|  |  |
| --- | --- |
| UpTAKE  Social Networking Web App | Abstract  This is the engineering proof of concept. The goal is to exercise the technologies we learnt in this course.  Davaabayar Battogtokkh  CS544 Enterprise Architecture |

Contents

[Profject Requirement Check List 2](#_Toc27658361)

[Report Summary or Essay 3](#_Toc27658362)

[Analysis phase: Use case level 1 4](#_Toc27658363)

[Domain object model or Entity Relatioin diagram 5](#_Toc27658364)

[Use case based Task Planning / Tracking diagram 6](#_Toc27658365)

[Friday 13 6](#_Toc27658366)

[Saturday 14. Initial Use case based Task Planning and Tracking diagram. 7](#_Toc27658367)

[Sunday 15 8](#_Toc27658368)

[Monday 16 9](#_Toc27658369)

[Tuesday 17 10](#_Toc27658370)

[Wednesday 18 – Thursday 19 | Final result 11](#_Toc27658371)

[My Tasks 12](#_Toc27658372)

[Screen shots 12](#_Toc27658373)

[12-13 User profile page 12](#_Toc27658374)

[12-13 Pop up post card 13](#_Toc27658375)

[12-13 Following ajax list from profile list 13](#_Toc27658376)

[12-15 Follow from Following pop list 14](#_Toc27658377)

[12-15 Profile page update 14](#_Toc27658378)

[Coding 17](#_Toc27658379)

[Follow functionality 17](#_Toc27658380)

[Login success handler 21](#_Toc27658381)

[GitHub repository insight 25](#_Toc27658382)

# Profject Requirement Check List

|  |  |  |
| --- | --- | --- |
| SN | TASK | PROGRESS |
| 1 | Requirement Analyses and create user case | DONE |
| 2 | Database design | DONE |
| 3 | UI design & navigation | DONE |
| 4 | Login & Registration Functionality | DONE |
| 5 | Add photo & text post | DONE |
| 6 | Add advertisement | DONE |
| 7 | Follow user | DONE |
| 8 | Notifucation | DONE |
| 9 | Like button | DONE |
| 10 | Comment | 70% |
| 11 | Update profile | 50% |
| 12 | Delete user post | DONE |
| 13 | Admin functionality | DONE |
| 14 | Search user | DONE |
| 15 | Search post | DONE |
| 16 | User profile | DONE |
| 17 | User home timeline | DONE |
| 18 | Show advertisement on users nesfeed | 0% |
|  | Validation | DONE |
|  | **Features with extra points** |  |
| 19 | Add video and text post | DONE |
| 20 | Internalization | DONE |

# Report Summary or Essay

As a working as a team, the first challenge was to analyse the project scope and devide to task and assign team members. For that we used Use Case diagram as out planning and communitaion tool. Project period was short. Till mid term we learnt fundamentals of Spring technology and each member worked on their labs to understand and concepts, like: DI, AOP, ORM, JPA, Spring MVC, Thymeleaf, Spring boot, Spring Data, Spring Security, Messaging, JWT.

From the 3rd week we started on working on the project. As we had exact 7 days to work on the project, we needed effective SDLC. Using our knowledge gained from SE course and previous work experience we used our version of Scrum Tool which you call it **“Use case based Task Planning / Tracking”.**

Each day we meet or chat and discuss previous days progress and assign a task, and made required changes to the use case. This method worked very well for us, because it helped us to see the project scope as a whole and monitor the project progress.

**“Use case based Task Planning / Tracking diagrams”** are included, as it was interesting for us to see how it expands.

## Analysis phase: Use case level 1

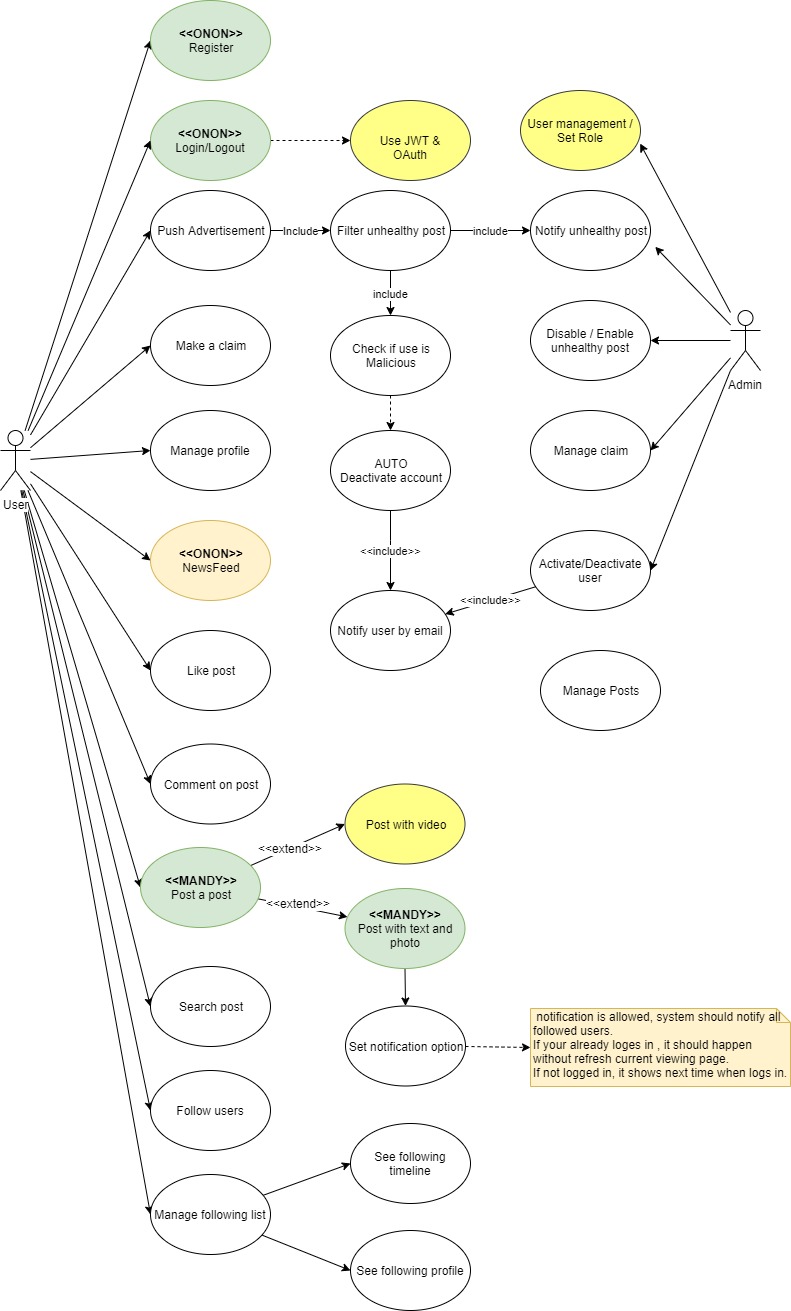
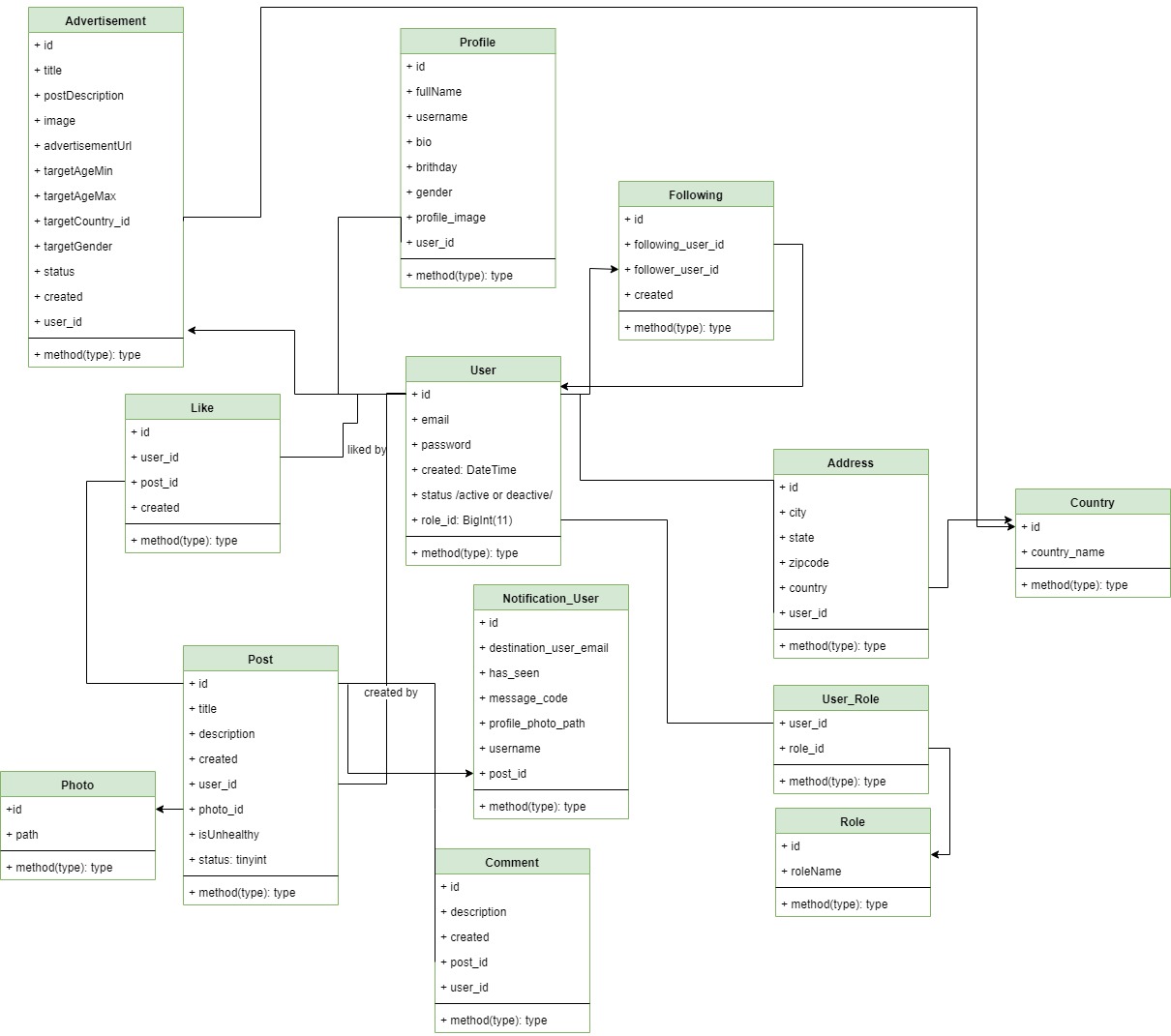
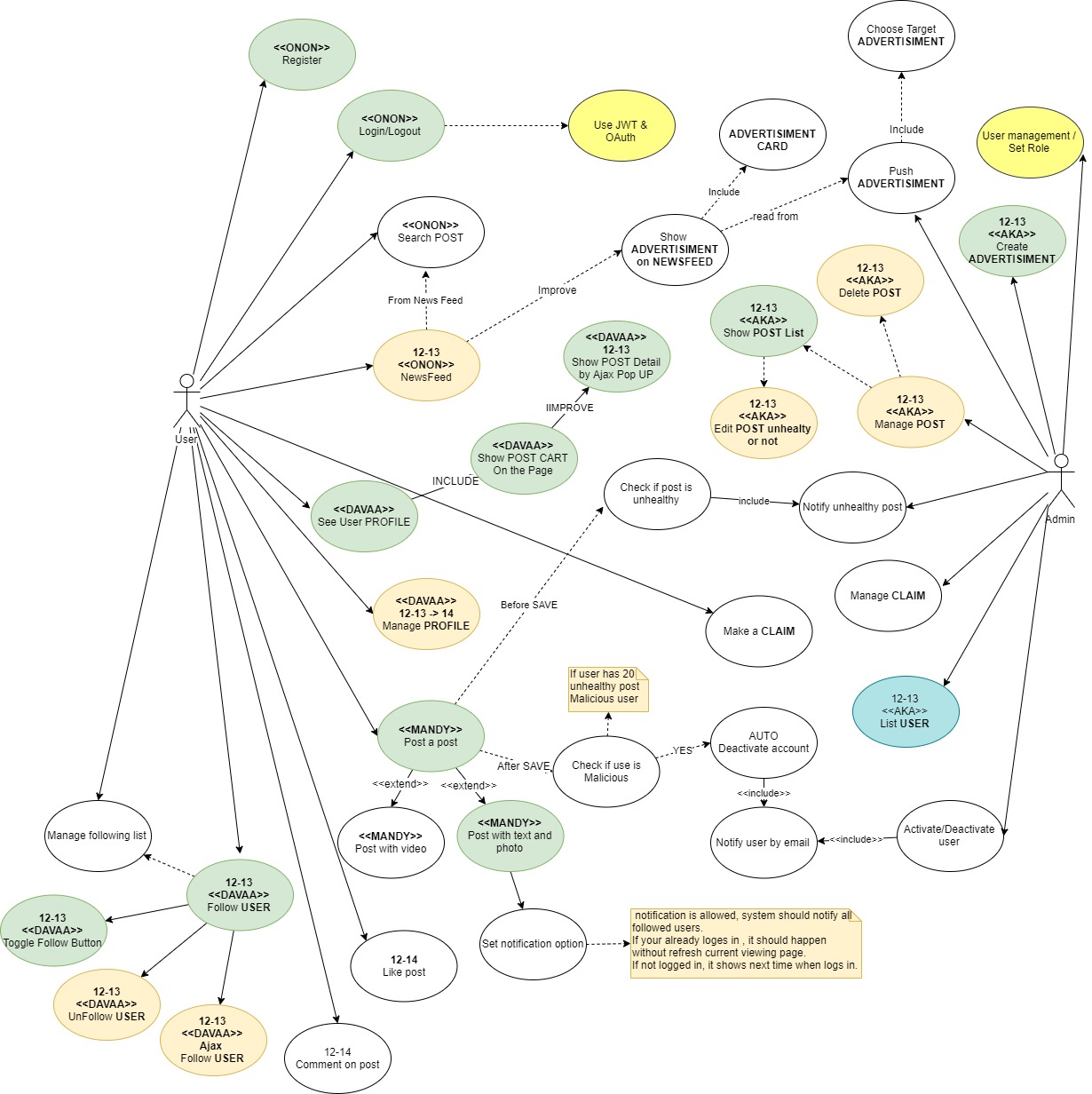


Figure 1Use Case diagram Level 1

## Domain object model or Entity Relatioin diagram

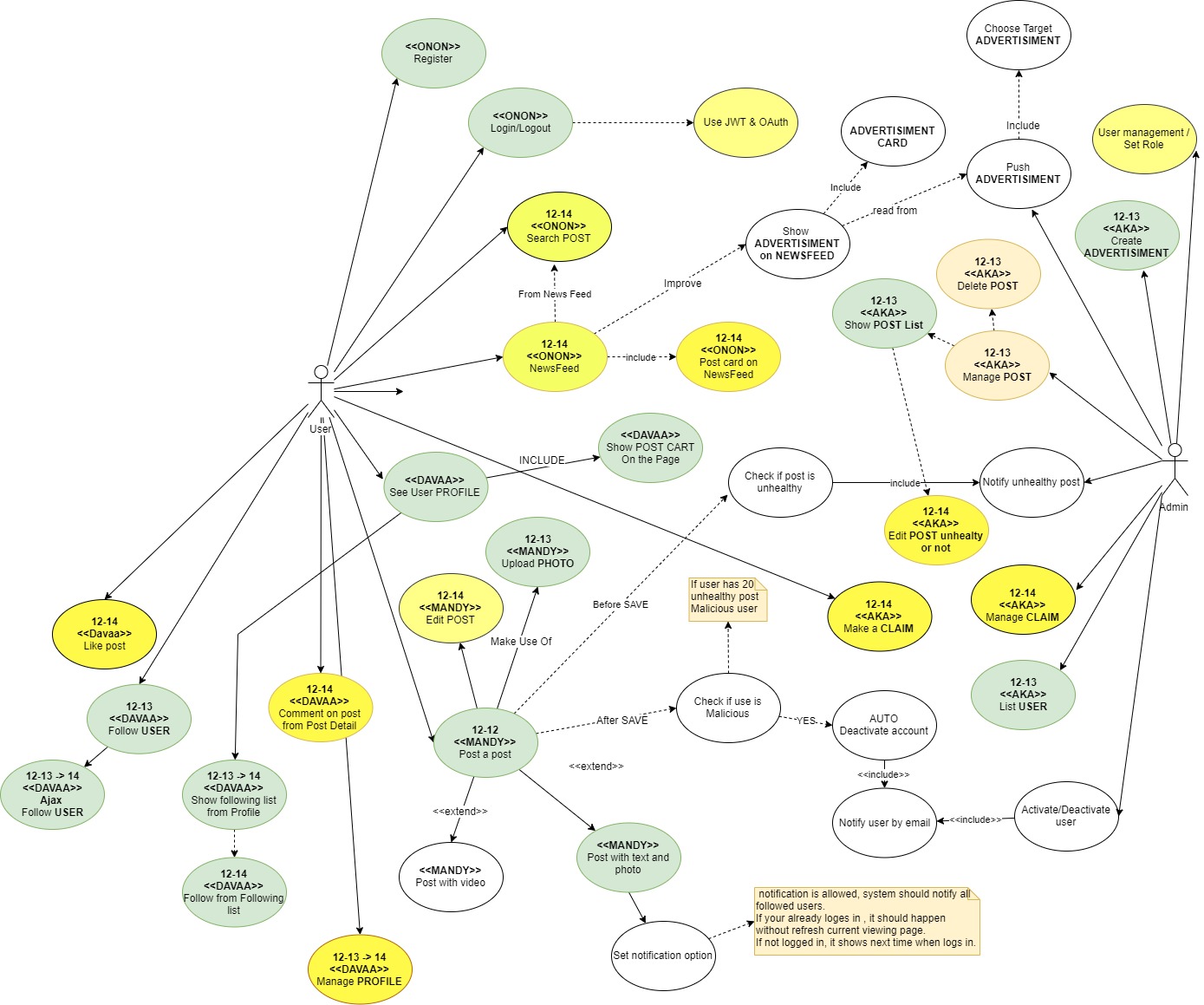


## Use case based Task Planning / Tracking diagram

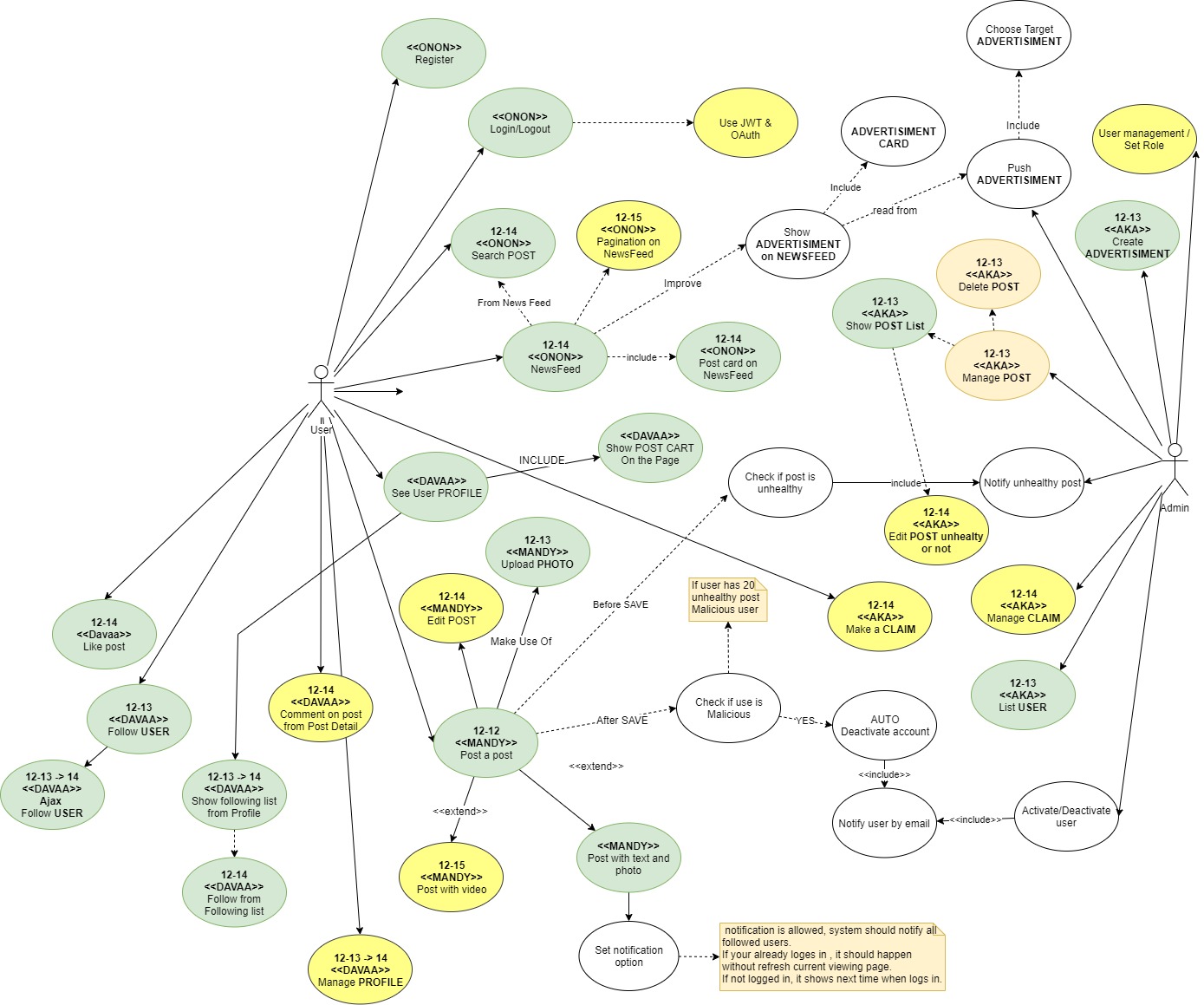
Friday 13 

### Saturday 14. Initial Use case based Task Planning and Tracking diagram.

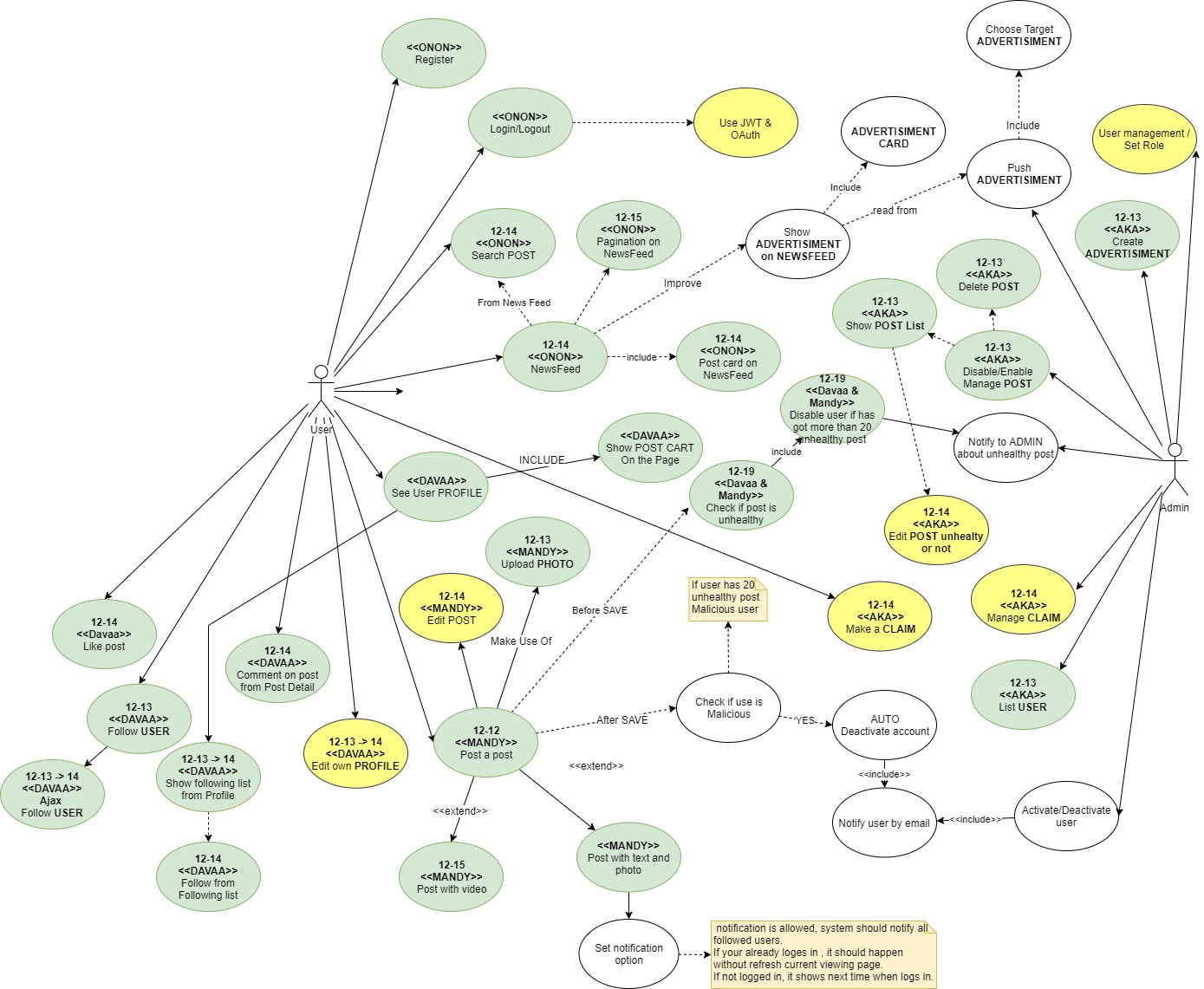
Initial use cases are divided into tasks and planned.



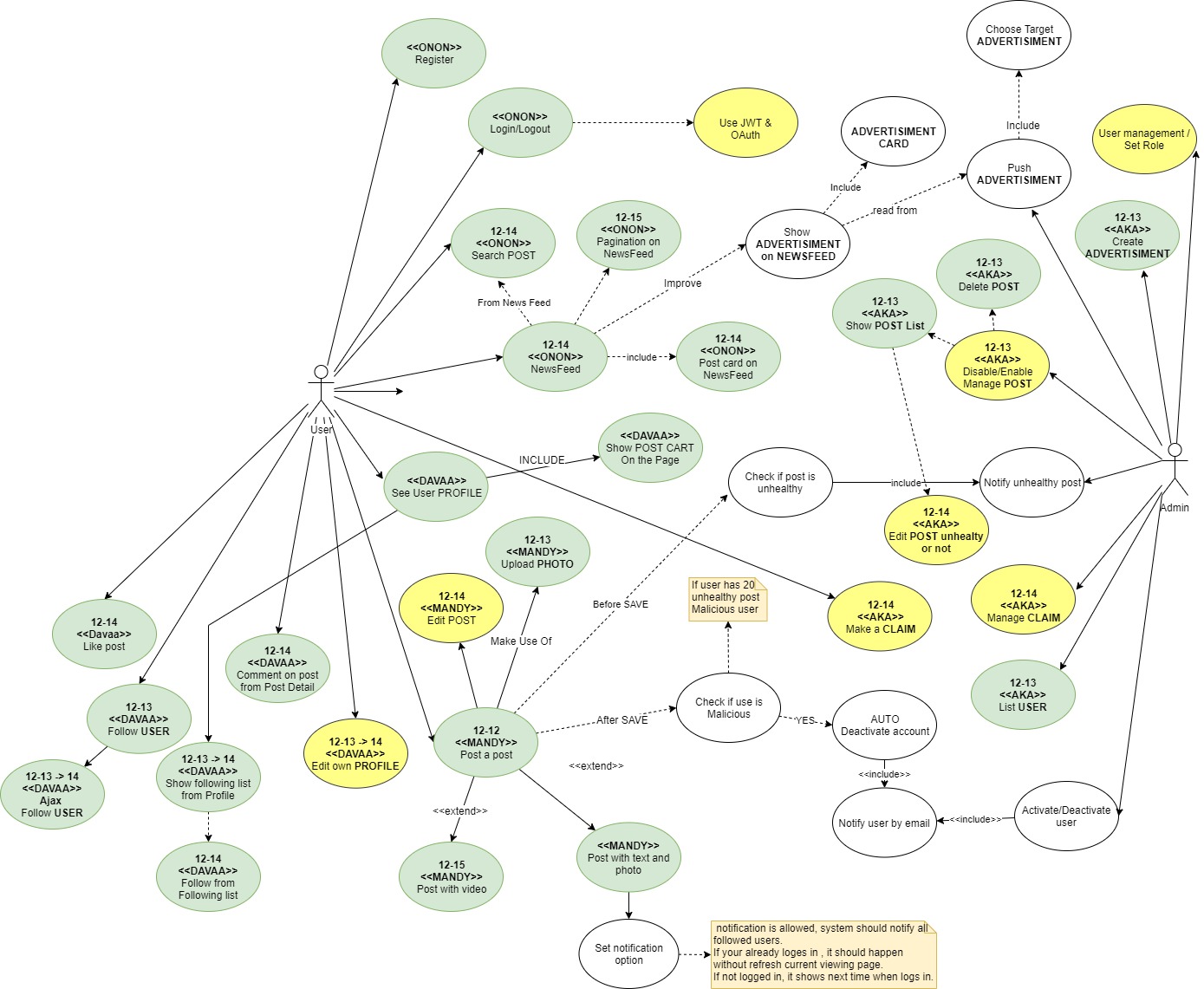
### Sunday 15



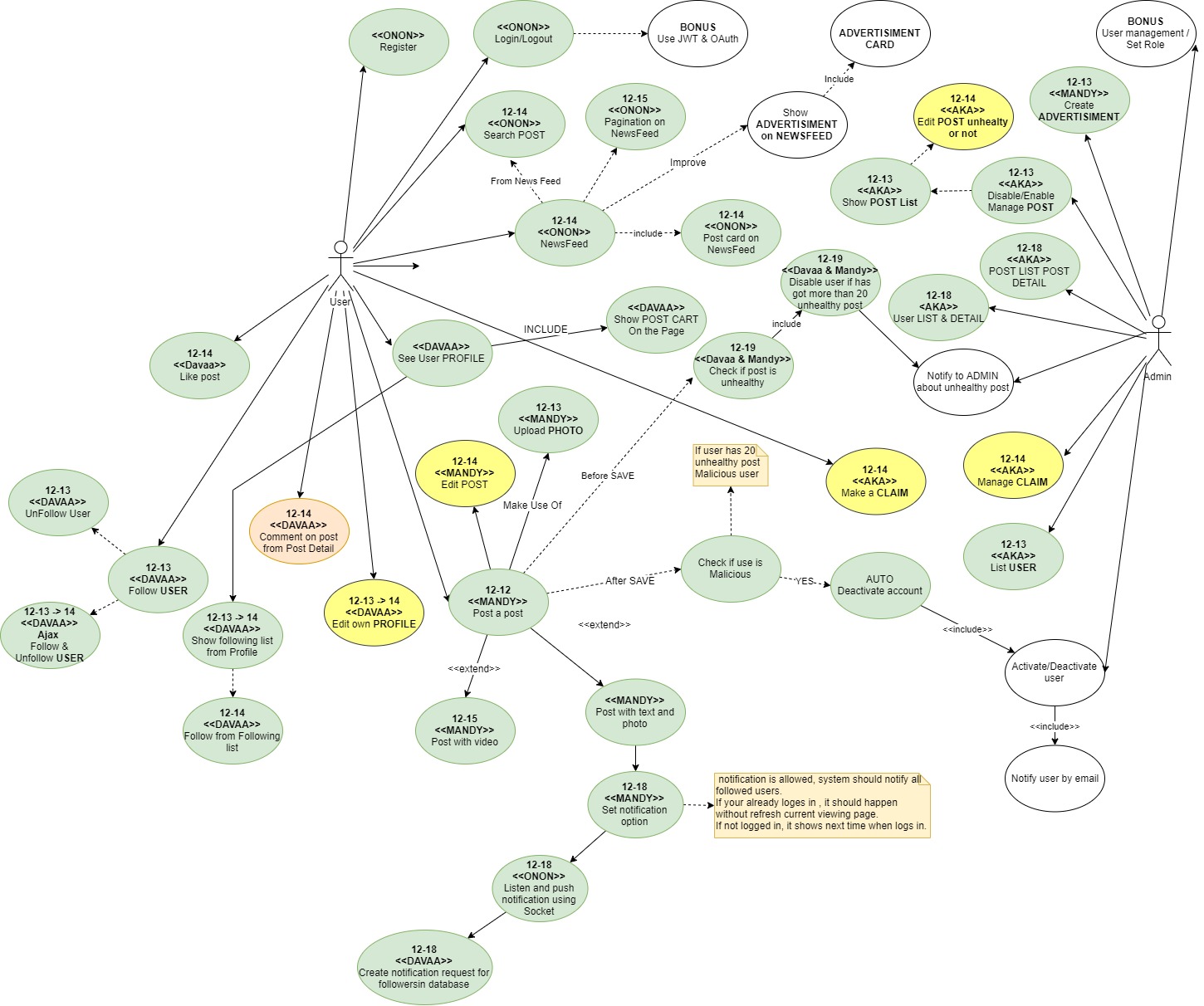
### Monday 16



### Tuesday 17



### Wednesday 18 – Thursday 19 | Final result



# My Tasks

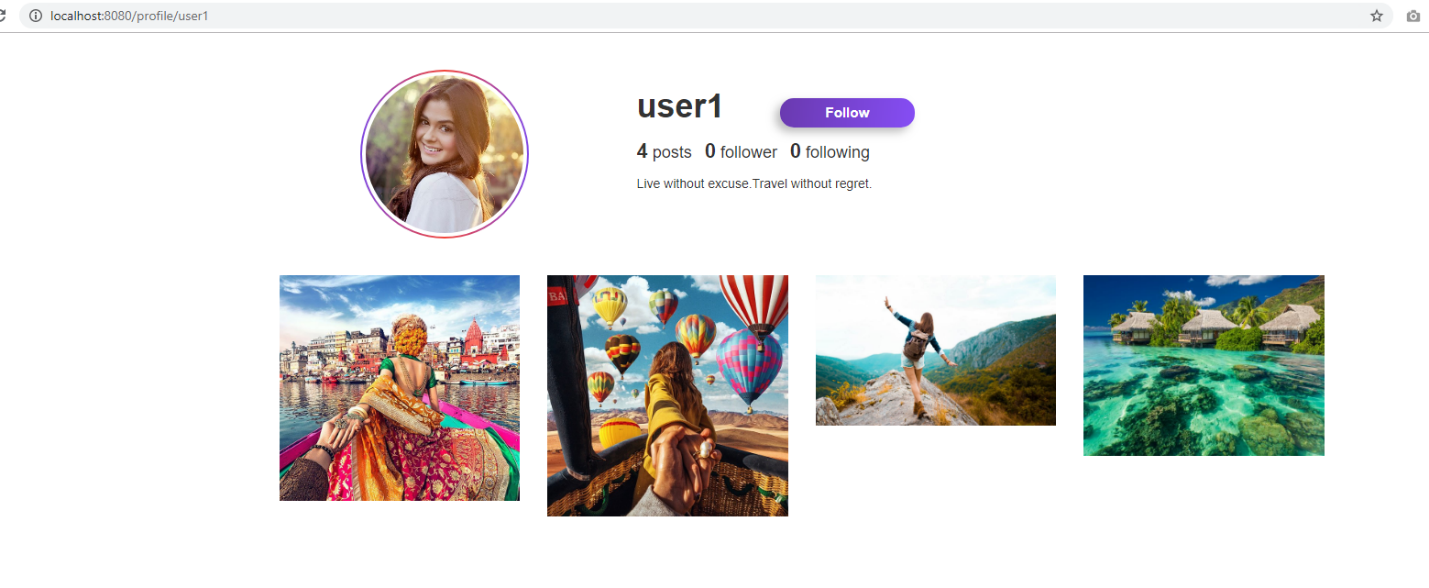
My task completion percentage

|  |  |  |
| --- | --- | --- |
| SN | TASK | PROGRESS |
| 1 | Requirement Analyses and create user case | 100% |
| 2 | Database design | 100% |
| 3 | UI design & navigation | 100% |
| 4 | Manage profile | 30% |
| 5 | Follow and Unfollow user | 100% |
| 6 | Manage following list | 100% |
| 7 | Record Notifucation record when notify to follower option is checked | 100% |
| 8 | Like button | 100% |
| 9 | Comment | 70% |
| 10 | User profile | 100% |
| 11 | Internalization | 100% |

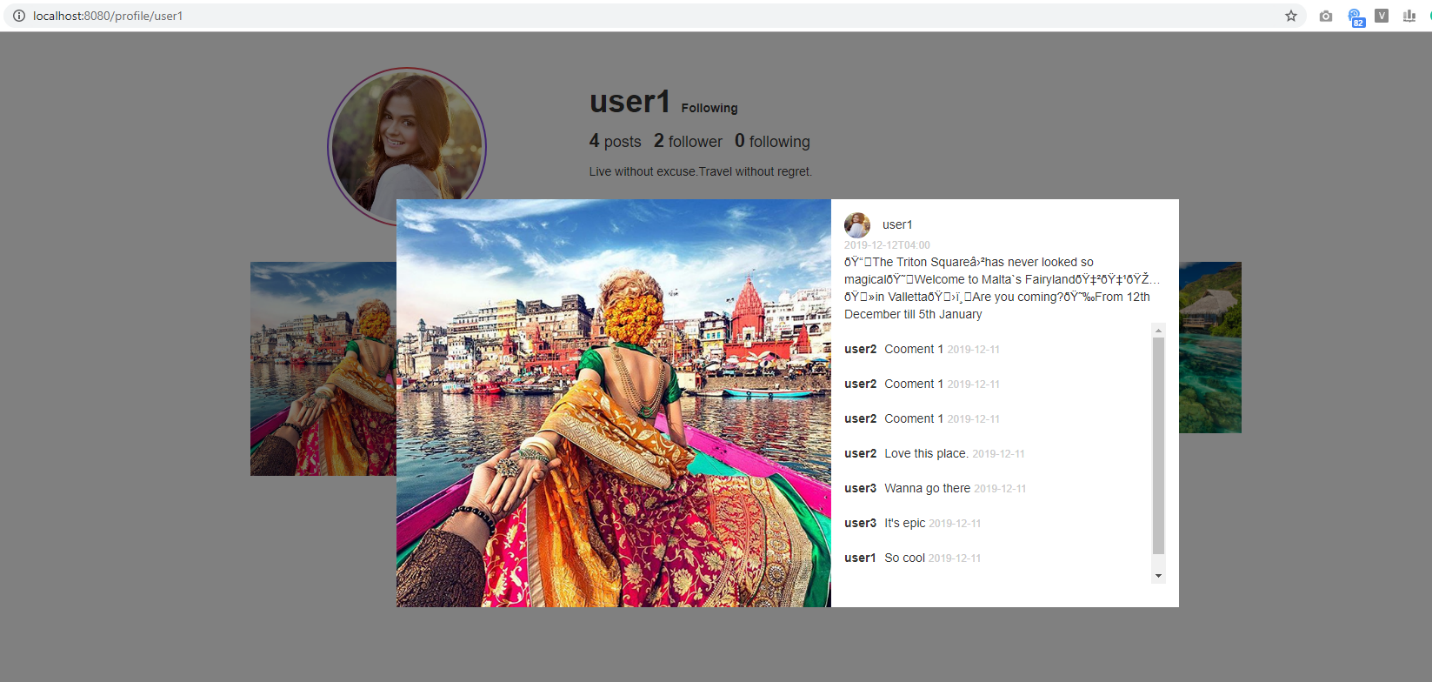
## Screen shots

Some of the screenshots I got during the development.

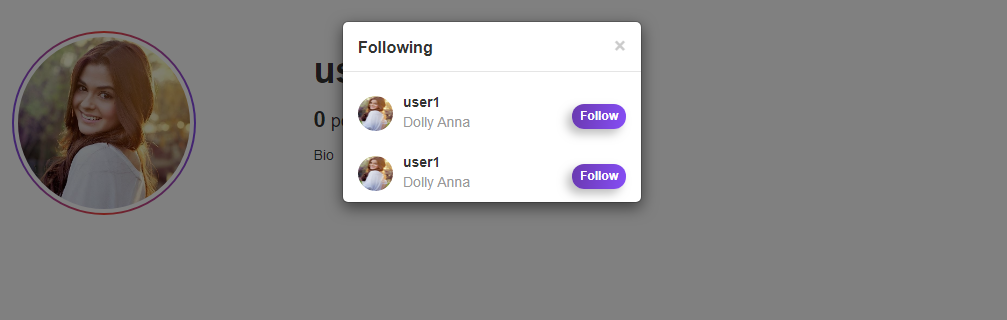
### 12-13 User profile page



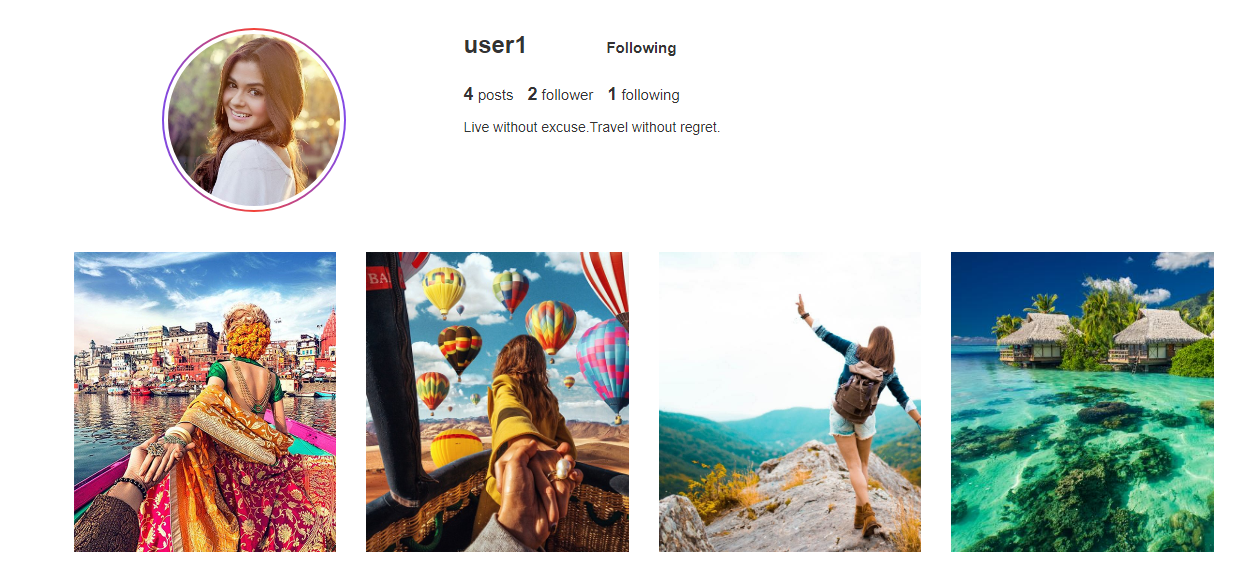
### 12-13 Pop up post card



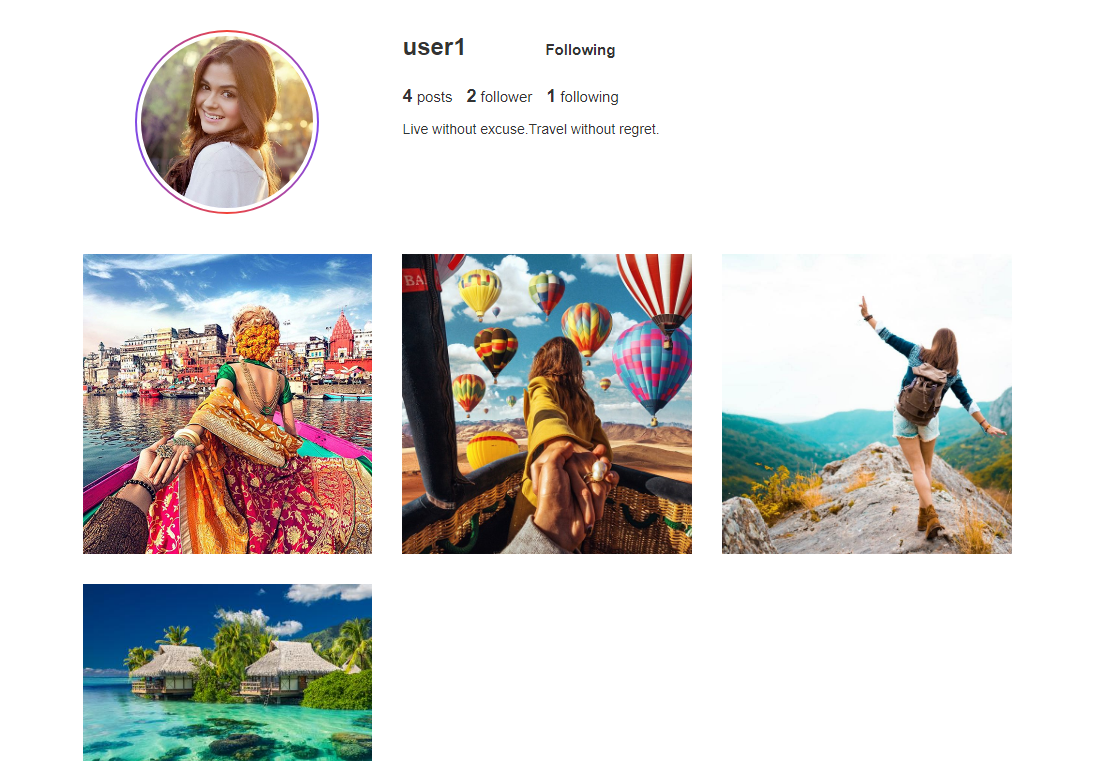
### 12-13 Following ajax list from profile list



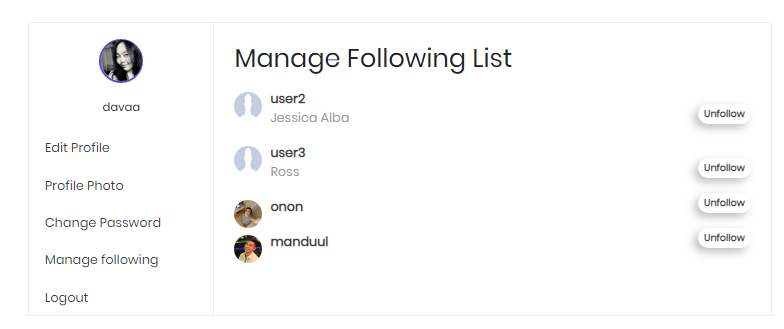
### 12-15 Follow from Following pop list



### 12-15 Profile page update



#### 12-19 Manage following list



#### 12-18 Manage profile



# Coding

### Follow functionality

#### User Entity

@Entity  
@Table(name = **"user"**)  
**public class** User {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 @Column(name = **"id"**)  
 **private** Long **id**;  
  
 @Column(name = **"username"**, unique = **true**)  
 **private** String **username**;  
  
 @Email  
 @NotBlank  
 @Column(name = **"email"**, unique = **true**)  
 **private** String **email**;  
  
 @NotBlank  
 @Size(min = 6)  
 **private** String **password**;  
  
 @Transient  
 **private** String **passwordConfirm**;  
  
 @CreatedDate  
 **private** LocalDate **createdDate**;  
  
 **private boolean status**;  
  
 **private** String **publicName**;  
  
 @OneToOne(cascade = CascadeType.***ALL***)  
 **private** Address **address**;  
  
 @JsonIgnore  
 @OneToMany(mappedBy = **"followedUser"**)  
 @LazyCollection(LazyCollectionOption.***EXTRA***)  
 **private** List<Follower> **followedUsers**;  
  
 @JsonIgnore  
 @OneToMany(mappedBy = **"followingUser"**)  
 @LazyCollection(LazyCollectionOption.***EXTRA***)  
 **private** List<Follower> **followingUser**;  
  
 @ManyToMany  
 @JoinTable(name = **"user\_role"**, joinColumns = {  
 @JoinColumn(name = **"user\_id"**, referencedColumnName = **"id"**)}, inverseJoinColumns = {  
 @JoinColumn(name = **"role\_id"**, referencedColumnName = **"id"**)})  
 @JsonIgnore  
 **private** List<Role> **role**;  
  
 @JsonIgnore  
 @OneToMany(mappedBy = **"user"**)  
 @LazyCollection(value = LazyCollectionOption.***EXTRA***)  
 **private** List<Post> **postList**;  
  
 @OneToOne(mappedBy = **"user"**, cascade = CascadeType.***ALL***, fetch = FetchType.***EAGER***)  
 **private** Profile **profile**;

….

}

#### Follower Entity

@Entity  
@Table(name = **"follower"**)  
**public class** Follower {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.***IDENTITY***)  
 **private long id**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 @JoinColumn(name=**"following\_user\_id"**)  
 **private** User **followingUser**;  
  
 @ManyToOne(fetch = FetchType.***LAZY***)  
 @JoinColumn(name=**"followed\_user\_id"**)  
 **private** User **followedUser**;  
  
 **public** Follower() {  
 }  
  
 **public** Follower(User followingUser, User followedUser) {  
 **this**.**followingUser** = followingUser;  
 **this**.**followedUser** = followedUser;  
 }

#### Follow Repository

@Repository  
**public interface** FollowRepository **extends** JpaRepository<Follower, Long> {  
  
 @Query(value = **"select case when** *count***(f.id)> 0 then true else false end as isfollowing from uptake.follower f where f.followed\_user\_id= :B and f.following\_user\_id = :A"**, nativeQuery = **true**)  
 **int** isAfollowingB(@Param(**"A"**) **long** A, @Param(**"B"**) **long** B);  
  
 Page<Follower> getAllByFollowedUser(User followingUser, PageRequest pageRequest);  
  
 @Query(value = **"delete from uptake.follower where id = :A"**, nativeQuery = **true**)  
 **int** unfollowB(@Param(**"A"**) **long** A);  
}

#### Follow Service

**public interface** FollowerService {  
*// Follower save(User user, User follow);* Follower addFollow(User followingUser, User followedUser);  
 **boolean** addFollowCheck(User followingUser, User followedUser);  
 **int** isAfollowingB(**long** A, **long** B);  
 **void** unfollowB(Long A);  
}

#### Follower Service implementation

@Service  
**public class** FollowerServiceImpl **implements** FollowerService {  
  
 **private** FollowRepository **followRepository**;  
  
 @Autowired  
 **public** FollowerServiceImpl(FollowRepository followRepository) {  
 **this**.**followRepository** = followRepository;  
 }  
  
 @Override  
 **public** Follower addFollow(User followingUser, User followedUser) {  
 **return followRepository**.save(**new** Follower(followingUser,followedUser));  
 }  
  
 @Override  
 **public boolean** addFollowCheck(User followingUser, User followedUser) {  
 System.***out***.println(**"AddFollowCheck"** + followingUser.getId() + **", "** + followedUser.getId());  
 **if**(followedUser.getId()!=followingUser.getId()){  
 **if**(isAfollowingB(followingUser.getId(),followedUser.getId())==0) {  
 **followRepository**.save(**new** Follower(followingUser,followedUser));  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 @Override  
 **public int** isAfollowingB(**long** A, **long** B) {  
 **return followRepository**.isAfollowingB(A,B);  
 }  
  
 @Override  
 **public void** unfollowB(Long A) {  
 **followRepository**.deleteById(A);  
 }  
  
*// @Override  
// public Follower save(User user, User follow) {  
// return followRepository.save(new Follower(user, follow));  
// }*}

#### Profile page’s part:

<**div class="col-md-8"**>  
 <**div class="relative"**>  
 <**strong class="title" th:text="${user.username}"**></**strong**>  
 <**a th:if="${isMyAccount}" th:text="#{message.profile.editprofile}" href="/account/edit" class="btn btn-outline-secondary"**></**a**>  
 <**span th:unless="${isMyAccount}"**>  
 <**span th:if="${isFollowing}" class="textFollowing" th:text="#{message.profile.following}"**></**span**>  
 <**a th:unless="${isFollowing}" class="follow uptake-btn enabled" th:href="'/profile/follow/' + ${user.id}" th:data-id="${user.id}" id="anchorFollow" th:text="#{message.profile.follow}"**></**a**>  
 <**span id="textFollowing" class="textFollowing"**></**span**>  
 </**span**>  
 </**div**>  
 <**div class="attr\_wrap mb10 mt20"**>  
 <**div class="attr"**><**span th:text="${user.getPostList().size()}"**></**span**> posts</**div**>  
 <**div class="attr"**>  
 <**div th:onclick="'openFollowerModal(\'' + ${user.id} + '\');'"**>  
 <**span th:text="${user.getFollowedUsers().size()}" id="numOfFollower"**></**span**> follower  
 </**div**>  
 </**div**>  
 <**div class="attr"**>  
 <**div th:onclick="'openFollowingModal(\'' + ${user.id} + '\');'"**>  
 <**span th:text="${user.getFollowingUser().size()}"**></**span**> following  
 </**div**>  
 </**div**>  
 </**div**>

Ajax call from profile

**$**(**".follow"**).click(**function** (e) {  
 e.preventDefault();  
 **var** id = **$**(**this**).data(**"id"**);  
 ***console***.log(**"Follow called by ajax called"** + id);  
 **$**.ajax({  
 **url**: **"/profile/followbyajax/"** + id,  
 success: **function** (data) {  
 **var** obj = ***JSON***.parse(data);  
 **if**(obj.**result**){  
 **$**(**"#numOfFollower"**).text(obj.**n**);  
 **$**(**"#anchorFollow"**).hide(300);  
 **$**(**"#textFollowing"**).text(**"Following"**);  
 }**else**{  
 ***console***.log(**"Already following"**);  
 }  
 }  
 });  
});

#### Follow and Unfollow method in FollowController

@RequestMapping(value=**"profile/followbyajax/{userId}"**, method = RequestMethod.***GET***)  
@ResponseBody  
**public** String ajaxFollow(@PathVariable **long** userId, Model model) {  
 **long** currentUserId = 1;  
 String email = **null**;  
 User loggedUser = **null**;  
 Object principal = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 **if** (principal **instanceof** UserDetails){  
 email = ((UserDetails) principal).getUsername();  
 loggedUser = **userService**.getUserByEmail(email);  
 }  
  
 **if**(loggedUser != **null** ){  
 currentUserId = loggedUser.getId();  
 }  
 User followingUser = **userService**.getUserById(currentUserId);  
 User followedUser = **userService**.getUserById(userId);  
  
 JSONObject response = **new** JSONObject();  
 **if**(**followerService**.addFollowCheck(followingUser,followedUser)) {  
 response.put(**"result"**,**true**);  
 response.put(**"followingUser"**,currentUserId);  
 response.put(**"followedUser"**,userId);  
 response.put(**"n"**, followedUser.getFollowedUsers().size());  
  
 }**else**{  
 response.put(**"result"**,**false**);  
 }  
 **return** response.toString();  
}  
  
@RequestMapping(value=**"profile/unfollowbyajax/{id}"**, method = RequestMethod.***GET***)  
@ResponseBody  
**public** String ajaxUnFollow(@PathVariable **long** id, Model model) {  
 JSONObject response = **new** JSONObject();  
 Long lId = Long.*valueOf*(id);  
 **followerService**.unfollowB(lId);  
 response.put(**"result"**,**true**);  
 response.put(**"n"**, lId);  
 **return** response.toString();  
}

### Login success handler

This handler redirect CLIENT to Home page and ADMIN to admin dashboard after sucessful login.

#### 1. Inject AuthenticationSuccessHandler bean in Security config.

@EnableWebSecurity  
**public class** SecurityConfig **extends** WebSecurityConfigurerAdapter {

@Bean  
**public** AuthenticationSuccessHandler myAuthenticationSuccessHandler(){  
 **return new** MySimpleUrlAuthenticationSuccessHandler();  
}

}

#### 2.Register successHandler in formLogin()

@Override  
**protected void** configure(HttpSecurity http) **throws** Exception {  
  
 http.authorizeRequests()  
 .antMatchers(**"/"**,**"/home"**).authenticated()  
 .antMatchers(**"/postPhoto"**).authenticated()  
 .antMatchers(**"/postVideo"**).authenticated()  
 .antMatchers(**"/admin"**,**"/admin/\*\*"**).hasRole(**"ADMIN"**)  
 .antMatchers(**"/profile"**,**"/profile/\*\*"**).authenticated()  
 .antMatchers(**"/login"**).permitAll()  
 .antMatchers(**"/logout"**).permitAll()  
 .and()  
 .formLogin()  
 .loginPage(**"/login"**)  
 .successHandler(myAuthenticationSuccessHandler())  
 .failureUrl(**"/login?error=true"**)  
 .usernameParameter(**"email"**)  
 .passwordParameter(**"password"**)  
 .and()  
 .logout()  
 .logoutRequestMatcher(**new** AntPathRequestMatcher(**"/logout"**))  
 .clearAuthentication(**true**)  
 .invalidateHttpSession(**true**)  
 .logoutSuccessUrl(**"/login"**)  
 .and()  
 .exceptionHandling().accessDeniedPage(**"/403Forbidden"**)  
 .and()  
 .csrf().disable();  
  
}

#### 3. MySimpleUrlAuthenticationSuccessHandler class implementing AuthenticationSuccessHandler

**public class** MySimpleUrlAuthenticationSuccessHandler **implements** AuthenticationSuccessHandler {  
 **protected** Log **logger** = LogFactory.*getLog*(**this**.getClass());  
 **private** RedirectStrategy **redirectStrategy** = **new** DefaultRedirectStrategy();  
  
 @Override  
 **public void** onAuthenticationSuccess(HttpServletRequest request,  
 HttpServletResponse response, Authentication authentication)  
 **throws** IOException {  
  
 handle(request, response, authentication);  
 clearAuthenticationAttributes(request);  
 }  
 **protected void** handle(HttpServletRequest request,  
 HttpServletResponse response, Authentication authentication)  
 **throws** IOException {  
  
 String targetUrl = determineTargetUrl(authentication);  
  
 **if** (response.isCommitted()) {  
 **logger**.debug(  
 **"Response has already been committed. Unable to redirect to "** + targetUrl);  
 **return**;  
 }  
  
 **redirectStrategy**.sendRedirect(request, response, targetUrl);  
 }  
 **protected** String determineTargetUrl(Authentication authentication) {  
 **boolean** isUser = **false**;  
 **boolean** isAdmin = **false**;  
 Collection<? **extends** GrantedAuthority> authorities  
 = authentication.getAuthorities();  
 **for** (GrantedAuthority grantedAuthority : authorities) {  
 **if** (grantedAuthority.getAuthority().equals(**"ROLE\_USER"**)) {  
 isUser = **true**;  
 **break**;  
 } **else if** (grantedAuthority.getAuthority().equals(**"ROLE\_ADMIN"**)) {  
 isAdmin = **true**;  
 **break**;  
 }  
 }  
  
 **if** (isUser) {  
 **return "/"**;  
 } **else if** (isAdmin) {  
 **return "/admin"**;  
 } **else** {  
 **throw new** IllegalStateException();  
 }  
 }  
  
 **protected void** clearAuthenticationAttributes(HttpServletRequest request) {  
 HttpSession session = request.getSession(**false**);  
 **if** (session == **null**) {  
 **return**;  
 }  
 session.removeAttribute(WebAttributes.***AUTHENTICATION\_EXCEPTION***);  
 }  
  
 **public void** setRedirectStrategy(RedirectStrategy redirectStrategy) {  
 **this**.**redirectStrategy** = redirectStrategy;  
 }  
 **protected** RedirectStrategy getRedirectStrategy() {  
 **return redirectStrategy**;  
 }  
}

# GitHub repository insight

