification.requestPermission(function (permission) {
      // If the user accepts, let's create a notification
      if (permission === "granted") {
        var notification = new window.Notification("Hello!",
          {
            body: message,
            data: "https://triptrack-ggd94.c9users.io/users/"+message.user_id+"/request_friends"
          });

        notification.onclick = function(e) {
          window.location.href = e.target.data;
        }
      }
    });
  }
}
```

Web Notification: Figura 2.17

2.3 Development Team

Il progetto è stato realizzato e sviluppato da:

- Massimiliano Cestra
- Giulia Giugno

Dopo aver effettuato un analisi preliminare e aver definito le risorse necessarie allo sviluppo dell'intero progetto TripTrack, gli studenti hanno cercato di realizzare l'applicazione secondo le linee guida tracciate nella raccolta dei requisiti.

2.3.1 Divisione del lavoro

Per quanto riguarda la divisione del lavoro, la mia collega Giulia ha sviluppato tutta la gestione con le Api di Google Maps e l'utilizzo, quindi l'integrazione nell'applicazione del cloud storage S3 di Amazon , mentre io mi sono preoccupato di integrare Devise nello User Model, per l'Oauth 2.0 con Facebook e l'estrazioni dei dati dell'utente, come foto, posti e feed, dalle Graph Api di Facebook.
Nel continuo della tesi la collega Giulia si è preoccupata di gestire l'architettura del sistema e il Presentation Layer, mentre io mi sono preoccupato della gestione del Data Layer e dell'Application Layer.

Il team, durante lo svolgimento del progetto, ha svolto continui meeting al fine di revisionare il lavoro svolto, risolvere eventuali problemi e pianificare il lavoro restante. Generalmente sono stati pianificati sprint (periodi di tempo in cui il lavoro doveva essere completato) giornalieri, in modo da tenere il lavoro sotto controllo e ripianificare, nel caso fosse stato necessario, le attività dei giorni seguenti.

2.3.2 Development Tools

2.3.2.1 Git e Bitbucket

Per la realizzazione dell'applicazione, il team ha optato per il tool Git il quale è in grado di risolvere nettamente i classici problemi di sviluppo del software come backup , organizzazione e collaborazione.
Git come ogni altro controllo di versione permette infatti di gestire la storia dei cambiamenti apportati ad ogni singolo file e nel caso si vuole, di recuperare qualsiasi versione precedente.

Inoltre permette di gestire uno sviluppo concorrente e di avere un'evoluzione non lineare del codice grazie al branching.
La scelta è ricaduta su Git in quanto, a differenza di altri controlli di versione, è open source ed è un controllo di versione distribuito, che è nettamente la scelta migliore.
Infatti esistono 3 tipi di controllo di versione:

- Locale (Database installato in locale)
- Centralizzato (Unico Server che contiene tutte le versioni dei file)
- Distribuito (Unione del locale e centralizzato)

Mentre il Locale e il Centralizzato, possono soffrire di problemi come guasti o problemi di rete, la tipologia Distribuita è nettamente più robusta.
Grazie a Git abbiamo potuto lavorare in parti del progetto separate, creando dei branch di sviluppo, testare il funzionamento e poi riagganciare il tutto al branch principale(Master) destinato alla produzione.
Come servizio di hosting web based per il repository remoto la scelta è andata verso Bitbucket. Bitbucket, a differenza del suo cugino più famoso Github offre repository privati gratuiti, tuttavia nel nostro caso la scelta era equivalente.

2.3.2.2 Cloud9

Per la realizzazione dell'applicazione come IDE la scelta è ricaduta su Cloud9.
Come dice il nome stesso Cloud9 è un ambiente di sviluppo integrato in cloud.
Quest'ultimo si è dimostrato veramente utile per la nostra applicazione in fase di sviluppo offrendo chat in tempo reale e tutte le caratteristiche necessarie che un ide dovrebbe fornire agli sviluppatori .Cloud9 oltre ad essere un IDE innovativo è, anche se a tempo limitato ma gratuito, un platform as a service(PAAS)che permette, di hostare la nostra applicazione, al fine di poterla farla provare ai vari tester. Unendo Cloud9 e Git, abbiamo potuto lavorare a distanza, sulla stessa istanza del progetto, su branch separati, oppure utilizzando la tecnica Agile "progamming pair" a distanza.

2.3.2.3 HEROKU

Per quanto riguarda la produzione, è stato scelto il platform as a service Heroku. Heroku offre gratuitamente, un servizio attivo 24/24 con buone caratteristiche hardware e ragionevoli limiti di richieste e accessi da parte dei clienti.

L'utilizzo è semplicissimo grazie a un interfaccia a riga di comando stile Unix Like che permette il controllo remoto e compatibile, quindi, a tutti gli effetti con Cloud9 che offre una macchina con sistema operativo Linux.

La condivisione del codice, avviene attraverso la creazione di un ulteriore repository Git all'interno della macchina offerta da heroku e nel nostro caso, dopo aver testato il tutto sul cloud9, il codice viene pushato prima sul repository remoto Bitbucket e poi sul repository remoto di Heroku. In questo modo, abbiamo separato nettamente le varie fasi di realizzazione del codice come test , sviluppo e produzione .

2.3.2.4 SCRUMDO

Scrumdo è stato utilizzato per mantenere in costante aggiornamento ogni membro del team di sviluppo, controllando e aggiornando costantemente il work flow quotidiano. In questo modo ogni membro del team aveva accesso ad una "board", la quale aveva l'obbligo di aggiornare ogni volta che iniziasse, terminasse o rivedesse un'attività. Inizialmente tutte le features sono state inseriti tra le attività da svolgere e man mano, ognuna di esse in base al suo stato (doing, reviewing, done) veniva spostata nella colonna più adatta. Questo approccio ha contibuito ad avere una visione più chiara del da farsi giorno per giorno.

2.4 Effort time

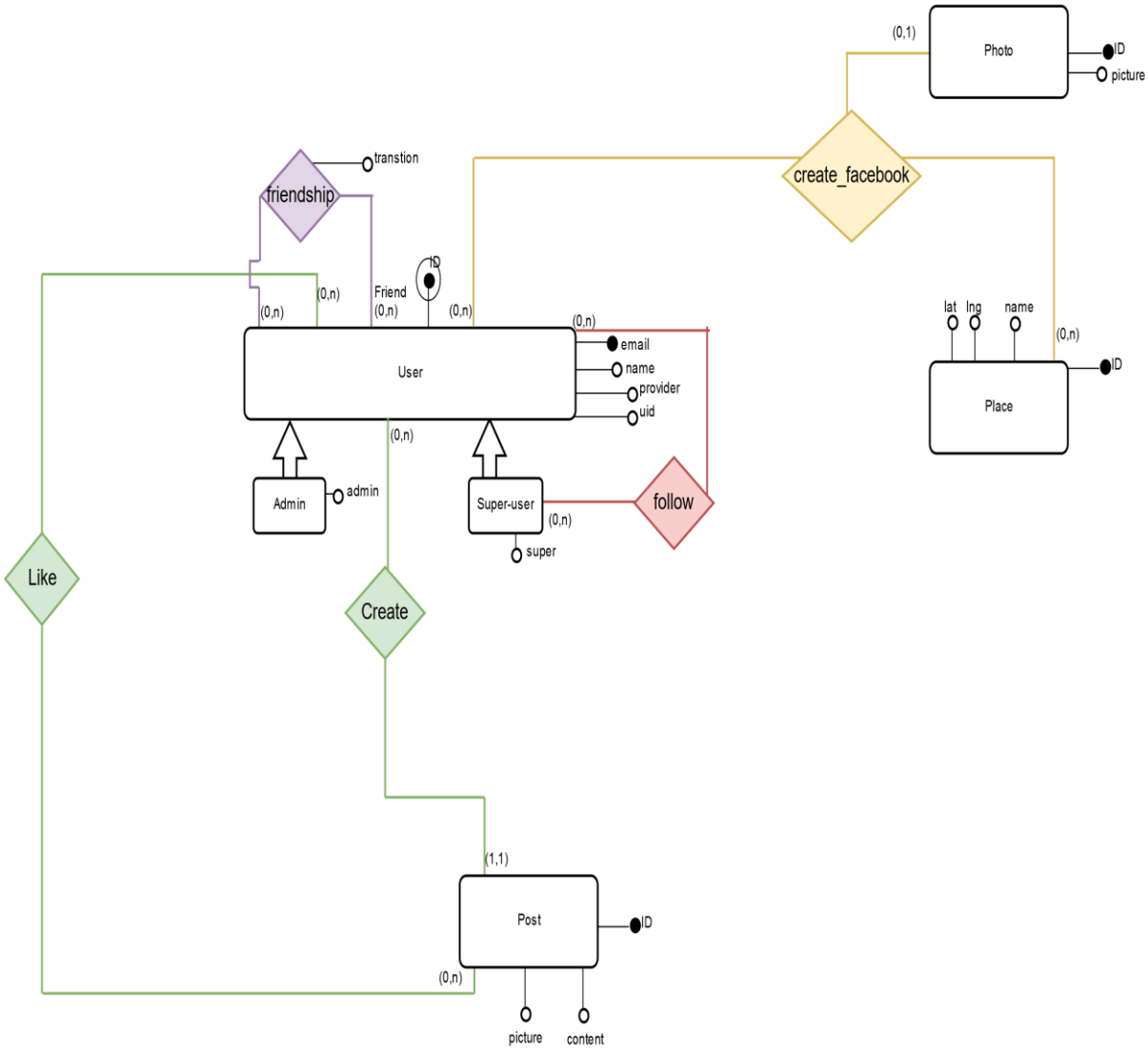
Sviluppando l'applicazione in un team, è stato necessario pianificare dettagliatamente il lavoro da svolgere. Dapprima è stato redatto un "project overview" (Appendix : 1.1 Project Overview) al fine di studiare ad ampio raggio il tempo necessario per portare a compimento l'intero progetto. Una volta che l'intero

team ha approvato il project overview, si è proceduto a stipulare il daily planning (Appendix: 1.2 Daily Planning), al fine di definire meglio gli obiettivi giornalieri, in modo da rispettare gli sprint nei tempi programmati.
Ogni membro del gruppo ha collaborato in modo ottimale per poter rispettare il work flow giornaliero, cosi che non è stato in alcun modo necessario dover modificare la pianificazione del lavoro.
Infatti, a progetto concluso l'effort time per lo sviluppo del lavoro da svolgere è risultato ottimale in quanto coincidente con la durata effettiva di ogni attività sviluppata.

Chapter 3

Analisi concettuale del sistema

3.1 ER SCHEMA



3.2 Glossario dei dati

3.2.1 Glossario delle entità

ENTITY	DESCRIPTION	ATTRIBUTES	IDENTIFIERS
USER	APPLICATION'S CLIENTS	ID NAME EMAIL CREATED_AT UPDATED_AT PASSWORD_DIGEST REMEMBER_DIGEST ACTIVATION_DIGEST ACTIVATED ACTIVATED_AT RESET_DIGEST RESET_SENT_AT PROVIDER UID FACEBOOK_DIGEST	ID(PK) EMAIL
ADMIN	APPLICATION'S ADMIN. IS-A USER	ADMIN	ID
SUPERUSER	SPECIAL USER WITH MORE SKILL. AN USER BECAME SUPER USER AFTER THAT HE GOT ALMOST 50 FRIENDS. IS-A USER.	SUPERUSER	ID
POST	APPLICATION'S USERS INSERT THE POSTS	ID CONTENT PICTURE	ID
PLACE	PLACES VISITED BY USERS	ID LATITUDINE LONGITUDINE NAME	ID(PK) [LATITUDINE, LONGITUDINE]
PHOTO	USERS' PHOTOS GOT THROUGHT FACEBOOK	ID PICTURE	{ID}

3.2.2 Glossario delle relazioni

RELATIONSHIP	DESCRIPTION	COMPONENTS	ATTRIBUTES	IDENTIFIERS
FRIENDSHIP	FRIENDSHIP WITH ANOTHER USER	[USER,USER]	TRANSITION	IMPLICIT
FOLLOW	A USER FOLLOW A SUPER USER	[USER,SUPERUSER]		IMPLICIT
LIKE	A USER CAN LIKE A POST	[USER,POST]		IMPLICIT
CREATE	A USER CAN CREATE A POST	[USER,POST]		IMPLICIT
CREATE_FACEBOOK	A USER COULD HAVE CREATED ON FACEBOOK A FEED WITH PLACE AND PHOTO	[USER,PHOTO,PLACE]		IMPLICIT

RELATIONSHIP	CARDINALITY	MOTIVATION
FRIENDSHIP	[(0..N),(0..N)]	A USER CAN HAVE FRIENDS BUT HE ISN'T OBLIGED
FOLLOW	[(0..N),(0..N)]	A USER CAN FOLLOW THE SUPERUSERS BUT HE ISN'T OBLIGED
LIKE	[(0..N),(0..N)]	A USER CAN LIKE THE POSTS BUT HE ISN'T OBLIGED
CREATE	[(0..N),(1..1)]	A USER CAN CREATE DIFFERENT POSTS,A POST IS CREATED BY ONLY ONE USER
CREATE_FACEBOOK	[(0..N),(1..1),(0..N)]	A USER HAS DIFFERENT PHOTOS WITH DIFFERENT PLACES, A PLACE HAS DIFFERENT USERS WITH DIFFERENT PLACES A PHOTO HAS ASSOCIATED ONLY PAIR [USER,PLACE]

3.2.3 Glossario degli attributi

ATTRIBUTES	ENTITY/REL.	DOMAIN	DESCRIPTION
ID	USER	INTEGER	PRIMARY KEY, NOT NULL FOR THE ENTITY USER
NAME	USER	VARCHAR	USER'S NAME NOT NULL
EMAIL	USER	VARCHAR	USER'S EMAIL
CREATED_AT	USER	DATETIME	TIME OF CREATION NOT NULL
UPDATED_AT	USER	DATETIME	TIME OF UPDATE NOT NULL
PASSWORD_DIGEST	USER	VARCHAR	DIGEST OF USER'S PASSWORD
REMEMBER_DIGEST	USER	VARCHAR	DIGEST OF USER'S REMEMBER_TOKEN
ACTIVATION_DIGEST	USER	VARCHAR	DIGEST OF USER'S ACTIVATION_TOKEN
ACTIVATED	USER	BOOLEAN	BOOLEAN TO VERIFY THAT THE USER'S ACCOUNT IS ACTIVATED
ACTIVATED_AT	USER	DATETIME	TIME OF ACCOUNT ACTIVATION
RESET_DIGEST	USER	VARCHAR	DIGEST OF USER'S RESET_TOKEN
RESET_SENT_AT	USER	VARCHAR	SENDING TIME OF THE TOKEN RESET
PROVIDER	USER	VARCHAR	PROVIDER FOR EXTERNAL SERVICES (FACEBOOK OR NULL)
UID	USER	VARCHAR	USER ID FOR EXTERNAL SERVICES
FACEBOOK_DIGEST	USER	VARCHAR	DIGEST OF USER'S FACEBOOK_TOKEN
ADMIN	USER/ADMIN	BOOLEAN	BOOLEAN TO VERIFY THAT THE USER IS AN ADMIN
SUPERUSER	USER/SUPERUSER	BOOLEAN	BOOLEAN TO VERIFY THAT THE USER IS A SUPER USER
SUPERUSER	USER/SUPERUSER	BOOLEAN	BOOLEAN TO VERIFY THAT THE USER IS A SUPER USER
ID	POST	INTEGER	PRIMARY KEY NOT NULL FOR THE ENTITY POST
CONTENT	POST	TEXT	POST'S CONTENT
CREATED_AT	POST	DATETIME	TIME OF CREATION NOT NULL

UPDATED_AT	POST	DATETIME	TIME OF UPDATING NOT NULL
PICTURE	POST	VARCHAR	AMAZON'S URL TO VIEW THE PHOTOS
ID	PHOTO	INTEGER	PRIMARY KEY NOT NULL FOR THE ENTITY PHOTO
ID	PHOTO	INTEGER	PRIMARY KEY NOT NULL FOR THE ENTITY PHOTO
PICTURE	PHOTO	VARCHAR	FACEBOOK'S URL TO VIEW THE PHOTOS
CREATED_AT	PHOTO	VARCHAR	TIME OF CREATION NOT NULL
UPDATED_AT	PHOTO	VARCHAR	TIME OF UPDATING NOT NULL
ID	PLACE	INTEGER	PRIMARY KEY NOT NULL FOR THE ENTITY PLACE
LATITUDE	PLACE	FLOAT	PLACE LATITUDE
LONGITUDE	PLACE	FLOAT	PLACE LONGITUDE
NAME	PLACE	VARCHAR	PLACE NAME
CREATED_AT	PLACE	DATETIME	TIME OF CREATION NOT NULL
UPDATED_AT	PLACE	DATETIME	TIME OF UPDATING NOT NULL
ID	FRIENDSHIP	INTEGER	PRIMARY KEY NOT NULL FOR THE RELATIONSHIP FRIENDSHIP
USER_ID	FRIENDSHIP	INTEGER	USER THAT SENT THE FRIEND REQUEST
FRIEND_ID	FRIENDSHIP	INTEGER	USER THAT RECEIVED THE FRIEND REQUEST
CREATED_AT	FRIENDSHIP	VARCHAR	TIME OF CREATION NOT NULL
UPDATED_AT	FRIENDSHIP	VARCHAR	TIME OF UPDATING NOT NULL
TRANSITION	FRIENDSHIP	VARCHAR	CURRENT STATE OF FRIENDSHIP
ID	LIKE	INTEGER	PRIMARY KEY NOT NULL FOR THE RELATIONSHIP LIKE
USER_ID	LIKE	INTEGER	USER THAT LIKE THE POST
POST_ID	LIKE	INTEGER	POST THAT LIKED FROM USER

CREATED_AT	LIKE	DATETIME	TIME OF CREATION NOT NULL
UPDATED_AT	LIKE	DATETIME	TIME OF UPDATING NOT NULL
ID	FOLLOW	INTEGER	PRIMARY KEY NOT NULL FOR THE RELATIONSHIP FOLLOW
FOLLOWER_ID	FOLLOW	INTEGER	...
FOLLOWED_ID	FOLLOW	INTEGER	...
CREATED_AT	FOLLOW	DATETIME	TIME OF CREATION NOT NULL
UPDATED_AT	FOLLOW	DATETIME	TIME OF UPDATING NOT NULL
ID	CREATE	INTEGER	PRIMARY KEY NOT NULL FOR THE RELATIONSHIP FOLLOW
USER_ID	CREATE	INTEGER	...
PLACE_ID	CREATE	INTEGER	...
POST_ID	CREATE	INTEGER	...
CREATED_AT	CREATE	DATETIME	TIME OF CREATION NOT NULL
UPDATED_AT	CREATE	DATETIME	TIME OF UPDATING NOT NULL
ID	CREATE_FACEBOOK	INTEGER	PRIMARY KEY NOT NULL FOR THE RELATIONSHIP CREATE_FACEBOOK
USER_ID	CREATE_FACEBOOK	INTEGER	...
PLACE_ID	CREATE_FACEBOOK	INTEGER	...
PHOTO_ID	CREATE_FACEBOOK	INTEGER	...
CREATED_AT	CREATE_FACEBOOK	DATETIME	TIME OF CREATION NOT NULL
UPDATED_AT	CREATE_FACEBOOK	DATETIME	TIME OF UPDATING NOT NULL NOT NULL

3.2.4 Vincoli esterni

- Uno User può seguire un super user, ma non vale il viceversa
- Non possono esistere due istanze delle tabelle friendship dove user_id delle prima istanza è uguale a friend_id della seconda istanza E friend_id della prima è uguale a user_id della seconda
- Un super user può aggiungere uno User solo se quest'ultimo non lo segue già
- Un user può seguire un super User solo se non sono già amici

Chapter 4

Progettazione del sistema

4.1 Architettura del sistema

Diagrammi UML progettati e creati dalla collega Giulia Giugno.

4.2 Progettazione logica del Data Layer

La progettazione logica del data layer, è avvenuta con l'utilizzo di Rails. Una delle caratteristiche ben apprezzate del framework Rails, è la possibilità di interagire con un qualsiasi database, attraverso un'unica interfaccia, nascondendo al programmatore tutte le eterogeneità del sistema sottostante.
Rails permette di creare i Model come se fossero delle semplice classi, ma in realtà esse estendono la classe ApplicationRecord, che ha già tutti i metodi necessari per poter interagire con qualsiasi Database. Una volta generato il Model, Rails permette di creare una migrazione al db, per garantire coerenza tra application layer e data layer.. Il procedimento è semplice ed istantaneo e dopo ogni migrazione, Rails aggiorna il file schema.rb, che rappresenta lo script che permette di creare o aggiornare le tabelle del db.
Di seguito mostriamo alcuni figure per la progettazione logica del database ad esempio per il model User.

Nella (Figura 4.1) , possiamo osservare il model User.rb, dove sono state inserite tutte le varie associazioni con gli altri model ,così che, all'atto della migrazione,Rails vado a gestire le relative relazioni e entità con il db.

Osserviamo inoltre, che Rails permette di gestire con semplicità alcuni vincoli, come il not null o la lunghezza di una stringa, o la particolare forma di una email direttamente nell'Application Layer.

```ruby
class User < ApplicationRecord

  has_many :places, dependent: :destroy
  has_many :notifies
  has_many :likes, dependent: :destroy
  has_many :liked_posts, :through => :likes, :source => :micropost
  has_many :friendships
  has_many :friends, :through => :friendships
  has_many :microposts, dependent: :destroy
  has_many :active_relationships, class_name:  "Relationship",
                                  foreign_key: "follower_id",
                                  dependent:   :destroy
  has_many :following, through: :active_relationships, source: :followed
  has_many :passive_relationships, class_name:  "Relationship",
                                   foreign_key: "followed_id",
                                   dependent:   :destroy
  has_many :followers, through: :passive_relationships, source: :follower

  attr_accessor :remember_token, :activation_token, :reset_token, :facebook_token

  before_save   :downcase_email
  before_create :create_activation_digest

  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i

  validates :name,  presence: true, length: { maximum: 50 }
  validates :email, presence: true, length: { maximum: 255 },
                    format: { with: VALID_EMAIL_REGEX },
                    uniqueness: { case_sensitive: false }
  validates :password, presence: true, length: { minimum: 6 }, allow_nil: true

  has_secure_password

  devise :omniauthable, :omniauth_providers => [:facebook]

```

Data_Layer:Figura 4.1

Per generare il Model, inizialmente è possibile utilizzare lo script "rails generate model' ,così che, vengono generati direttamente anche le migrazioni e files necessari. Per andare ad effettuare le modifiche nel db, è possibile utilizzare lo script "rails db:migrate " e ottenere il file "schema.rb" .(Figura 4.2)(Figura 4.3)(Figura 4.4)

```
$ rails generate model User name:string email:string
      invoke  active_record
      create    db/migrate/20160523010738_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
```

Data_Layer: Figura 4.2

```
class CreateUsers < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      t.string :name
      t.string :email

      t.timestamps
    end
  end
end
```

Data_Layer: Figura 4.3

```
create_table "users", force: :cascade do |t|
  t.string   "name"
  t.string   "email"
  t.datetime "created_at",                          null: false
  t.datetime "updated_at",                          null: false
  t.string   "password_digest"
  t.string   "remember_digest"
  t.boolean  "admin",                default: false
  t.string   "activation_digest"
  t.boolean  "activated",            default: false
  t.datetime "activated_at"
  t.string   "reset_digest"
  t.datetime "reset_sent_at"
  t.string   "provider"
  t.string   "uid"
  t.string   "facebook_digest"
  t.boolean  "super_user",           default: false
  t.index ["email"], name: "index_users_on_email", unique: true
end
```

Data_Layer:Figura 4.4(Configurazione finale)

Chapter 5

Realizzazione del sistema

5.1 Application Layer

5.1.1 User

METHOD	ROUTE	ACTION	PARAM	DESCRIPTION
GET	/SIGNUP	NEW		RENDER NEW
POST	/SIGNUP	CREATE	NAME EMAIL PASSWORD PASSWORD_CONFIRMATION	CREATE A NEW USER THEN SENDS A EMAIL, IF IT'S OK REDIRECT TO ROOT URL ELSE RENDER NEW
GET	/USERS	INDEX		RENDER INDEX IF USER IS LOGGED_IN ELSE RENDER NEW
GET	/USERS/:ID/FOLLOWING	FOLLOWING	ID	RENDER SHOW_FOLLOW IF USER IS LOGGED_IN ELSE RENDER NEW
GET	/USERS/:ID/FOLLOWERS	FOLLOWERS	ID	IF CURRENT_USER IS A SUPER_USER AND LOGGED_IN RENDER SHOW_FOLLOW ELSE REDIRECT TO ROOT_URL
GET	/USERS/:ID/FRIENDS	FRIENDS	ID	IF CURRENT_USER IS LOGGED_IN RENDER SHOW_FRIENDS ELSE RENDER NEW
GET	/USERS /:ID/REQUEST_FRIENDS	REQUEST_FRIENDS	ID	IF CURRENT_USER IS LOGGED_IN RENDER SHOW_REQUEST ELSE RENDER NEW
GET	/USERS/:ID/EDIT	EDIT	ID	IF CURRENT_USER IS LOGGED_IN AND IS THE CORRECT_USER RENDER EDIT ELSE RENDER NEW

GET	/USERS/:ID	SHOW	ID	RENDER SHOW
PATCH/PUT	/USERS/:ID	UPDATE	ID	IF CURRENT_USER IS LOGGED_IN REDIRECT TO SHOW ELSE RENDER EDIT
DELETE	/USERS/:ID	DESTROY	ID	IF CURRENT_USER IS ADMIN REDIRECT INDEX ELSE RENDER ROOT_URL
GET	/USERS /AUTH/FACEBOOK	PASSTHRU		REDIRECT TO FACEBOOK LOGIN
POST	/USERS /AUTH/FACEBOOK /CALLBACK	FACEBOOK	UID AND TOKEN	REDIRECT TO SHOW

5.1.2 Posts

METHOD	ROUTE	ACTION	PARAM	DESCRIPTION
POST	/MICROPOSTS	CREATE	CONTENT , PICTURE	RENDER ROOT_URL
GET	/MICROPOSTS /:ID/LIKE	LIKE	ID	IF CURRENT_USER IS LOGGED_IN RENDER SHOW_LIKES ELSE RENDER NEW
DELETE	/MICROPOSTS/:ID	DESTROY	ID	IF CURRENT_USER IS CORRECT_USER OR ADMIN REDIRECT TO PRECEDENT URL ELSE RENDER NEW

5.1.3 Relationships

METHOD	ROUTE	ACTION	PARAM	DESCRIPTION
POST	/RELATIONSHIPS	CREATE	FOLLOWED_ID	IF CURRENT_USER IS LOGGED_IN AND FOLLOWED USER IS SUPER_USER REDIRECT_TO PRECEDENT URL ELSE RENDER NEW
DELETE	/RELATIONSHIPS/:ID	DESTROY	ID	IF CURRENT_USER IS LOGGED_IN REDIRECT_TO PRECEDENT URL ELSE RENDER NEW

5.1.4 Friendships

METHOD	ROUTE	ACTION	PARAM	DESCRIPTION
POST	/FRIENDSHIPS	CREATE	FRIEND_ID	IF CURRENT_USER IS LOGGED_IN AND OTHER USER ISN'T SUPER USER REDIRECT_TO PRECEDENT URL AND UPDATE THE COLUMN TRANSITION WITH VALUE "PENDING" ELSE RENDER NEW
DELETE	/FRIENDSHIPS/:ID	DESTROY	ID , OTHER_USER ID	IF CURRENT_USER IS LOGGED_IN RENDER PRECEDENT URL ELSE RENDER NEW
PATCH/PUT	/FRIENDSHIPS/:ID	UPDATE	ID	IF CURRENT_USER IS LOGGED_IN UPDATE THE COLUMN TRANSITION WITH VALUE "ACCEPTED" AND REDIRECT_TO PRECEDENT URL ELSE RENDER NEW

5.1.5 Likes

METHOD	ROUTE	ACTION	PARAM	DESCRIPTION
POST	/LIKES	CREATE	POST_ID , USER_ID	IF CURRENT_USER IS LOGGED_IN REDIRECT TO PRECEDENT URL ELSE RENDER LOGIN_URL

5.1.6 Route without Model

Helper	HTTP Verb	Path	Controller#Action
Path / Url		Path Match	
password_resets_new_path	GET	/password_resets/new(.:format)	password_resets#new
password_resets_edit_path	GET	/password_resets/edit(.:format)	password_resets#edit
root_path	GET	/	static_pages#home
about_path	GET	/about(.:format)	static_pages#about
contact_path	GET	/contact(.:format)	static_pages#contact
login_path	GET	/login(.:format)	sessions#new
	POST	/login(.:format)	sessions#create
logout_path	DELETE	/logout(.:format)	sessions#destroy
edit_account_activation_path	GET	/account_activations/:id/edit(.:format)	account_activations#edit
password_resets_path	POST	/password_resets(.:format)	password_resets#create
new_password_reset_path	GET	/password_resets/new(.:format)	password_resets#new

edit_password_reset_path	GET	/password_resets/:id/edit(.:format)	password_resets#edit
password_reset_path	PATCH	/password_resets/:id(.:format)	password_resets#update
	PUT	/password_resets/:id(.:format)	password_resets#update

5.2 Presentation Layer

Sequence Diagram creati e progettati dalla collega Giulia Giugno.

Chapter 6

Deployment & Validation

Quest' ultima sezione del documento, vuole porre l'attenzione su come l'intero sistema sviluppato può essere installato per poterlo utilizzare, e spiegare in modo dettagliato il piano test seguito per lo sviluppo.

6.1 Download Source Code

Al fine di verificare il funzionamento di TripTrack, descritto nei capitoli precedenti, è possibile seguire i seguenti step:

1. Creare una nuova applicazione su "https://c9.io", scegliendo come opzione "Rails tutorial"
2. Effettuare il clone dal repository remoto, situato su bitbucket, lanciando da riga di comando:
   "git clone https://ggd94@bitbucket.org/ggd94/triptrack.git"

Per avviare l'applicazione,invece:

3. Entrare nella directory "realtime", digitando il seguente comando:
   "cd sample_app/realtime"
4. Una volta entrati nella cartella realtime, bisogna procedere ad avviare il server redis e quello in node:
   -"redis-server"
   -"node realtime-server.js"
5. Impostare le keys di Facebook nelle variabili d'ambiente definite in config/devise.rb
6. Infine installare le gemme e avviare l'applicazione:
   "bundle install"
   "rails server -b $IP -p $PORT"

6.1 Validation testing

I test di validazione verificano che il prodotto rispetti in modo veritiero i requisiti del committente. In questo caso, per massimizzare il risultato ottenuto dalla fase di testing, essi sono stati suddivisi in:

1. Unit testing, che verificano la correttezza di piccole porzioni di programma che talvolta risultano essere incomplete (model)
2. Integration testing, si occupano di verificare i requisiti performanti e funzionali dell'applicazione
3. Others testing

Nel caso di TripTrack, utilizzando un framework rails, attraverso la creazione iniziale di una nuova applicazione, lo scaffolding ha automaticamente creato, all'interno della cartella test, una suddivisione in sottocartelle ed i test effettuati per ognuna risultano:

- Controllers:
    - Account activations test
    - Friendship test
    - Likes test
    - Posts test
    - Relationship test
    - Sessions test
    - Static pages test
    - User controller test
- Helpers
    - Application helper test
    - Sessions helper test
- Integration
    - Facebook test
    - Following test
    - Friends test

    - Posts inerface test
    - Password reset test
    - Site layout test
    - Super users test
    - User edit test
    - User index test
    - User login test
    - User profile test
    - User signup test
- Mailers
    - User_mailer_test
- Models
    - Friendship test
    - Like test
    - Post test
    - Photo test
    - Place test
    - Relationship test
    - User test

Ogni modello al fine di testare le funzionalità e i vincoli progettati, ha nella sezione test associata una fixture, al fine di creare delle istanze fittizie utilizzabili nella fase di testing.

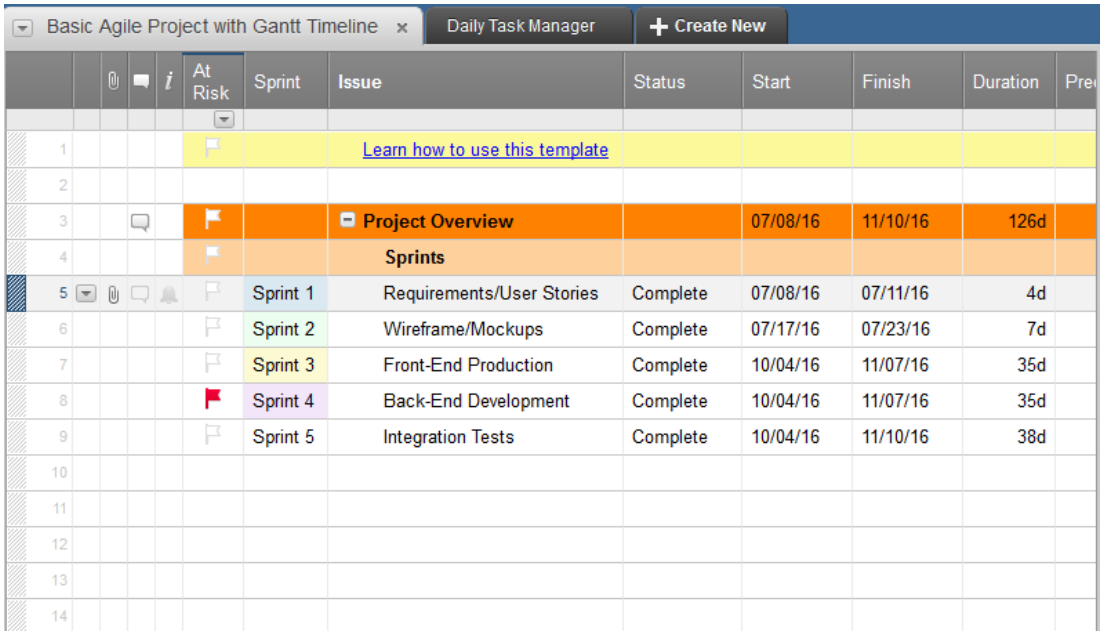
Appendix

Si introducono quelli che sono i tools utilizzati per la pianificazione del lavoro.
Al fine di organizzarsi nel migliore dei modi, prima di iniziare lo sviluppo sono stati redatti due documenti:

1. Project Overview
2. Daily Planning

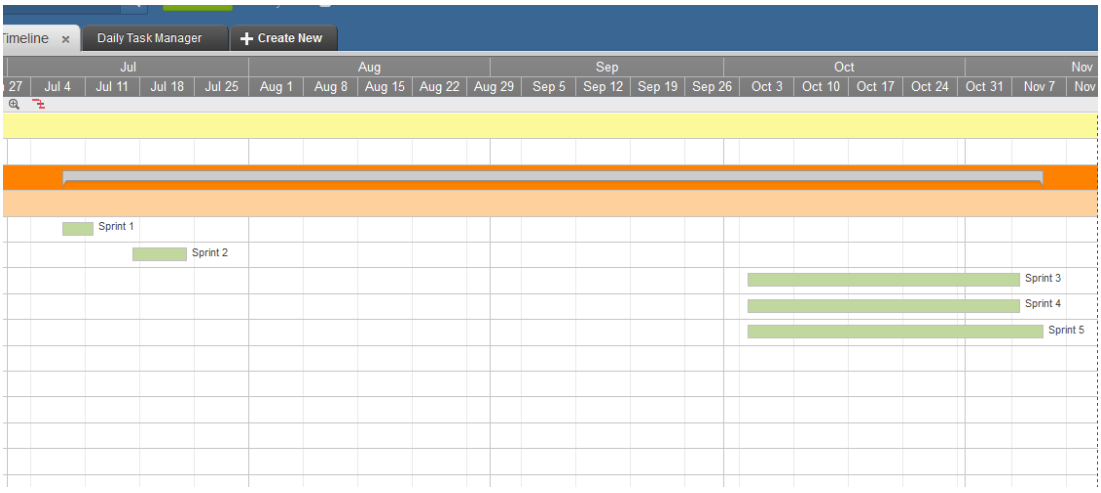
Per la realizzazione di entrambi è stato utilizzato il tool “smartsheet”.

1.1 Appendix: Project Overview



Basic Agile Project with Gantt Timeline × | Daily Task Manager | + Create New

	At Risk	Sprint	Issue	Status	Start	Finish	Duration	Pre
1			Learn how to use this template					
2								
3			**Project Overview**		07/08/16	11/10/16	126d	
4			**Sprints**					
5		Sprint 1	Requirements/User Stories	Complete	07/08/16	07/11/16	4d	
6		Sprint 2	Wireframe/Mockups	Complete	07/17/16	07/23/16	7d	
7		Sprint 3	Front-End Production	Complete	10/04/16	11/07/16	35d	
8		Sprint 4	Back-End Development	Complete	10/04/16	11/07/16	35d	
9		Sprint 5	Integration Tests	Complete	10/04/16	11/10/16	38d	
10								
11								
12								
13								
14								



1.1 Appendix: Daily Planning

Basic Agile Project with Gantt Timeline | Daily Task Manager | + Create New

	Done	Started at	Finished at	Tasks	Status	Difficulty	Comment
1				**High Priority**			
2	☐	10/04/16	10/04/16	⊕ Initialize Application	Complete	🟢	
7	☐	10/06/16	11/09/16	⊖ User Model	Complete	🔴	steadily modified
8	☐			Basic User	Complete		
9	☐			Super User	Complete		
10	☐			Admin	Complete		
11	☐	10/07/16	10/07/16	Sign Up	Complete	🟢	
12	☐	10/09/16	10/11/16	Basic Log In		🟢	
13	☐	10/11/16	10/15/16	⊖ Advanced Log in	Complete	🟡	
14	☐	10/15/16	10/15/16	Manage password	Complete		
15	☐	10/15/16	10/15/16	Logout	Complete	🟢	
16	☐	10/12/16	10/14/16	Account activation	Complete		
17	☐	10/16/16	11/16/16	Create Post	Complete	🟢	
18	☐	10/16/16	10/16/16	Follow a user	Complete	🟢	
19	☐	10/17/16	10/20/16	Facebook Login	Complete	🟡	
20	☐	10/20/16	10/23/16	Add google maps	Complete	🟡	
21	☐	10/23/16	11/29/16	⊖ Add Friend	Complete	🟡	
22	☐	10/23/16	10/23/16	Friendship request	Complete		
23	☐	10/29/16	10/29/16	Delete Friend	Complete		
24	☐	10/29/16	10/30/16	Search friend	Complete	🟢	
25	☐	11/01/16	11/08/16	Add Place	Complete	🔴	
26	☐	11/10/16	11/11/16	Review tests		🟡	
27				⊖ **Medium Priority**			
28	☐	10/24/16	10/29/16	⊖ Specific Feature Add Friend	Complete	🔴	
29	☐	10/24/16	10/29/16	Notification system	Complete		
30	☐	10/16/16	10/16/16	⊖ Specific Feature Post resource	Complete	🟢	
31	☐	10/16/16	10/16/16	Add photo	Complete		
32	☐	10/16/16	10/16/16	Delete	Complete		
33	☐	10/20/16	10/22/16	⊕ Specific Feature Facebook	Complete	🟡	
36				⊖ **Low Priority**			
37	☐	10/29/16	10/30/16	⊖ Specific feature Post resource	Complete	🟡	
38	☐	10/29/16	10/30/16	Add Like	Complete		