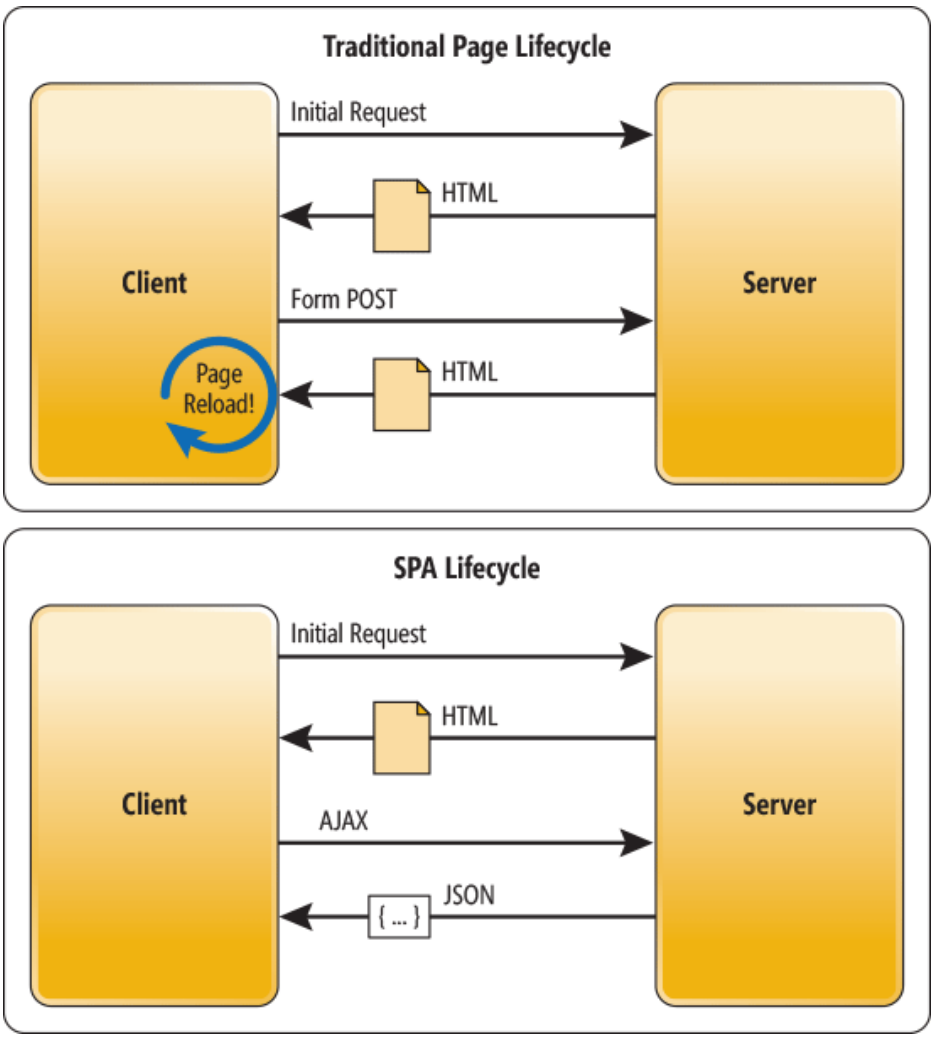
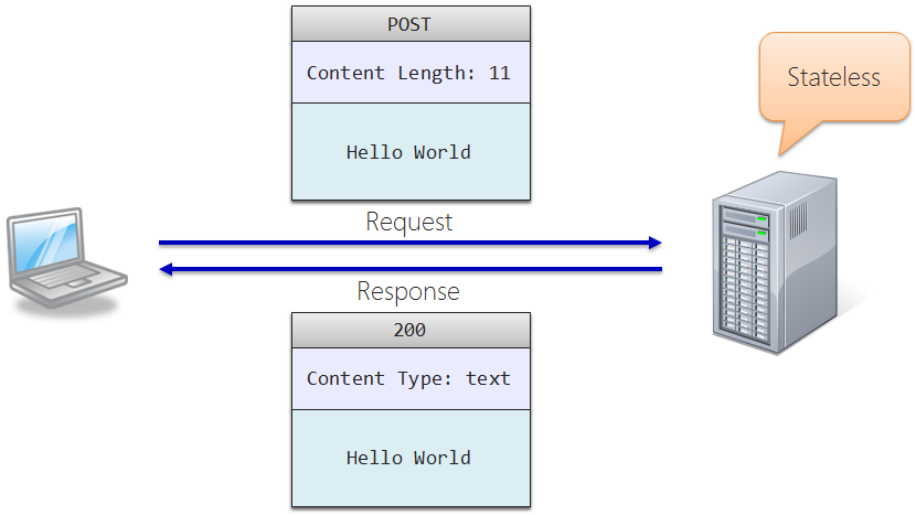
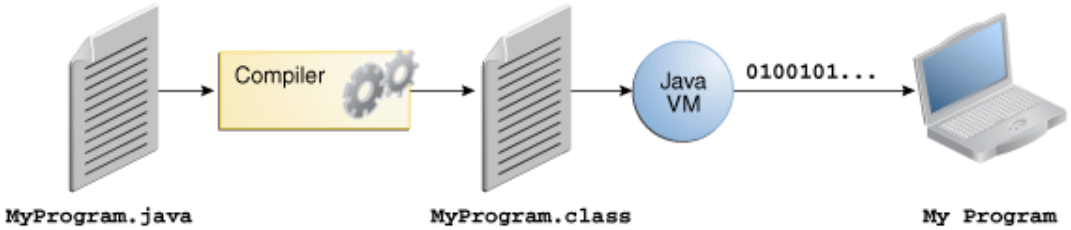
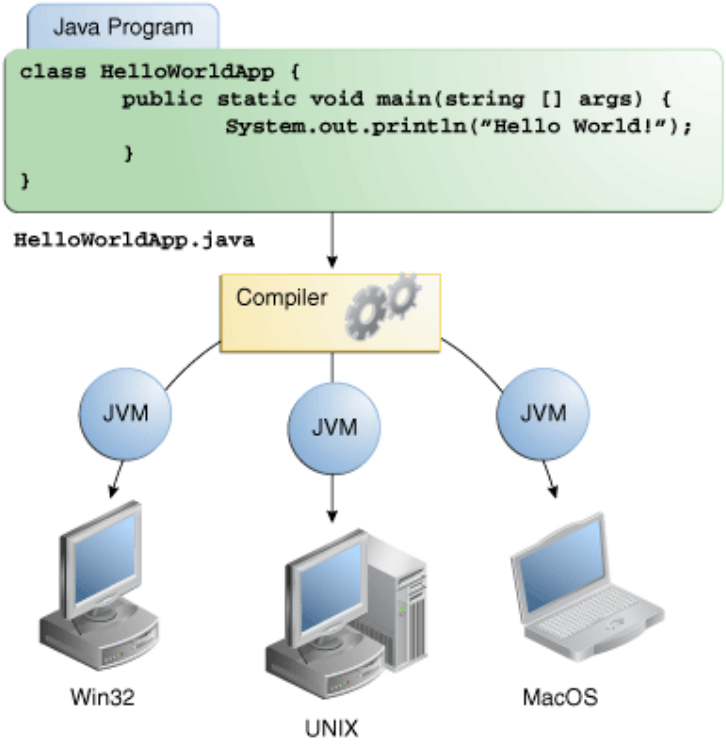
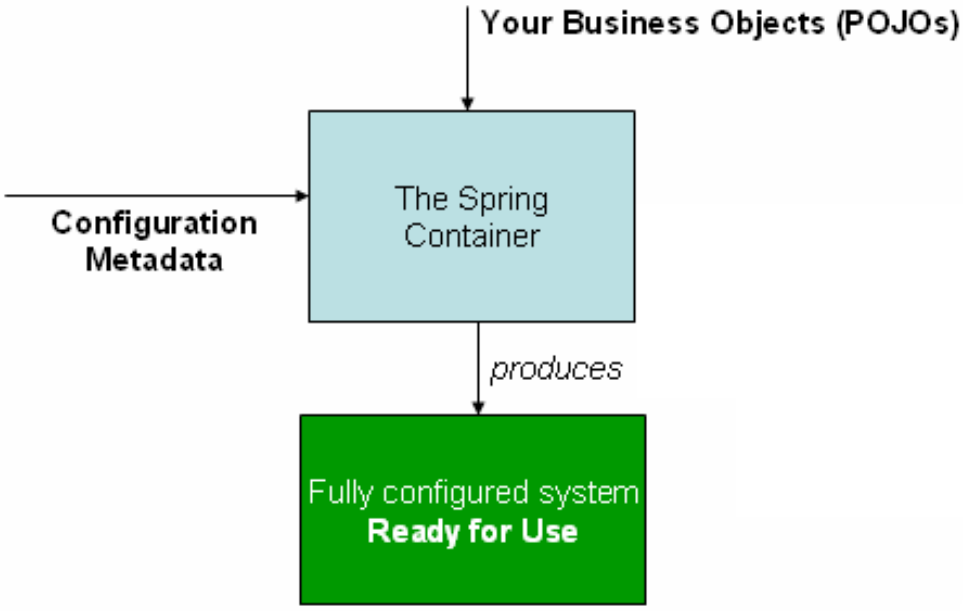

Database
Web API
Single Page Application

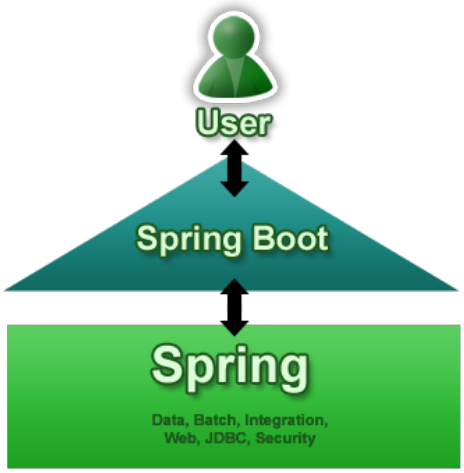

Traditional Page Lifecycle
Client
Server
Initial Request
HTML
Form POST
HTML
Page Reload!
SPA Lifecycle
Client
Server
Initial Request
HTML
AJAX
JSON
{ ... }

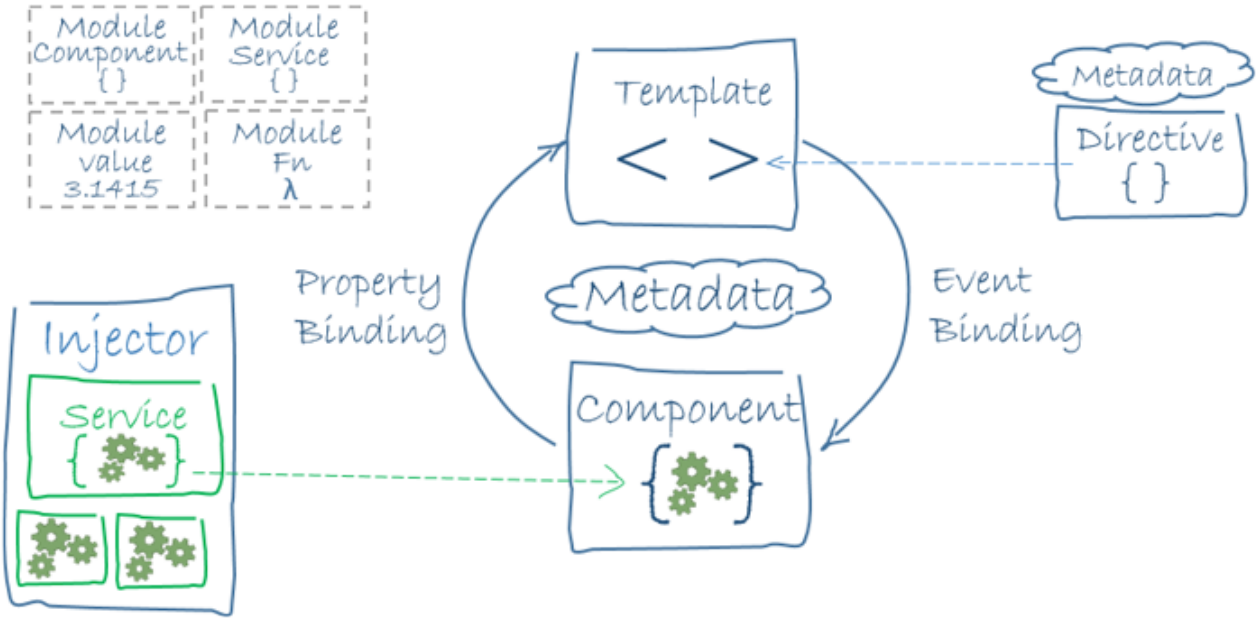

POST
Content Length: 11
Hello World
Stateless
Request
Response
200
Content Type: text
Hello World

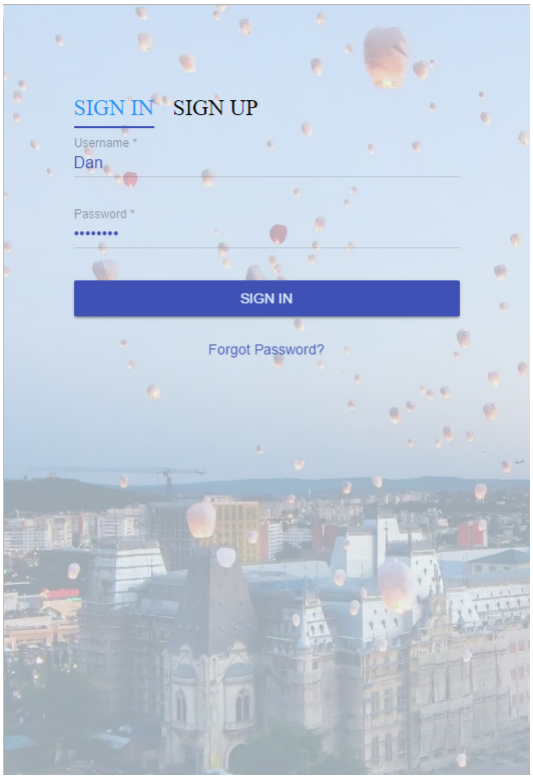

Compiler
Java VM
0100101...
MyProgram.java
MyProgram.class
My Program

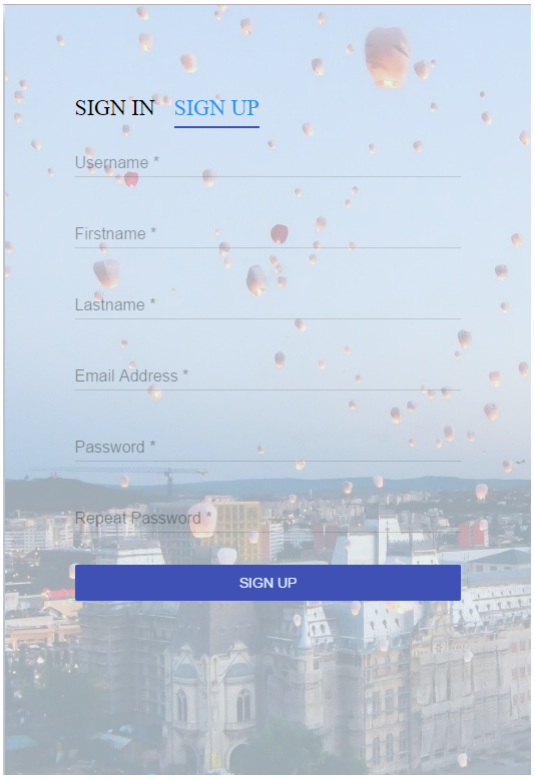

Java Program
class HelloWorldApp {
public static void main(string [] args) {
System.out.println("Hello World!");
}
}
HelloWorldApp.java
Compiler
JVM
JVM
JVM
Win32
UNIX
MacOS

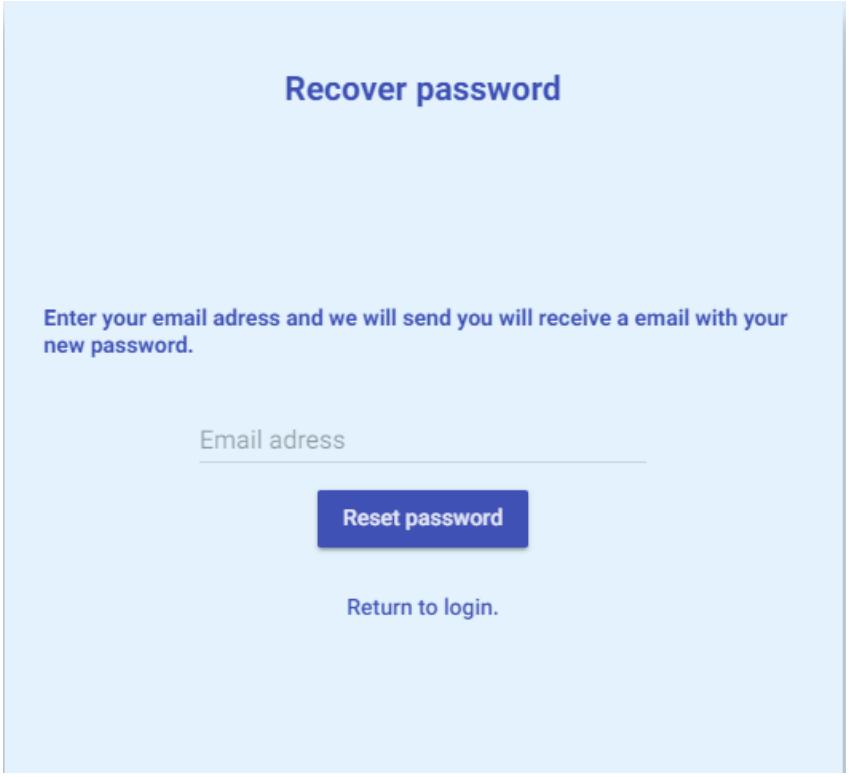

Your Business Objects (POJOs)
Configuration Metadata
The Spring Container
produces
Fully configured system
Ready for Use

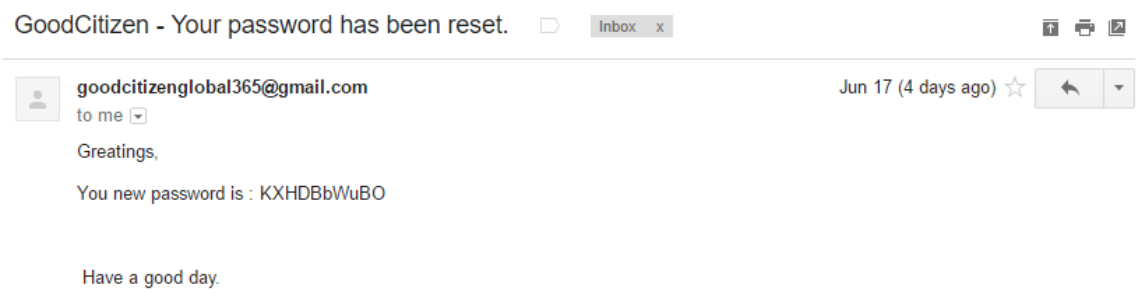

User
Spring Boot
Spring
Data, Batch, Integration,
Web, JDBC, Security

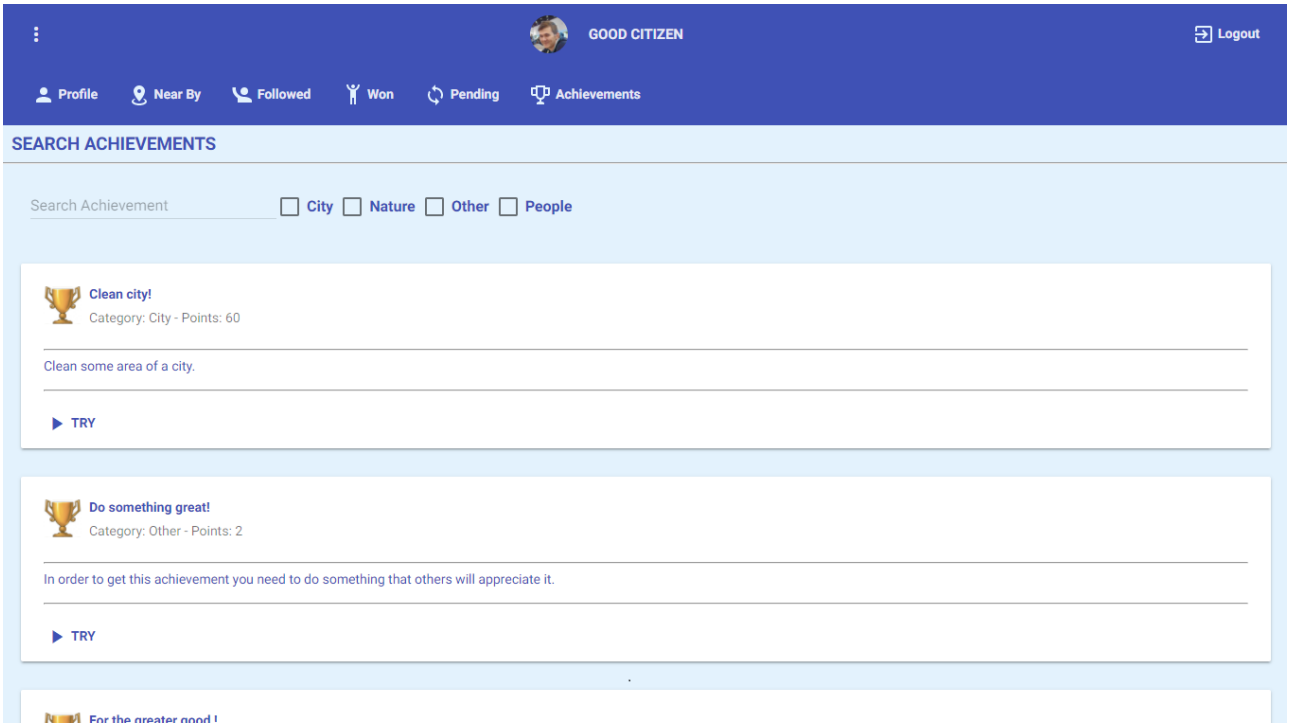

Module
Component
{ }
Module
Service
{ }
Module
value
3.1415
Module
Fn
λ
Template
< >
Metadata
Directive
{ }
Property
Binding
Metadata
Event
Binding
Injector
Service
Component

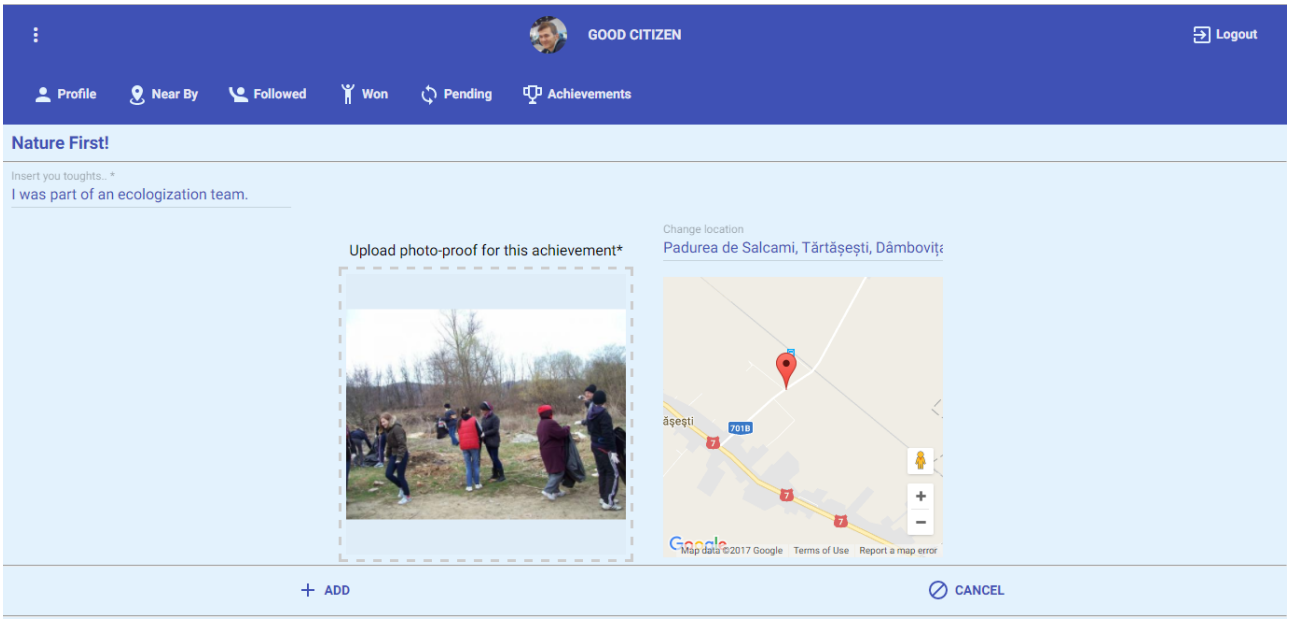

SIGN IN
SIGN UP
Username *
Dan
Password *
SIGN IN
Forgot Password?

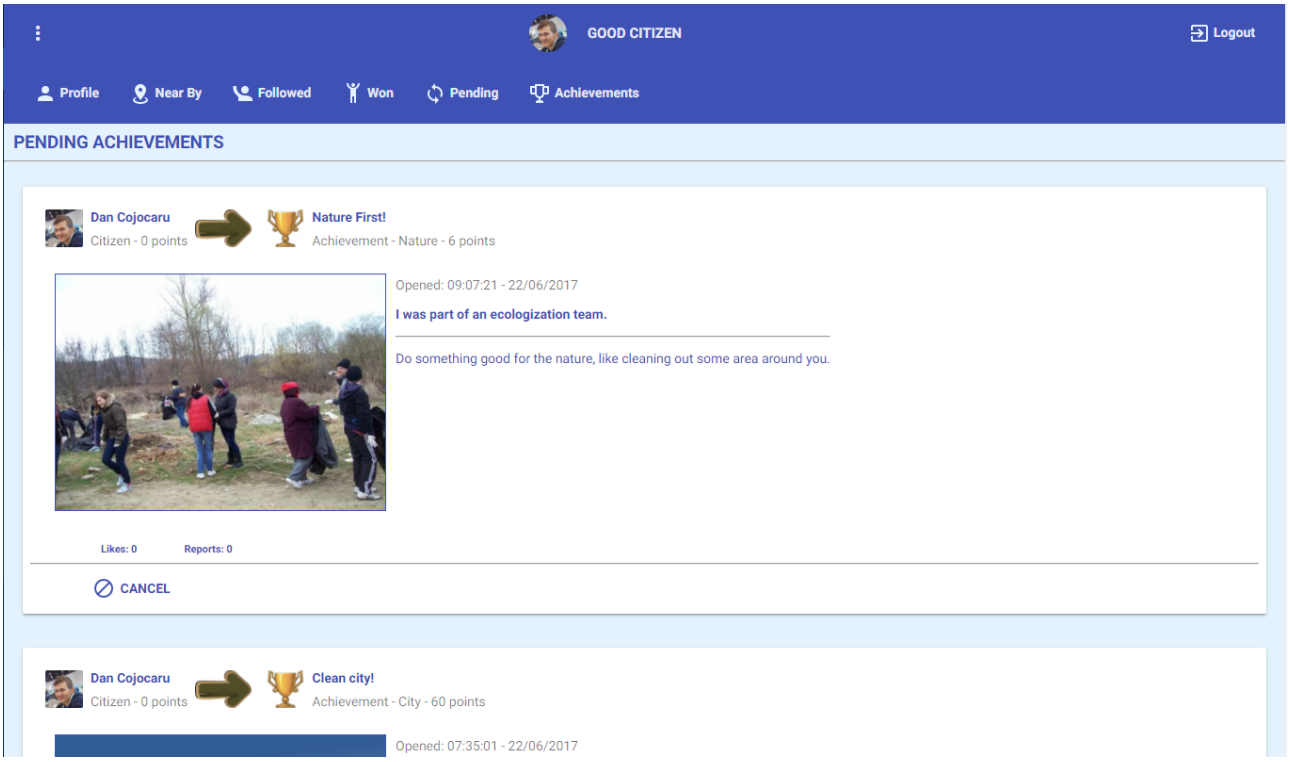

SIGN IN
SIGN UP
Username *
Firstname *
Lastname *
Email Address *
Password *
Repeat Password *
SIGN UP

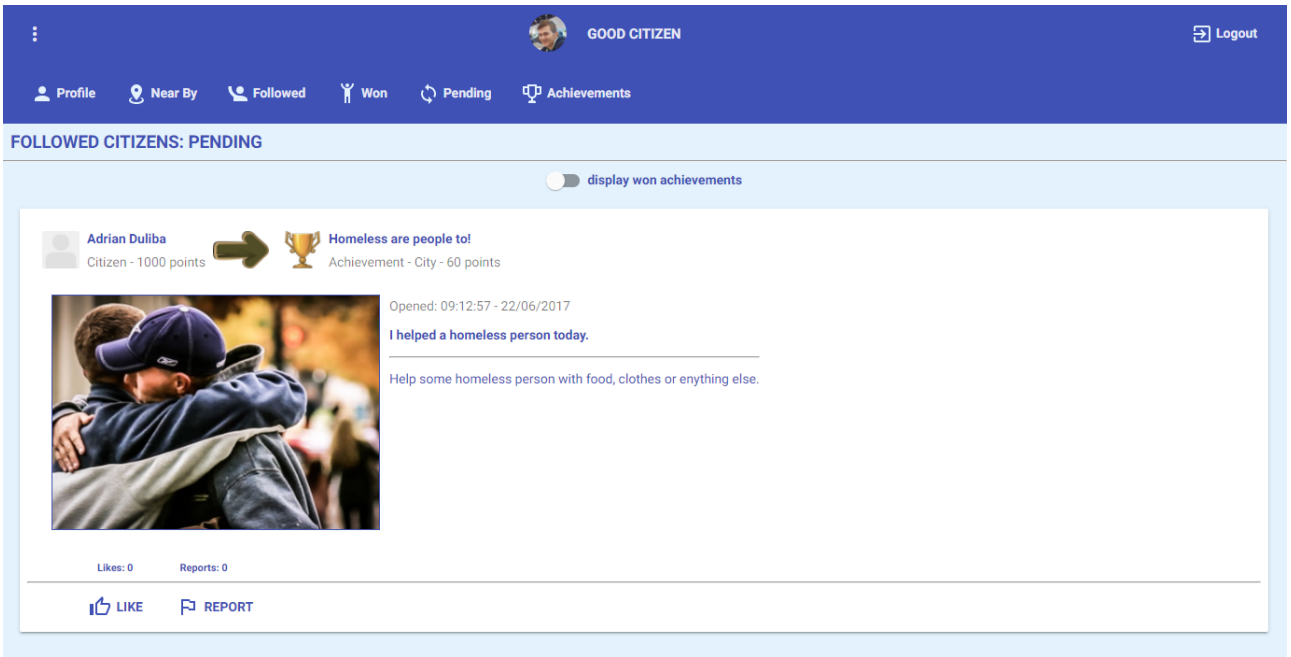

Recover password
Enter your email adress and we will send you will receive a email with your new password.
Email adress
Reset password
Return to login.

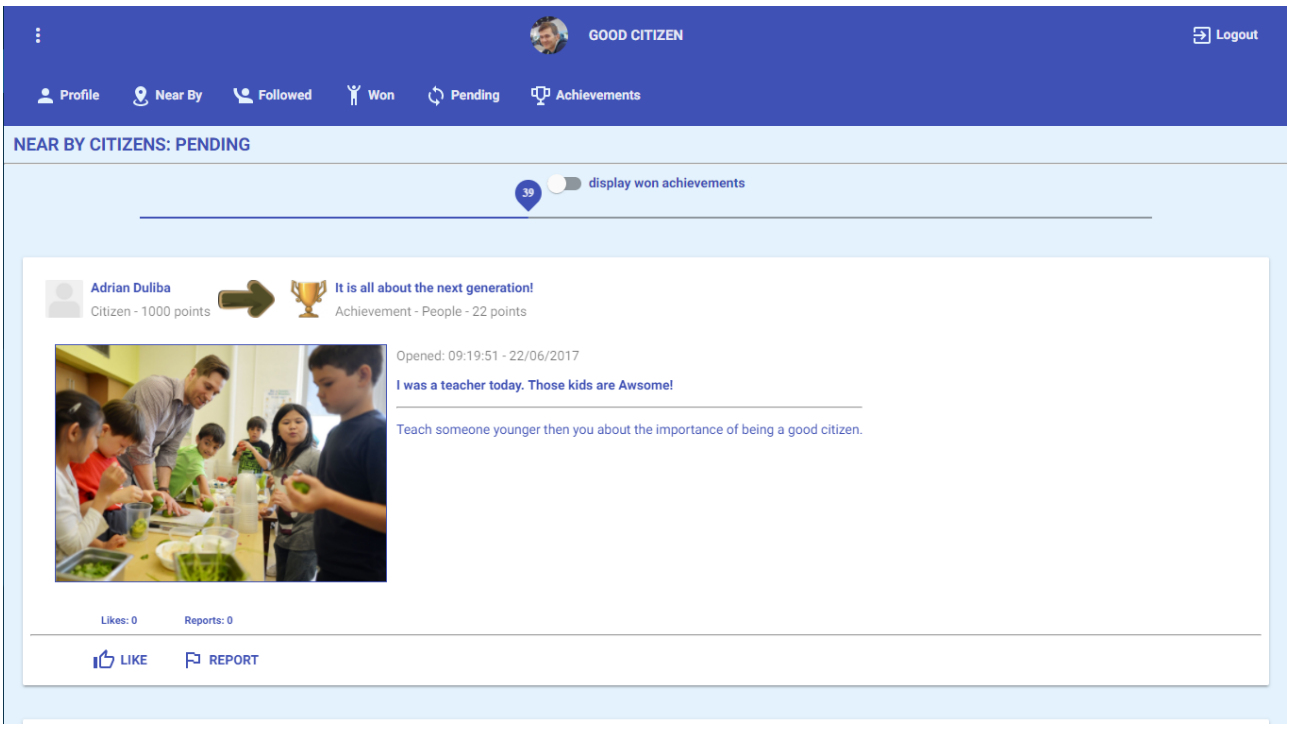

GoodCitizen - Your password has been reset.
Inbox x
goodcitizenglobal365@gmail.com
to me
Jun 17 (4 days ago)
Greatings,
You new password is : KXHDBbWuBO
Have a good day.

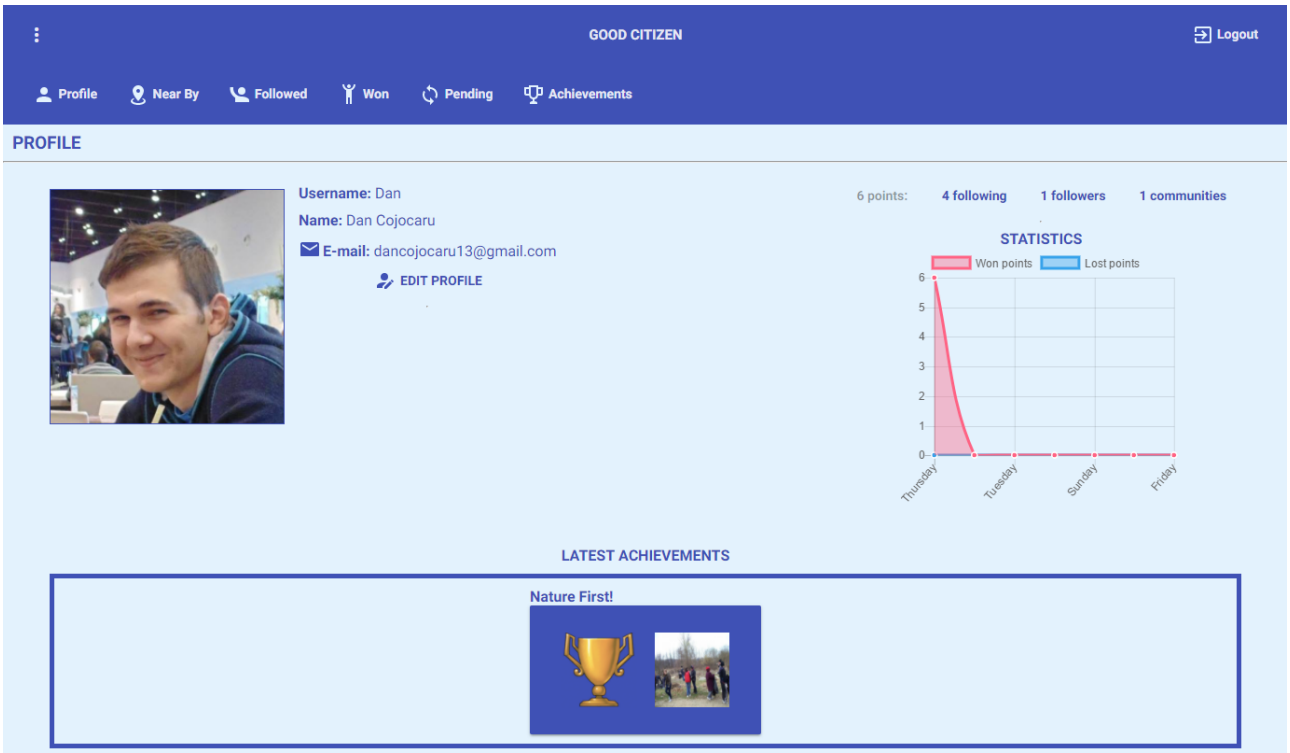

GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
SEARCH ACHIEVEMENTS
Search Achievement
City
Nature
Other
People
Clean city!
Category: City - Points: 60
Clean some area of a city.
TRY
Do something great!
Category: Other - Points: 2
In order to get this achievement you need to do something that others will appreciate it.
TRY

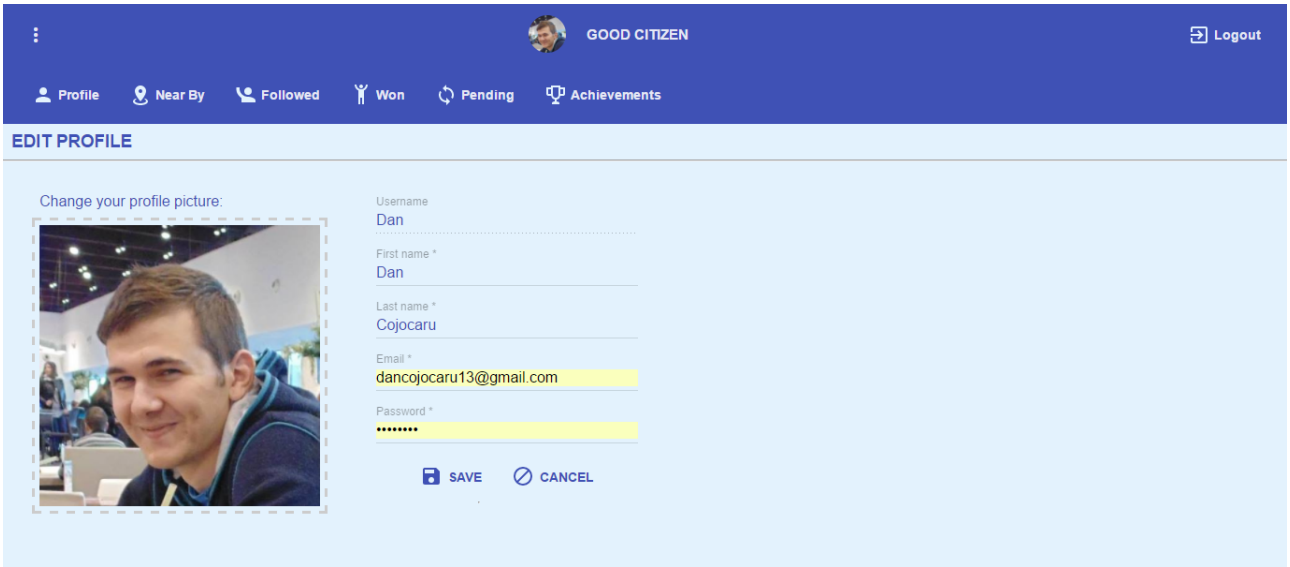

GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
Nature First!
Insert you toughts... *
I was part of an ecologization team.
Upload photo-proof for this achievement*
Change location
Padurea de Salcami, Tărtășești, Dâmboviț
Map data ©2017 Google
Terms of Use
Report a map error
ADD
CANCEL

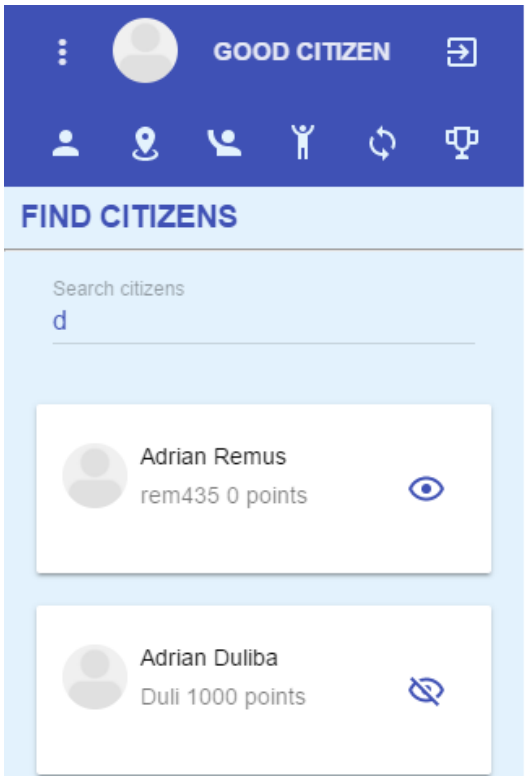

GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
PENDING ACHIEVEMENTS
Dan Cojocaru
Citizen - 0 points
Nature First!
Achievement - Nature - 6 points
Opened: 09:07:21 - 22/06/2017
I was part of an ecologization team.
Do something good for the nature, like cleaning out some area around you.
Likes: 0
Reports: 0
CANCEL
Dan Cojocaru
Citizen - 0 points
Clean city!
Achievement - City - 60 points
Opened: 07:35:01 - 22/06/2017

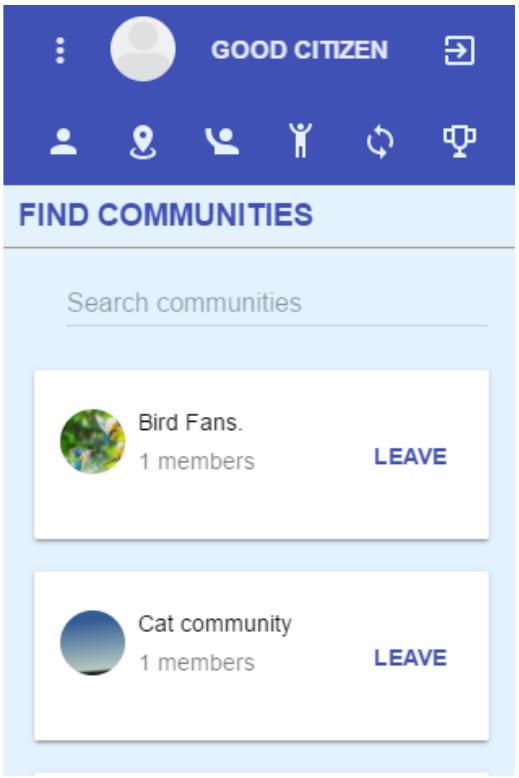

GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
FOLLOWED CITIZENS: PENDING
display won achievements
Adrian Duliba
Citizen - 1000 points
Homeless are people to!
Achievement - City - 60 points
Opened: 09:12:57 - 22/06/2017
I helped a homeless person today.
Help some homeless person with food, clothes or enything else.
Likes: 0
Reports: 0
LIKE
REPORT

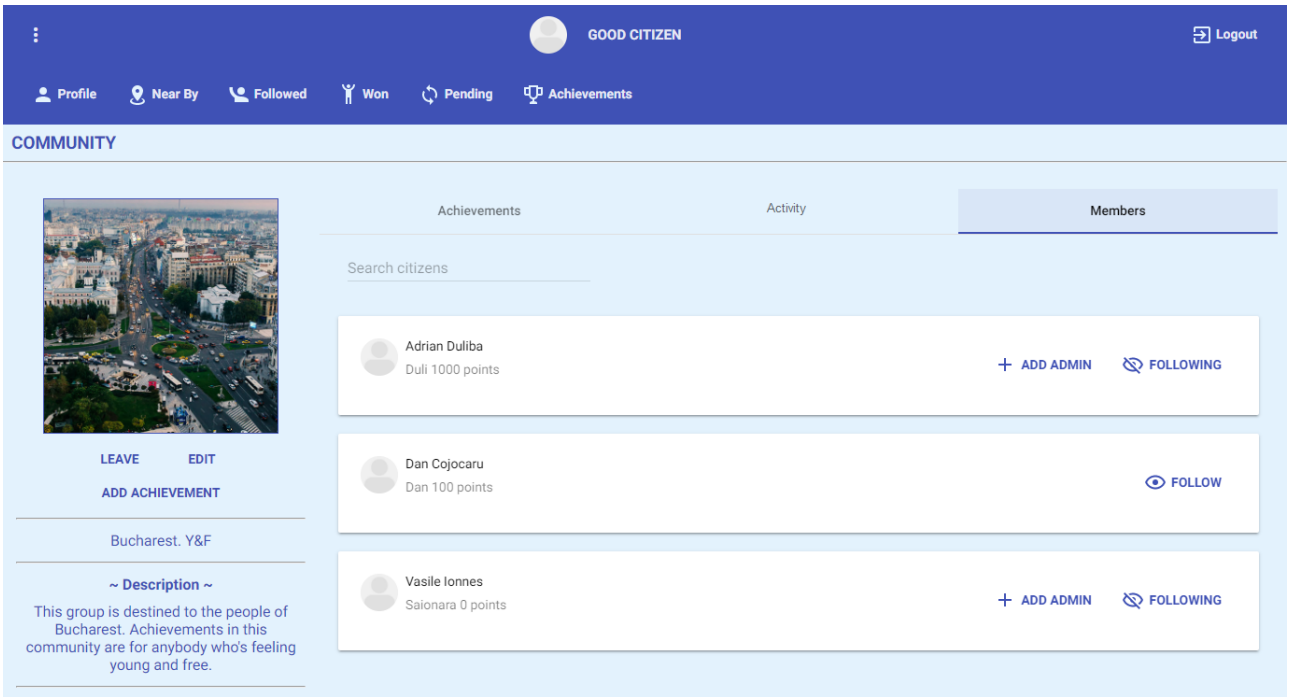

GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
NEAR BY CITIZENS: PENDING
39
display won achievements
Adrian Duliba
Citizen - 1000 points
It is all about the next generation!
Achievement - People - 22 points
Opened: 09:19:51 - 22/06/2017
I was a teacher today. Those kids are Awsome!
Teach someone younger then you about the importance of being a good citizen.
Likes: 0
Reports: 0
LIKE
REPORT

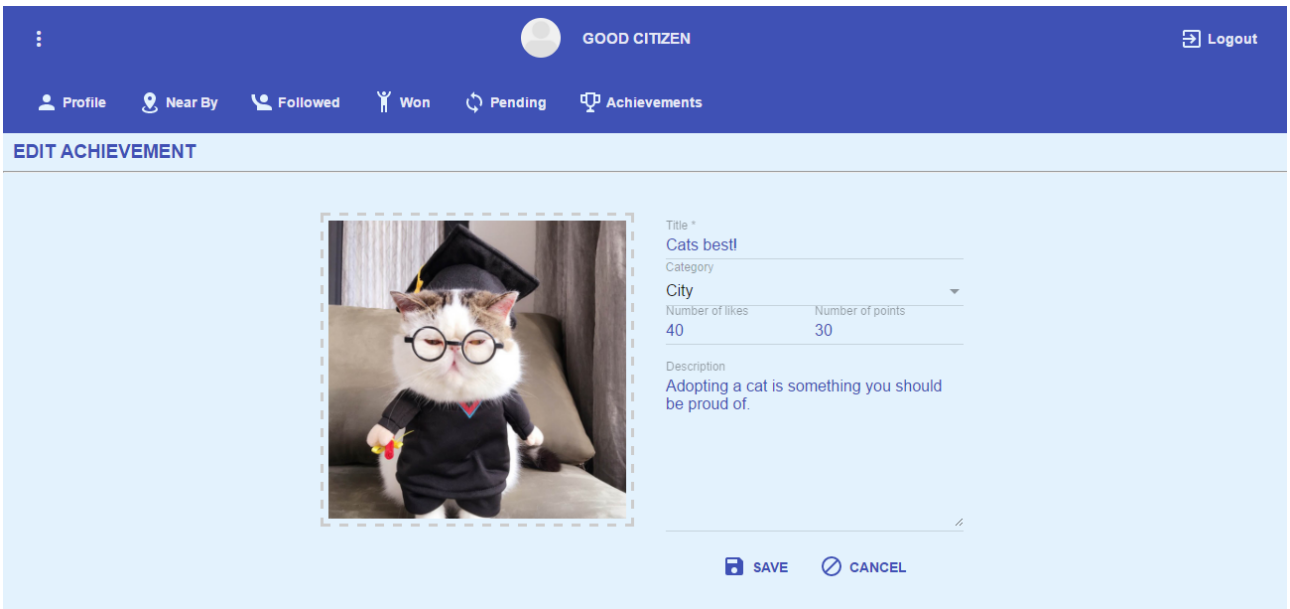

GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
PROFILE
Username: Dan
Name: Dan Cojocaru
E-mail: dancojocaru13@gmail.com
EDIT PROFILE
6 points:
4 following
1 followers
1 communities
STATISTICS
Won points
Lost points
6
5
4
3
2
1
0
Thursday
Tuesday
Sunday
Friday
LATEST ACHIEVEMENTS
Nature First!


GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
EDIT PROFILE
Change your profile picture:
Username
Dan
First name *
Dan
Last name *
Cojocaru
Email *
dancojocaru13@gmail.com
Password *
SAVE
CANCEL


GOOD CITIZEN
FIND CITIZENS
Search citizens
d
Adrian Remus
rem435 0 points
Adrian Duliba
Duli 1000 points


GOOD CITIZEN
FIND COMMUNITIES
Search communities
Bird Fans.
1 members
LEAVE
Cat community
1 members
LEAVE


GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
COMMUNITY
LEAVE
EDIT
ADD ACHIEVEMENT
Bucharest. Y&F
~ Description ~
This group is destined to the people of Bucharest. Achievements in this community are for anybody who's feeling young and free.
Achievements
Activity
Members
Search citizens
Adrian Duliba
Duli 1000 points
ADD ADMIN
FOLLOWING
Dan Cojocaru
Dan 100 points
FOLLOW
Vasile Ionnes
Saionara 0 points
ADD ADMIN
FOLLOWING


GOOD CITIZEN
Logout
Profile
Near By
Followed
Won
Pending
Achievements
EDIT ACHIEVEMENT
Title *
Cats best!
Category
City
Number of likes
40
Number of points
30
Description
Adopting a cat is something you should be proud of.
SAVE
CANCEL

Profile | Near By | Followed | Won | Pending | Achievements

COMMUNITY

LEAVE ADD ACHIEVEMENT

Iasi Area

~ Description ~

In this community are achievements focused on our beloved city, Iasi. Feel free to join this group to earn points by helping others. Welcome!

Achievements | Activity | Members

Search Achievement

☑ City ☐ Nature ☐ Other ☐ People

Cats best!
Category: City - Points: 30

Adopting a cat is something you should be proud of.

TRY EDIT

I all about culture!
Category: City - Points: 40

You have to be part of a cultural event around Iasi. Our city culture define us as citizens so you better win this achievement.

Profile | Near By | Followed | Won | Pending | Achievements

COMMUNITY

LEAVE

Iasi Area

~ Description ~

In this community are achievements focused on our beloved city, Iasi. Feel free to join this group to earn points by helping others. Welcome!

Achievements | Activity | Members

Search Achievement

☐ City ☐ Nature ☐ Other ☐ People display won achievements

Adrian Duliba
Citizen - 1000 points

Make a fellow citizen happy!
Achievement - People - 20 points

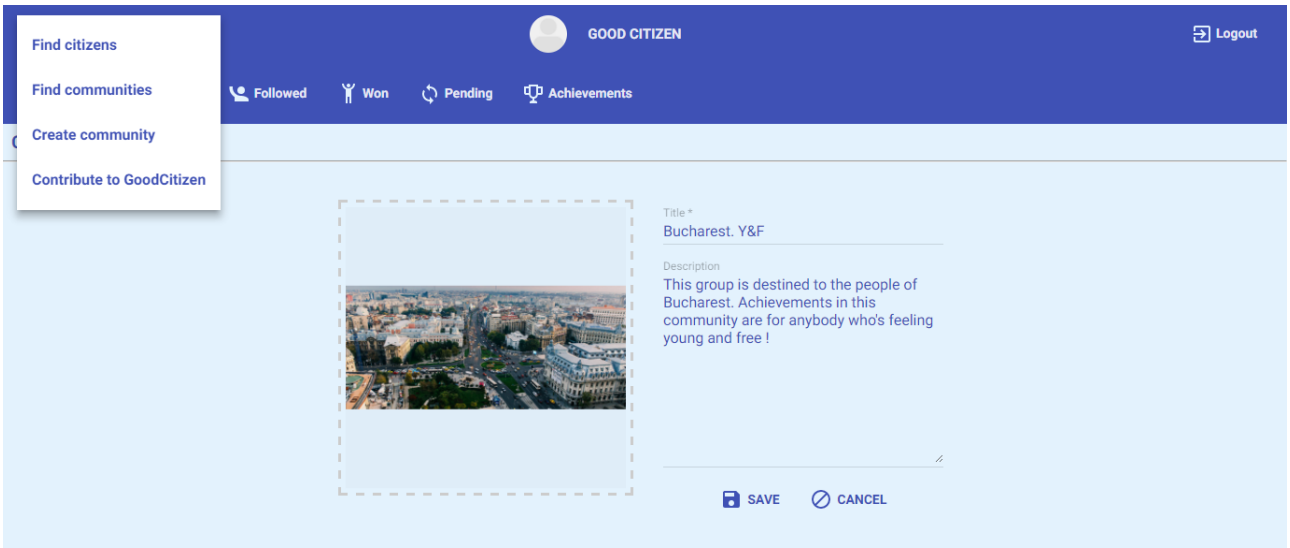
Opened: 04:39:15 - 22/06/2017

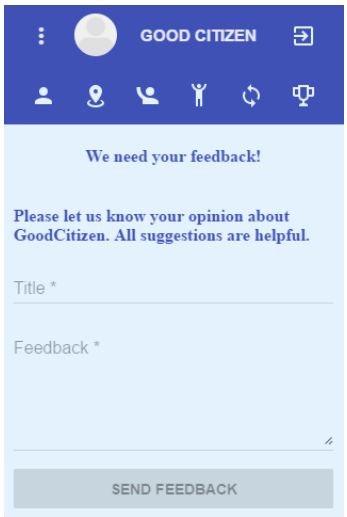
Today, I made this guy happy, by helping him with some food.

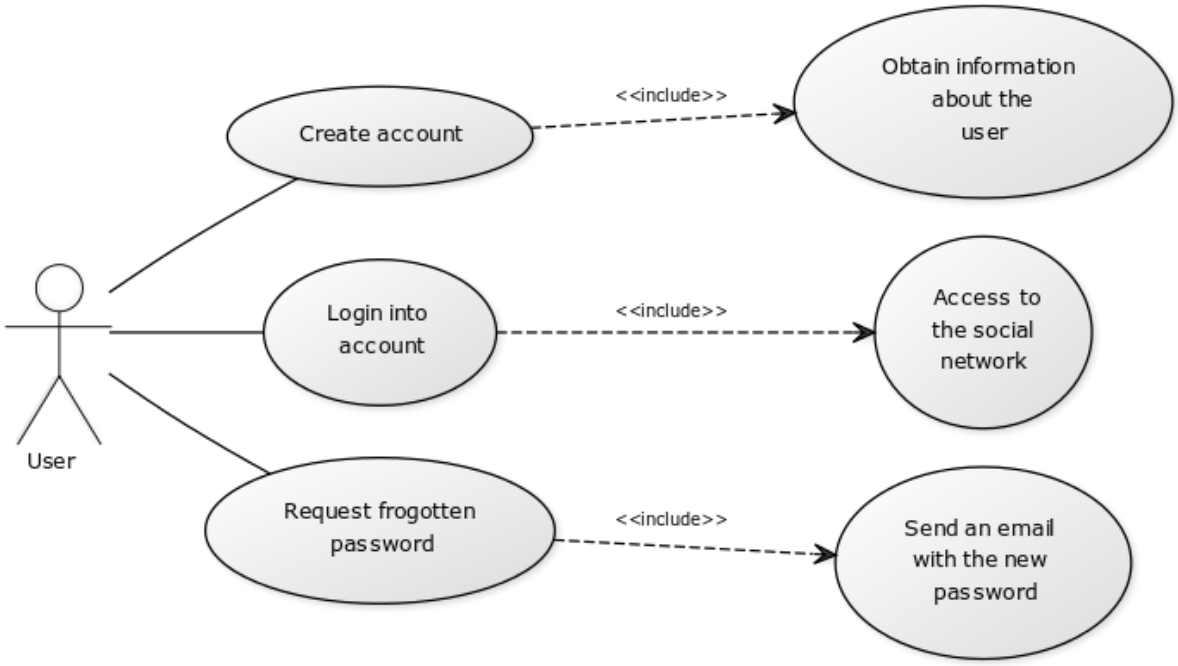
This achievment is a piece of cake. Just make someoane happy and get points.

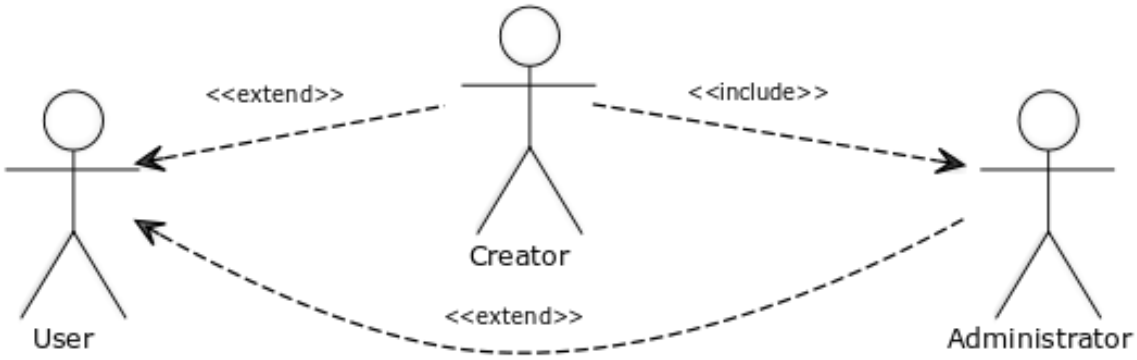
Likes: 1 Reports: 0

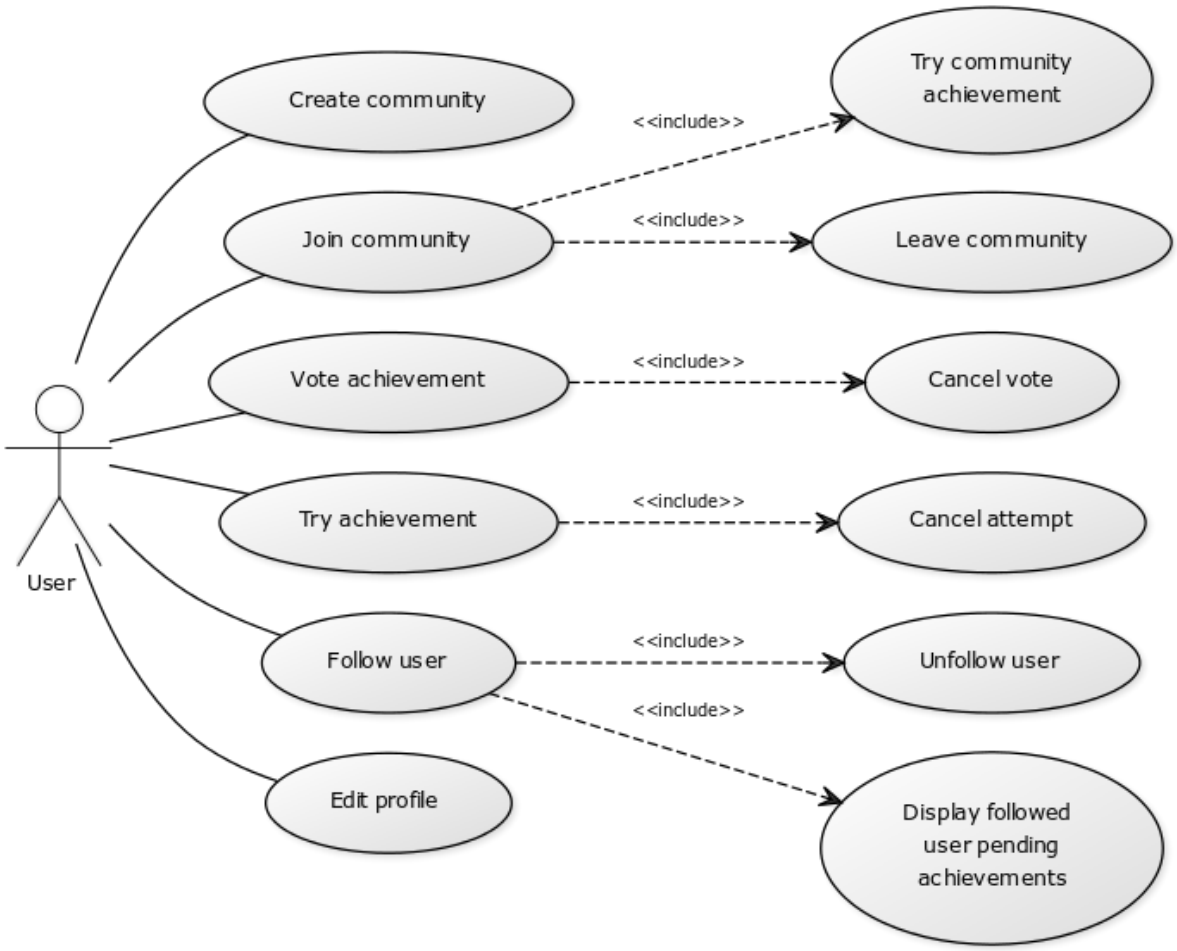
DISLIKE REPORT

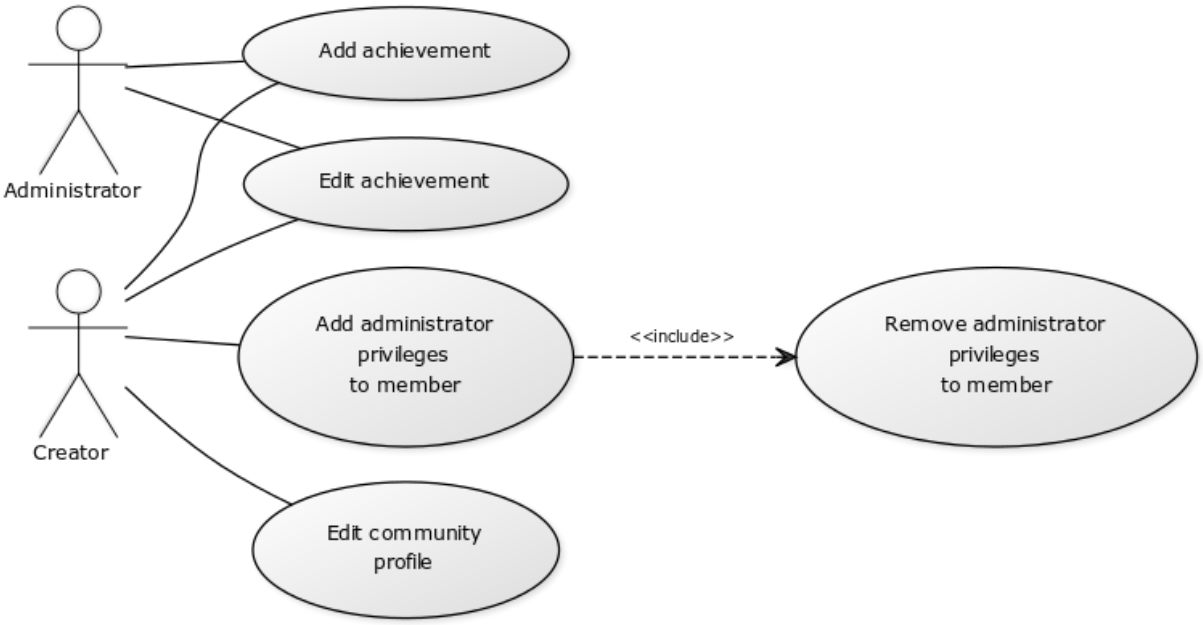

Find citizens
Find communities
Create community
Contribute to GoodCitizen
GOOD CITIZEN
Logout
Followed
Won
Pending
Achievements
Title *
Bucharest. Y&F
Description
This group is destined to the people of Bucharest. Achievements in this community are for anybody who's feeling young and free !
SAVE
CANCEL

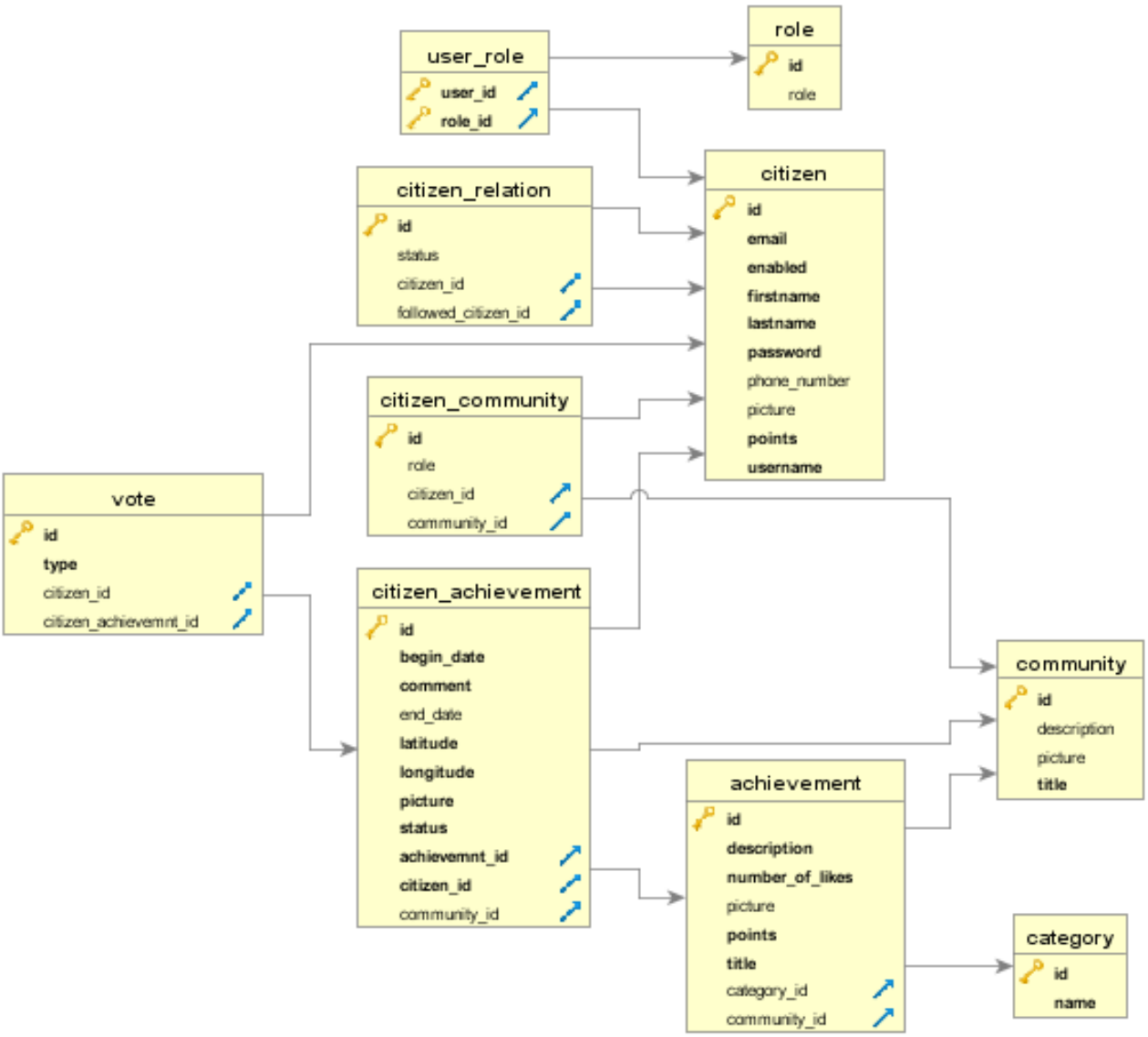

GOOD CITIZEN
We need your feedback!
Please let us know your opinion about GoodCitizen. All suggestions are helpful.
Title *
Feedback *
SEND FEEDBACK

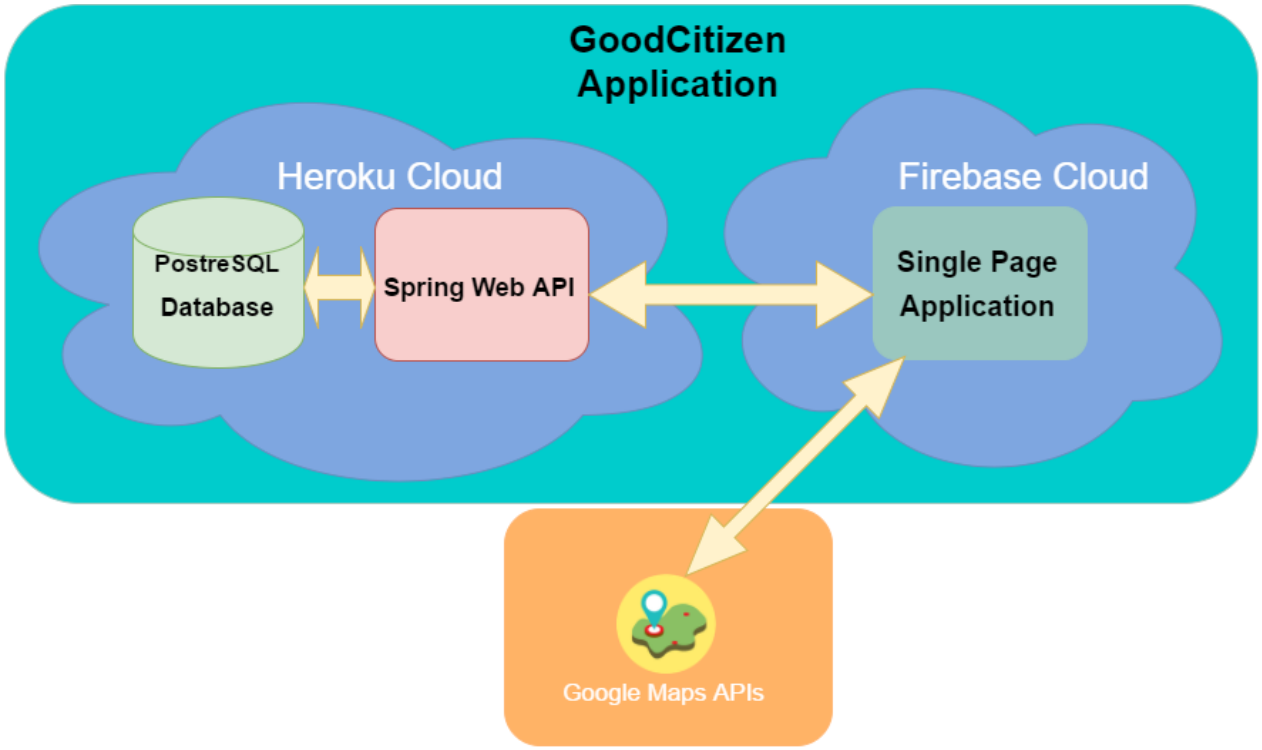

Create account
<<include>>
Obtain information about the user
Login into account
<<include>>
Access to the social network
User
Request frogotten password
<<include>>
Send an email with the new password


<<extend>>
<<include>>
Creator
User
<<extend>>
Administrator


User
Create community
Join community
Vote achievement
Try achievement
Follow user
Edit profile
Try community achievement
Leave community
Cancel vote
Cancel attempt
Unfollow user
Display followed user pending achievements
<<include>>

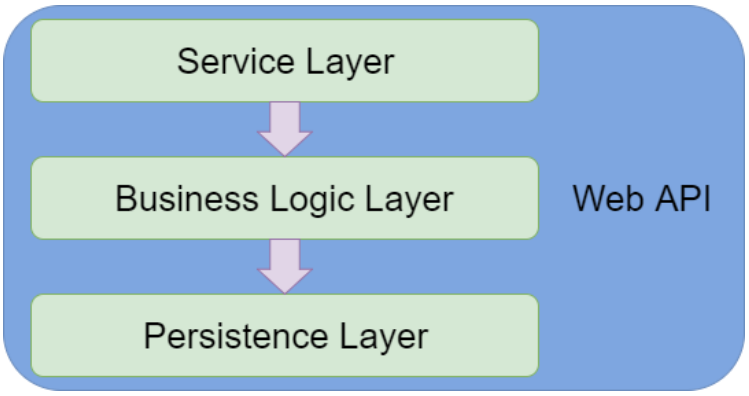

Administrator
Creator
Add achievement
Edit achievement
Add administrator privileges to member
Edit community profile
Remove administrator privileges to member
<<include>>


role
id
role
user_role
user_id
role_id
citizen
id
email
enabled
firstname
lastname
password
phone_number
picture
points
username
citizen_relation
id
status
citizen_id
followed_citizen_id
citizen_community
id
role
citizen_id
community_id
vote
id
type
citizen_id
citizen_achievemnt_id
citizen_achievement
id
begin_date
comment
end_date
latitude
longitude
picture
status
achievemnt_id
citizen_id
community_id
community
id
description
picture
title
achievement
id
description
number_of_likes
picture
points
title
category_id
community_id
category
id
name




```
@Entity
public class Achievement extends Base {

  @Column(unique = false, nullable = false)
  @Size(min = AchievementConstraints.titleMin, max = AchievementConstraints.
    titleMax)
```

```java
  private String title;

  @Column(unique = false, nullable = false)
  @Size(min = AchievementConstraints.descriptionMin, max =
    AchievementConstraints.descriptionMax)
  private String description;

  private byte[] picture;

  @ManyToOne
  @JoinColumn(name = "COMMUNITY_ID")
  private Community community;

  @ManyToOne
  @JoinColumn(name = "CATEGORY_ID")
  private Category category;

  @OneToMany(mappedBy = "achievement")
  private Set<CitizenAchievement> citizenAchievements;

  @Column(unique = false, nullable = false)
  @Min(AchievementConstraints.numberOfLikesMin)
  @Max(AchievementConstraints.numberOfLikesMax)
  private int numberOfLikes;

  //... + getters and setters
}
```



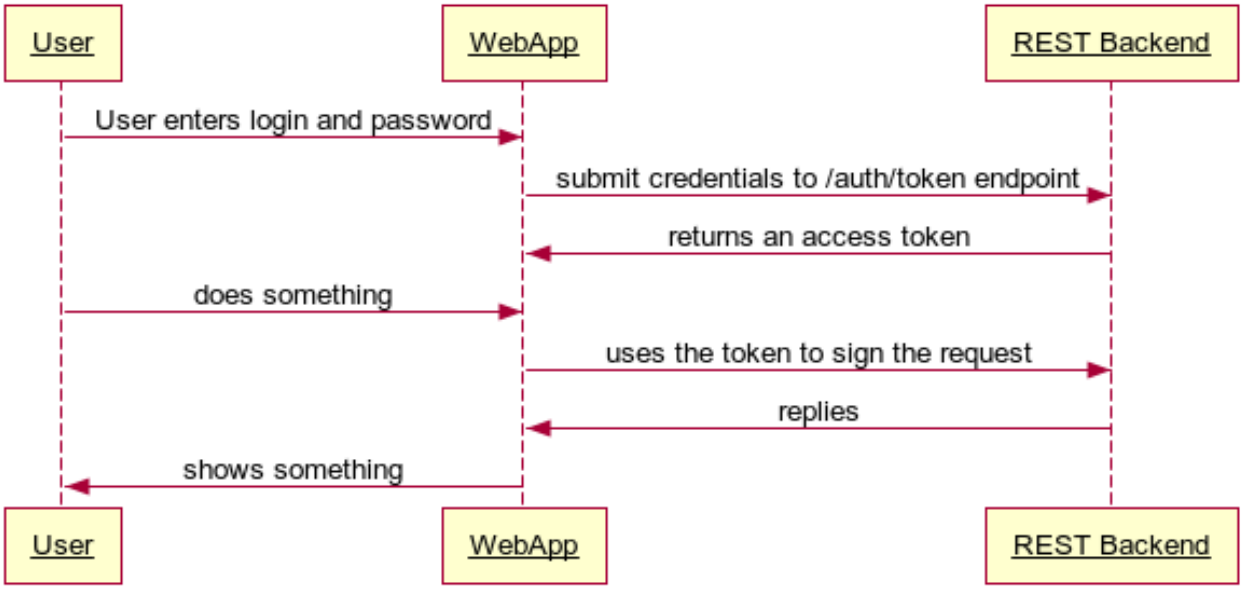

```java
public interface ICommunityRepository extends CrudRepository<Community, UUID> {

  @Query("SELECT c FROM Community c "
      + " WHERE "
      + "(LOWER(c.title) like CONCAT('%',:title,'%') or LOWER(c.description)
    like CONCAT('%',:description,'%')) "
      + "order by c.title ")
```

```java
  Page<Community> findByCriteria(
      @Param("title") String title,
      @Param("description") String description,
      Pageable pageable);

  @Query("SELECT c FROM Community c order by c.title asc")
  Page<Community> findAllPageable(Pageable pageable);
}
```

```java
@Transactional
public abstract class BaseService {

}

public interface ICommunityService {
  Page<CommunityThumbnailModel> findByCriteria(String term, int page, int size);

  //...
}

@Service
public class CommunityService extends BaseService implements ICommunityService {

  private final ICommunityRepository communityRepository;

    private final CommunityMapper communityMapper;

    private final ICitizenService citizenService;

    private final ICitizenCommunityRepository citizenCommunityRepository;

    public CommunityService(
        ICommunityRepository communityRepository,
        IAuthenticationFacade authenticationFacade,
        CommunityMapper communityMapper,
        ICitizenService citizenService,
        ICitizenCommunityRepository citizenCommunityRepository) {
        this.communityRepository = communityRepository;
        this.communityMapper = communityMapper;
        this.citizenService = citizenService;
        this.citizenCommunityRepository = citizenCommunityRepository;
    }

  @Override
  public Page<CommunityThumbnailModel> findByCriteria(String term, int page, int
     size) {
    if(!term.isEmpty()){
      term = term.toLowerCase();
      return communityRepository.findByCriteria(term, term,new PageRequest(page,
     size))
          .map(communityMapper::toCommunityThumbnail);
    }else{
      return communityRepository.findAllPageable(new PageRequest(page, size))
          .map(communityMapper::toCommunityThumbnail);
    }
  }
```

```
  //...
}
```

```
@Mapper(componentModel = "spring")
public interface AchievementMapper {

  @Mappings({
  @Mapping(source = "achievement.category.name", target="category"),
  @Mapping(source = "achievement.community.id", target="communityId")})
    AchievementThumbnailModel toAchievementModel(Achievement achievement);

  @Mappings({
    @Mapping(target = "achievement.category.name", source="category"),
    @Mapping(target = "achievement.community.id", source="communityId")})
  Achievement toAchievement(AchievementThumbnailModel achievement);

}
```




```
@RestController
@RequestMapping(value = "/email", produces = { "application/json" })
public class EmailController extends BaseController {

  private final IResetPasswordService resetPasswordService;

  private final IFeedBackService feedbackService;

  public EmailController(
      IResetPasswordService resetPasswordService,
```

```
      IFeedBackService feedbackService) {
    this.resetPasswordService = resetPasswordService;
    this.feedbackService = feedbackService;
  }

  @PatchMapping(value = "/reset-password")
    public ResponseEntity<?> resetPassword(
        @Valid @RequestBody @Email String email,Errors errors){

    ResponseEntity<?> response = this.buildErrorMessage(errors);
    if(response == null){
        return ResponseEntity
          .status(HttpStatus.OK)
          .body(resetPasswordService.resetPassword(email));
    }

    return response;
  }

  @PostMapping(value = "/feedback")
  public void sendFeedBack(@RequestBody FeedbackModel feedbackModel){
    this.feedbackService.sendFeedBack(feedbackModel);
  }
}
```

```
  @NgModule({
  providers: [

    AuthService,
    AnonymousGuard,
    AuthenticatedGuard,
    AchievementService,
    CategoryService,
    CitizenService,
    CitizenAchievementService,
    CommunityService,
    FeedbackService],

  declarations: [
    AppComponent,
    NavigationComponent,
    PageHeaderComponent,
    AlertDialogComponent,
    ConfirmDialogComponent,
  //...more components
```

```
  ],

  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    FormsModule,
    ReactiveFormsModule,
    ChartsModule,

    HttpModule,
    routing,
    NgbModule.forRoot(),
    MaterialModule.forRoot(),
    AgmCoreModule.forRoot({
      apiKey: 'AIzaSyBZDeaviSbBrrg9gdLQeZYbsnhD15RvMZQ',
      libraries: ["places"]
    })
  ],
  entryComponents: [
    AlertDialogComponent,
    ConfirmDialogComponent
  ],

  bootstrap: [AppComponent]
})
export class AppModule {
}
```

```
<div  class="card-container">
    <div class="title-ach">
      <b>{{citizenAchievement.achievementTitle}}</b>
        </div>
        <md-card>
            <md-card-content>
                <div class="image-container">
                    <img [src]="!citizenAchievement.achievementPicture ?
                                defaultAchievementPicture :
                                '' + citizenAchievement.
   achievementPicture">
                </div>
                <div class="image-container">
                    <img [src]="''+ citizenAchievement.
   picture">
                </div>
            </md-card-content>
        </md-card>
</div>
```

```
#citizenPage{
    padding-bottom: 40px;
    .citizen-filter-container{
        flex-wrap: nowrap  wrap  wrap-reverse;
        justify-content: space-between;
        padding-left: 20px;
        padding-right: 20px;
```

```
        .mat-input-container{
            padding:10px;
            input{
                width: 240px;
            }
        }
    //...

    }
}
```

https://goodcitizen-ap

No Environment

POST

https://goodcitizen-api.herokuapp.com/oauth/token?grant_type=password&username=Dan&password=IgSwcchhIN

Params | Send | Save

Authorization | Headers (2) | Body | Pre-request Script | Tests | Code

Key	Value	Description
Authorization	Basic Z29vZGNpdGl6ZW46Z29vZGNpdGl6ZW...	
Accept	application/json	
New key	Value	Description

Bulk Edit | Presets

Body | Cookies | Headers (16) | Tests

Status: 200 OK | Time: 176 ms

Pretty | Raw | Preview | JSON


```
{
    "access_token": "319dd672-2295-49c4-acee-665110bc770b",
    "token_type": "bearer",
    "refresh_token": "1f8be240-31c5-48af-89f9-f066c5ca237e",
    "expires_in": 29486,
    "scope": "USER"
}
```

$\rightarrow$

$\rightarrow$

$\rightarrow$

$\rightarrow$

$\rightarrow$

$\rightarrow$

$\rightarrow$

```java
@MappedSuperclass
public abstract class Base {
    @Id
    private UUID id;

    public Base() {
        id = java.util.UUID.randomUUID();
    }
    public UUID getId() {
        return id;
    }
    public void setId(UUID id) {
        this.id = id;
    }
}
```

```java
@Entity
@Table(name = "VOTE", uniqueConstraints = @UniqueConstraint(columnNames = {
"CITIZEN_ID","CITIZEN_ACHIEVEMNT_ID","TYPE"}))
public class Vote extends Base {
    @NotNull
    private String type;

    @ManyToOne
    @JoinColumn(name = "CITIZEN_ACHIEVEMNT_ID", nullable = true)
    private CitizenAchievement citizenAchievement;

    @ManyToOne
    @JoinColumn(name = "CITIZEN_ID", nullable= true)
    private Citizen citizen;

    //...+ setters and getters
}
```

```java
@Entity
```

```java
@Table(name="CITIZEN_ACHIEVEMENT")
public class CitizenAchievement extends Base {

    @ManyToOne
    @JoinColumn(name = "CITIZEN_ID")
    @NotNull
    private Citizen citizen;

    @ManyToOne
    @JoinColumn(name = "ACHIEVEMNT_ID")
    @NotNull
    private Achievement achievement;

    @ManyToOne
    @JoinColumn(name = "COMMUNITY_ID")
    private Community community;

    @OneToMany(mappedBy="citizenAchievement")
    private Set<Vote> votes;

    @NotNull
    private Date beginDate;

    private Date endDate;

    @Column(unique = false, nullable = false)
    @Size(min = CitizenAchievementConstraints.commentMin, max =
CitizenAchievementConstraints.commentMax)
    private String comment;

    @NotNull
    private double longitude ;

    @NotNull
    private double latitude;

    @NotNull
    private byte[] picture;

    @NotNull
    private String status;

    //...+ setters and getters
}
```

```java
@Entity
@Table(name = "CITIZEN_COMMUNITY", uniqueConstraints = @UniqueConstraint(
columnNames = { "CITIZEN_ID",
        "COMMUNITY_ID" }))
public class CitizenCommunity extends Base {

    @ManyToOne
    @JoinColumn(name = "CITIZEN_ID")
    private Citizen citizen;

    @ManyToOne
    @JoinColumn(name = "COMMUNITY_ID")
    private Community community;
```

```java
        private String role;

        //...+ setters and getters
    }
```

```java
    @Entity
    @Table(name = "CITIZEN_RELATION", uniqueConstraints = @UniqueConstraint(
   columnNames = { "CITIZEN_ID",
            "FOLLOWED_CITIZEN_ID" }))
    public class CitizenRelation extends Base {

        @ManyToOne
        @JoinColumn(name = "CITIZEN_ID")
        private Citizen citizen;

        @ManyToOne
        @JoinColumn(name = "FOLLOWED_CITIZEN_ID")
        private Citizen followedCitizen;

        private String status;

        //...+ setters and getters
    }
```

```java
    public interface CitizenConstraints {
        // username
        int usernameMin = 2;
        int usernameMax = 15;

        // password
        int passwordMin = 8;
        int passwordMax = 50;

        //firstname
        int firstnameMin = 2;
        int firstnameMax = 15;

        //lastname
        int lastnameMin = 2;
        int lastnameMax = 15;
    }
```

```java
    public interface AchievementConstraints {

        int titleMax = 40;
        int titleMin = 3;
```

```
        int pointsMax = 1000;
        int pointsMin = 1;

        int descriptionMin = 0;
        int descriptionMax = 255;

        int numberOfLikesMin = 1;
        int numberOfLikesMax = 1000;

    }
```

```
public interface IAchievementRepository extends CrudRepository<Achievement, UUID
    > {
  Optional<Achievement> findByTitle(String title);

  Stream<Achievement> findByDescriptionLike(String description);

  @Query("SELECT a FROM Achievement a "
      + "inner join a.category c"
      + " WHERE "
      + " (LOWER(a.title) like CONCAT('%',:title,'%') or LOWER(a.description)
    like CONCAT('%',:description,'%')) "
      + " and c.name in :categories "
      + " and a.community.id is null "
      + " order by a.title ")
  Page<Achievement> findByCriteria(
      @Param("title") String title,
      @Param("description") String description,
      @Param("categories") Set<String> categories,
      Pageable pageable);

  @Query("SELECT a FROM Achievement a "
      + "inner join a.category c"
      + " WHERE "
      + " (LOWER(a.title) like CONCAT('%',:title,'%') or LOWER(a.description)
    like CONCAT('%',:description,'%')) "
      + " and c.name in :categories "
      + " and a.community.id = :communityId "
      + " order by a.title ")
  Page<Achievement> findByCriteriaWithCommunity(
      @Param("title") String title,
      @Param("description") String description,
      @Param("categories") Set<String> categories,
      @Param("communityId") UUID communityId,
      Pageable pageable);

  @Query("SELECT a FROM Achievement a "
      + " inner join a.category c "
      + " WHERE "
      + " c.name in :categories "
      + " and a.community.id is null "
      + " order by a.title ")
  Page<Achievement> findByCategories(
```

```
        @Param("categories") Set<String> categories,
        Pageable pageable);

    @Query("SELECT a FROM Achievement a "
        + " inner join a.category c "
        + " WHERE "
        + " c.name in :categories "
        + " and a.community.id = :communityId "
        + " order by a.title ")
    Page<Achievement> findByCategoriesWithCommunity(
        @Param("categories") Set<String> categories,
        @Param("communityId") UUID communityId,
        Pageable pageable);
}
```

```
public interface ICitizenAchievementRepository extends CrudRepository<
    CitizenAchievement, UUID> {

    int countByStatusAndCitizen(String status, Citizen citizen);

    boolean existsByAchievementAndCitizenAndStatus(Achievement Achievement,Citizen
      citizen, String status);

    @Query("SELECT ca FROM CitizenAchievement ca "
        + " WHERE "
        + " ca.citizen.id in :friendsIds"
        + " and ca.status = :status")
    Page<CitizenAchievement> getFriendsAchievements(
        @Param("friendsIds") Set<UUID> friendsIds,
        @Param("status") String status,
        Pageable pageable);


    @Query("SELECT ca FROM CitizenAchievement ca "
        + " inner join ca.achievement a "
        + " inner join a.category cat "
        + " WHERE "
        + " (LOWER(a.title) like CONCAT('%',:title,'%') or LOWER(a.description)
     like CONCAT('%',:description,'%')) "
        + " and cat.name in :categories "
        + " and ca.community.id = :communityId "
        + " and ca.status = :status")
    Page<CitizenAchievement> findByCriteria(
        @Param("title") String title,
        @Param("description") String description,
        @Param("categories") Set<String> categories,
        @Param("communityId") UUID communityId,
        @Param("status") String status,
        Pageable pageable);

    @Query("SELECT ca FROM CitizenAchievement ca "
        + " WHERE "
        + " ca.achievement.category.name in :categories "
        + " and ca.community.id = :communityId "
        + " and ca.status = :status")
    Page<CitizenAchievement> findByCategories(
        @Param("categories") Set<String> categories,
        @Param("communityId") UUID communityId,
```

```
        @Param("status") String status,
        Pageable pageable);

    @Query("SELECT ca FROM CitizenAchievement ca "
        + " WHERE "
        + " ca.citizen.id = :citizenId "
        + " and ca.status = :status ")
    Page<CitizenAchievement> getCurrentCitizenAchievements(
        @Param("citizenId") UUID citizenId,
        @Param("status") String status,
        Pageable pageable);

    @Query("SELECT ca FROM CitizenAchievement ca "
        + " WHERE "
        +" (6371*2*atan2("
        +"    sqrt( "
        +"       (sin(radians(ca.latitude - :latitude)/2 )) "
        +"        *(sin(radians(ca.latitude - :latitude)/2 )) "
        +"        +(sin(radians(ca.longitude - :longitude)/2)) "
        +"      *(sin(radians(ca.longitude - :longitude)/2)) "
        +"      *(cos(radians(ca.latitude))) "
        +"      *(cos(radians( :latitude ))) "
        +"      ),"
        +"  1-sqrt("
        +"       (sin(radians(ca.latitude - :latitude)/2 )) "
        +"      *(sin(radians(ca.latitude - :latitude)/2 )) "
        +"          +(sin(radians(ca.longitude - :longitude)/2)) "
        +"      *(sin(radians(ca.longitude - :longitude)/2)) "
        +"      *(cos(radians(ca.latitude))) "
        +"      *(cos(radians( :latitude ))) "
        +"        ) "
        +" )) < :distance"
        +" and ca.citizen.id != :citizenId"
        +" and ca.status = :status")
    Page<CitizenAchievement> findNearBy(
        @Param("latitude") double latitude,
        @Param("longitude") double longitude,
        @Param("distance") int distance,
        @Param("citizenId") UUID citizenId,
        @Param("status") String status,
        Pageable pageable);
    @Query("SELECT ca FROM CitizenAchievement ca "
        + " WHERE "
        + " ca.citizen.id = :citizenId "
        + " and ca.status = :status "
        + " and ca.endDate > current_date - :numberOfDays")
    Stream<CitizenAchievement> getCitizenAchievementFromLastDays(
        @Param("citizenId") UUID citizenId,
        @Param("status") String status,
        @Param("numberOfDays") int numberOfDays);

    Boolean existsByIdAndStatus(
        @Param("id") UUID id,
        @Param("status") String status);
}
```

```java
public interface IAchievementService {

    Set<AchievementThumbnailModel> getAll();

  Page<AchievementThumbnailModel> findByCriteria(String criteria,Set<String>
    categories,String communityId,int page, int size);

  Optional<AchievementThumbnailModel> findById(String id);

  AchievementThumbnailModel save(AchievementThumbnailModel achievementModel);

}

@Service
public class AchievementService extends BaseService implements
    IAchievementService {

    private final IAchievementRepository achievementRepository;

    private final AchievementMapper achievementMapper;

    private final ICategoryRepository categoryRepository;

    private final ICitizenService citizenService;

    private final ICitizenCommunityService citizenCommunityService;

    public AchievementService(
        IAchievementRepository achievementRepository,
        ICategoryRepository categoryRepository,
        AchievementMapper achievementMapper,
        ICitizenService citizenService,
        ICitizenCommunityService citizenCommunityService) {
        this.achievementRepository = achievementRepository;
        this.achievementMapper = achievementMapper;
        this.categoryRepository = categoryRepository;
        this.citizenService = citizenService;
        this.citizenCommunityService = citizenCommunityService;
    }

    public Set<AchievementThumbnailModel> getAll() {
        return StreamSupport.stream((Spliterator<Achievement>)
                achievementRepository.findAll()
                .spliterator(), Boolean.FALSE)
                .map(achievementMapper::toAchievementModel)
                .collect(toSet());
    }

  @Override
  public Page<AchievementThumbnailModel> findByCriteria(
        String criteria,Set<String> categories,String communityId,int page,int
    size) {

    if(categories==null || categories.isEmpty()){
      categories = StreamSupport.stream(
          (Spliterator<Category>)categoryRepository.findAll().spliterator(),
```

```
  Boolean.FALSE)
        .map(Category::getName)
        .collect(toSet());
   }

   if(communityId != null){

     if(criteria == null || criteria.isEmpty()){
        return achievementRepository
           .findByCategoriesWithCommunity(
                           categories,UUID.fromString(communityId), new
  PageRequest(page, size))
           .map(achievementMapper::toAchievementModel);
     }
     return achievementRepository
         .findByCriteriaWithCommunity(
                       criteria.toLowerCase(),criteria.toLowerCase(),
                       categories,UUID.fromString(communityId),new PageRequest(
  page, size))
        .map(achievementMapper::toAchievementModel);
   }else{
     if(criteria == null || criteria.isEmpty()){
        return achievementRepository
           .findByCategories(categories, new PageRequest(page, size))
           .map(achievementMapper::toAchievementModel);
     }
     return achievementRepository
         .findByCriteria(
                       criteria.toLowerCase(),criteria.toLowerCase(),categories
  ,new PageRequest(page, size))
        .map(achievementMapper::toAchievementModel);
   }
 }

 @Override
 public Optional<AchievementThumbnailModel> findById(String id) {
    return Optional.of(achievementMapper.toAchievementModel(
            achievementRepository.findOne(UUID.fromString(id))));
 }

 @Override
 public AchievementThumbnailModel save(AchievementThumbnailModel
  achievementModel) {

   String currentCitizenId = this.citizenService.getLoggedUserId().toString();
   Boolean isAdmin = this.citizenCommunityService.checkIfRole(
           achievementModel.getCommunityId().toString(), currentCitizenId,
  GlobalConstants.ADMIN);
   Boolean isCreator = this.citizenCommunityService.checkIfRole(
           achievementModel.getCommunityId().toString(), currentCitizenId,
  GlobalConstants.CREATOR);
   if( isAdmin || isCreator){
     if(achievementModel.getId() == null){
       achievementModel.setId(UUID.randomUUID());
     }
     String categoryName = achievementModel.getCategory();
     if(achievementModel.getCategory() == null){
       categoryName = GlobalConstants.CATEGORY_DEFAULT;
     }
     if(achievementModel.getPoints() <= GoodCitizenGameConstraints.points
           || achievementModel.getPoints() >= GoodCitizenGameConstraints.
```

```
    maxPoints){
        achievementModel.setPoints(GoodCitizenGameConstraints.points);
      }
      if(achievementModel.getNumberOfLikes() <= GoodCitizenGameConstraints.
    numberOfLikes
            || achievementModel.getNumberOfLikes() >= GoodCitizenGameConstraints
    .maxLikes){
        achievementModel.setNumberOfLikes(GoodCitizenGameConstraints.
    numberOfLikes);
      }


      Optional<Category> category = this.categoryRepository.findByName(
    categoryName);
      Achievement achievment = achievementMapper.toAchievement(achievementModel)
    ;
      achievment.setCategory(category.get());

      return this.achievementMapper.toAchievementModel(
                this.achievementRepository.save(achievment));
    }
    else{
      throw new UnauthorizedUserException("You are not authorized to add/change
    this achievment!");
    }
  }
}
```

```
public interface ICitizenRelationService {

  boolean followUnfollow(CitizenRelationFollowModel citizenRelationFollowModel);

  Set<CitizenThumbnailModel> getFollowers(String id);

  Boolean checkIfExists(String citizenId,String followedId);

  public Set<CitizenThumbnailModel> getFollowed(String id);
}

@Service
public class CitizenRelationService extends BaseService implements
    ICitizenRelationService {

  private final ICitizenRelationRepository citizenRelationRepository;

  private final ICitizenService citizenService;

  private final CitizenMapper citizenMapper;

  private final CitizenRelationMapper citzenRelationMapper;
  public CitizenRelationService(
      ICitizenRelationRepository citizenRelationRepository,
      ICitizenService citizenService,
      CitizenRelationMapper citzenRelationMapper,
      CitizenMapper citizenMapper){
    this.citizenRelationRepository = citizenRelationRepository;
    this.citizenService = citizenService;
    this.citzenRelationMapper = citzenRelationMapper;
```

```
    this.citizenMapper = citizenMapper;
  }

  @Override
  public boolean followUnfollow(CitizenRelationFollowModel
   citizenRelationFollowModel) {
    Optional<CitizenRelation> citizenRelation =
            citizenRelationRepository.getByCitizenAndFollowed(
                citizenService.getLoggedUserId(),
                citizenRelationFollowModel.getFollowedId());
    if(citizenRelation.isPresent()){
      citizenRelationRepository.delete(citizenRelation.get().getId());
      return false;
    }else{
      citizenRelationFollowModel.setId(UUID.randomUUID());
      citizenRelationRepository
                .save(citzenRelationMapper
                .fromCitizenRelationFollow(citizenRelationFollowModel));
      return true;
    }

  }

  public Set<CitizenThumbnailModel> getFollowers(String id){
    return citizenRelationRepository
                 .getFollowers(UUID.fromString(id))
                 .map(CitizenRelation::getCitizen)
                 .map(citizenMapper::toCitizenThumbnail)
                 .collect(Collectors.toSet());
  }

  public Set<CitizenThumbnailModel> getFollowed(String id){
    return citizenRelationRepository
                 .getFollowed(UUID.fromString(id))
                 .map(CitizenRelation::getFollowedCitizen)
                 .map(citizenMapper::toCitizenThumbnail)
                 .collect(Collectors.toSet());
  }

  public Boolean checkIfExists(String citizenId,String followedId){
    Optional<CitizenRelation> citizenRelation =
        citizenRelationRepository
        .getByCitizenAndFollowed(UUID.fromString(citizenId)
                                      ,UUID.fromString(followedId));
    if(citizenRelation.isPresent()){
      return Boolean.TRUE;
    }
    return Boolean.FALSE;
  }

}
```

```
@Generated(
    value = "org.mapstruct.ap.MappingProcessor",
```

```
    date = "2017-06-20T22:00:19+0300",
    comments = "version: 1.2.0.Beta2, compiler: javac, environment: Java 1.8.0
   _101 (Oracle Corporation)"
)
@Component
public class AchievementMapperImpl implements AchievementMapper {

    @Override
    public AchievementThumbnailModel toAchievementModel(Achievement achievement)
    {
        if ( achievement == null ) {
            return null;
        }

        AchievementThumbnailModel achievementThumbnailModel = new
   AchievementThumbnailModel();

        String name = achievementCategoryName( achievement );
        if ( name != null ) {
            achievementThumbnailModel.setCategory( name );
        }
        UUID id = achievementCommunityId( achievement );
        if ( id != null ) {
            achievementThumbnailModel.setCommunityId( id );
        }
        achievementThumbnailModel.setId( achievement.getId() );
        byte[] picture = achievement.getPicture();
        if ( picture != null ) {
            achievementThumbnailModel.setPicture( Arrays.copyOf( picture,
   picture.length ) );
        }
        achievementThumbnailModel.setTitle( achievement.getTitle() );
        achievementThumbnailModel.setDescription( achievement.getDescription() )
   ;
        achievementThumbnailModel.setPoints( achievement.getPoints() );
        achievementThumbnailModel.setNumberOfLikes( achievement.getNumberOfLikes
   () );
        return achievementThumbnailModel;
    }

    @Override
    public Achievement toAchievement(AchievementThumbnailModel achievement) {
        if ( achievement == null ) {
            return null;
        }
        Achievement achievement1 = new Achievement();
        achievement1.setCategory( achievementThumbnailModelToCategory(
   achievement ) );
        achievement1.setCommunity( achievementThumbnailModelToCommunity(
   achievement ) );
        achievement1.setId( achievement.getId() );
        achievement1.setNumberOfLikes( achievement.getNumberOfLikes() );
        achievement1.setTitle( achievement.getTitle() );
        achievement1.setDescription( achievement.getDescription() );
        byte[] picture = achievement.getPicture();
        if ( picture != null ) {
            achievement1.setPicture( Arrays.copyOf( picture, picture.length ) );
        }
        achievement1.setPoints( achievement.getPoints() );

        return achievement1;
```

```java
    }

    private String achievementCategoryName(Achievement achievement) {
        if ( achievement == null ) {
            return null;
        }
        Category category = achievement.getCategory();
        if ( category == null ) {
            return null;
        }
        String name = category.getName();
        if ( name == null ) {
            return null;
        }
        return name;
    }

    private UUID achievementCommunityId(Achievement achievement) {
        if ( achievement == null ) {
            return null;
        }
        Community community = achievement.getCommunity();
        if ( community == null ) {
            return null;
        }
        UUID id = community.getId();
        if ( id == null ) {
            return null;
        }
        return id;
    }

    protected Category achievementThumbnailModelToCategory(
        AchievementThumbnailModel achievementThumbnailModel) {
        if ( achievementThumbnailModel == null ) {
            return null;
        }

        Category category = new Category();

        category.setName( achievementThumbnailModel.getCategory() );

        return category;
    }

    protected Community achievementThumbnailModelToCommunity(
        AchievementThumbnailModel achievementThumbnailModel) {
        if ( achievementThumbnailModel == null ) {
            return null;
        }

        Community community = new Community();

        community.setId( achievementThumbnailModel.getCommunityId() );

        return community;
    }
}
```

```java
public abstract class BaseController {

    protected static Message responseMessage = new Message();

    protected ResponseEntity<?> buildErrorMessage(Errors errors){
    if (errors.hasErrors()) {
      return ResponseEntity
          .badRequest()
          .body(responseMessage
              .setMessage(
              errors.getAllErrors()
              .stream()
              .map(ObjectError::getDefaultMessage)
              .collect(joining(" "))));
    }
    return null;
    }
}

@RestController
@RequestMapping(value = "/citizen-relation", produces = { "application/json" })
public class CitizenRelationController extends BaseController {

  private final ICitizenRelationService citizenRelationService;

  public CitizenRelationController(ICitizenRelationService
    citizenRelationService) {
    this.citizenRelationService = citizenRelationService;
  }

  @PatchMapping
  public ResponseEntity<?> followUnfollow(
        @Valid @RequestBody CitizenRelationFollowModel
    citizenRelationFollowModel, Errors errors) {

    ResponseEntity<?> response = this.buildErrorMessage(errors);
    if(response == null){
      if(citizenRelationService.followUnfollow(citizenRelationFollowModel)){
        return ResponseEntity
            .status(HttpStatus.CREATED)
            .body(responseMessage.setMessage("Citizen is followed!"));
      }
      return ResponseEntity
          .status(HttpStatus.NO_CONTENT)
          .body(responseMessage.setMessage("Unfollowed citizen"));

    }
    return response;
  }

  @GetMapping(value = "followers/")
    public Set<CitizenThumbnailModel> getCitizenFollowers(
        @RequestParam(value="id", required = true) String id){
      return citizenRelationService.getFollowers(id);
    }

  @GetMapping(value = "followed/")
    public Set<CitizenThumbnailModel> getCitizenFollowed(
        @RequestParam(value="id", required = true) String id){
```

```
      return citizenRelationService.getFollowed(id);
    }

  @GetMapping(value = "check/")
    public Boolean checkIfExists(
        @RequestParam(value="citizenId", required = true) String citizenId,
        @RequestParam(value="followedId", required = true) String followedId){
      return citizenRelationService.checkIfExists(citizenId, followedId);
    }
}

@RestController
@RequestMapping(value = "/vote", produces = { "application/json" })
public class VoteController extends BaseController {

  private final IVoteService voteService;

  private final IVoteValidator voteValidator;

  public VoteController(
      IVoteService voteService,
      IVoteValidator voteValidator) {
    this.voteService = voteService;
    this.voteValidator = voteValidator;
  }

  @PatchMapping
  public ResponseEntity<?> register(@Valid @RequestBody VoteModel voteModel,
   Errors errors) {

    voteValidator.validate(voteModel, errors);
    ResponseEntity<?> response = this.buildErrorMessage(errors);
    if(response == null){
      if(voteService.patchVote(voteModel)){
        return ResponseEntity
            .status(HttpStatus.CREATED)
            .body(responseMessage.setMessage("Vote changed!"));
      }
      return ResponseEntity
          .status(HttpStatus.NO_CONTENT)
          .body(responseMessage.setMessage("Vote deleted"));

    }
    return response;
  }

@GetMapping(value = "check/")
public Boolean checkIfExists(
        @RequestParam(value="citizenId", required = true) String citizenId,
        @RequestParam(value="citizenAchievementId", required = true) String
   citizenAchievementId,
        @RequestParam(value="type", required = true) String type){
    VoteModel voteModel = new VoteModel();
    voteModel.setCitizenId(UUID.fromString(citizenId));
    voteModel.setCitizenAchievementId(UUID.fromString(citizenAchievementId));
    voteModel.setType(type);
      return voteService.checkIfVoteExists(voteModel);
    }

  @GetMapping(value = "number-votes/")
    public int getNumberOfLikes(
```

```
        @RequestParam(value="citizenAchievementId", required = true) String
    citizenAchievementId,
        @RequestParam(value="type", required = true) String type){
     VoteModel voteModel = new VoteModel();
     voteModel.setCitizenAchievementId(UUID.fromString(citizenAchievementId));
     voteModel.setType(type);
       return voteService.getNumberOfVotes(voteModel);
     }
}
```

```
import { Headers, RequestOptions } from '@angular/http';
import { Observable } from "rxjs/Rx";
import { PageFilter } from "./models/index";

export class BaseService {
    protected serviceUrl: string;
    protected accessTokenUrl: string;
    protected headers: Headers;
    protected options: RequestOptions;

    constructor() {
        this.serviceUrl = "https://goodcitizen-api.herokuapp.com/";
        //  this.serviceUrl = "http://localhost:8080/";
        this.accessTokenUrl = "?access_token=";
        this.headers = new Headers({ 'Content-Type': 'application/json' });
        this.options = new RequestOptions({ headers: this.headers });
    }
    public handleError(error): Observable<any> {
        console.log(error);
        return Observable.throw(error.json());
    }

    public search(pageFilter: PageFilter): Observable<any>{
        return null;
    }

    public buildQuery(object : any): string {
        let query: string = "";
        for (var prop in object) {
            if (object.hasOwnProperty(prop)) {
                if(object[prop]){
                    if(object[prop] instanceof Array){
                        if(object[prop].length > 0){
                            query = query.concat('&' + prop + '=' + object[prop
   ].join(','));
                        }
                    }else{
                        query = query.concat('&' + prop + '=' + object[prop]);
                    }
                }
            }
```

```
        }
        return query;
    }
}
```

```
import { Injectable } from "@angular/core";
import { BaseService } from '../commons/base.service';
import { Http } from '@angular/http';
import { Observable, Subject } from 'rxjs/Rx';
import { AuthService } from '../authentication/index';
import {
    Profile,
    CitizenRelation,
    CitizenThumbnail,
    CitizenThumbnailPage } from './models/index';
import { CitizenFilter } from './filters/index';
@Injectable()
export class CitizenService extends BaseService {

    constructor(
        public http: Http,
        public authService: AuthService) {
        super();
    }

    public findById(id: string): Observable<Profile> {
        return this.http.get(
            this.serviceUrl
            + 'citizen/id/'
            + this.accessTokenUrl
            + this.authService.getAcessToken()
            + '&id=' + id)
            .map(response => response.json() as Profile)
            .catch(this.handleError);
    }

    public update(profile: Profile): Observable<number> {
        let body = JSON.stringify(profile);
        return this.http.patch(
            this.serviceUrl
            + 'citizen/update/'
            + this.accessTokenUrl
            + this.authService.getAcessToken()
            , body, this.options)
            .map(response => response.status as number);
    }

    public followUnfollow(citizenRelation: CitizenRelation): Observable<any> {
        let body = JSON.stringify(citizenRelation);
        return this.http.patch(
            this.serviceUrl
            + 'citizen-relation/'
            + this.accessTokenUrl
            + this.authService.getAcessToken()
            , body, this.options)
            .map(response => response.status as number)
            .catch(this.handleError);
    }
```

```
public getFollowers(id: string): Observable<CitizenThumbnail[]> {
    return this.http.get(
        this.serviceUrl
        + 'citizen-relation/followers/'
        + this.accessTokenUrl
        + this.authService.getAcessToken()
        + "&id=" + id)
        .map(citizens => citizens.json() as CitizenThumbnail[])
        .catch(this.handleError);
}

public getFollowed(id: string): Observable<CitizenThumbnail[]> {
    return this.http.get(
        this.serviceUrl
        + 'citizen-relation/followed/'
        + this.accessTokenUrl
        + this.authService.getAcessToken()
        + "&id=" + id)
        .map(citizens => citizens.json() as CitizenThumbnail[])
        .catch(this.handleError);
}

public getCitizenBasicInfo(id: string): Observable<CitizenThumbnail> {
    return this.http.get(
        this.serviceUrl
        + 'citizen/basic-info/'
        + this.accessTokenUrl
        + this.authService.getAcessToken()
        + "&id=" + id)
        .map(citizens => citizens.json() as CitizenThumbnail)
        .catch(this.handleError);
}

public getLoggedCitizenPicture(): Observable<string> {
    return this.http.get(
        this.serviceUrl
        + 'citizen/picture/'
        + this.accessTokenUrl
        + this.authService.getAcessToken())
        .map(response => response.json().picture as string)
        .catch(this.handleError);
}

public checkIfFollows(citizenRelation: CitizenRelation): Observable<boolean>
 {
    return this.http.get(
        this.serviceUrl
        + 'citizen-relation/check/'
        + this.accessTokenUrl
        + this.authService.getAcessToken()
        + this.buildQuery(citizenRelation))
        .map(flag => flag.json() as boolean)
        .catch(this.handleError);
}

public search(citizenFilter: CitizenFilter): Observable<CitizenThumbnailPage
> {
    return this.http
        .get(
        this.serviceUrl
```

```
            + 'citizen/'
            + this.accessTokenUrl
            + this.authService.getAcessToken()
            + this.buildQuery(citizenFilter))
            .map(citizenPage => citizenPage.json() as CitizenThumbnailPage);
    }
}
```

```
import { Component, Input, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { CitizenThumbnail, CitizenRelation } from '../../models/index';
import { CitizenService } from '../../citizen.service';
import { AuthService } from "../../../authentication/auth.service";
import { CommunityService } from "../../../community/index";

@Component({
    selector: 'citizen-thumbnail',
    templateUrl: 'citizen-thumbnail.component.html',
    styleUrls: ['citizen-thumbnail.component.scss']
})
export class CitizenThumbnailComponent implements OnInit {

    @Input()
    public citizen: CitizenThumbnail;

    @Input()
    public currentCitizenId: string;

    @Input()
    public showMakeAdminButton: boolean;

    @Input()
    public communityId: string;

    public picture: string = '';
    public defaultCitizenPicture: string = '../../assets/default-profile-pic.png
    ';
    public followButton: string = '';
    public makeAdminButtonLabel: string = '';
    public isFollowed: boolean;
    public isAdmin: boolean;

    constructor(
        private router: Router,
        private authService: AuthService,
        private citizenService: CitizenService,
        private communityService: CommunityService) {
    }

    ngOnInit(): void {
        this.checkIfFollowed();
        this.checkIfAdmin();
    }

    public setMakeAdminButton(status: boolean) {
```

```
        this.isAdmin = status;
        if (!status) {
            this.makeAdminButtonLabel = "ADD ADMIN";
        } else {
            this.makeAdminButtonLabel = "REMOVE ADMIN";
        }
    }

    public checkIfAdmin(): void {
        if (this.showMakeAdminButton) {
            this.communityService.checkIfRole(
                this.communityId, this.citizen.id, "ADMIN").subscribe(response
=> {
                if (this.citizen.id !== this.currentCitizenId) {
                    this.setMakeAdminButton(response);
                }
            });
        }
    }

    public followUnfollow(): void {
        this.citizenService
            .followUnfollow(new CitizenRelation(this.currentCitizenId, this.
citizen.id))
            .subscribe(status => this.setFollowButtonStatus(status))
    }


    public setFollowButtonStatus(status: number): void {
        if (status == 201) {
            this.isFollowed = true;
        }else{
            this.isFollowed = false;
        }
        this.setFollowButtonFlag(this.isFollowed);
    }

    public checkIfFollowed(): void {
        this.citizenService
            .checkIfFollows(new CitizenRelation(this.currentCitizenId, this.
citizen.id))
            .subscribe(flag => {
                this.isFollowed = flag;
                this.setFollowButtonFlag(flag)
            });
    }

    public setFollowButtonFlag(flag: boolean): void {
        if (flag) {
            this.followButton = "FOLLOWING";
        } else {
            this.followButton = "FOLLOW";
        }
    }

    public goToCitizenProfile(): void {
        let link = ['/citizen/', this.citizen.id];
        this.router.navigate(link);
    }

    public addRemoveAdmin() {
```

```
        this.communityService
            .addRemoveAdmin(this.citizen.id, this.communityId)
            .subscribe(response => {
                this.setMakeAdminButton(response);
            })
    }
}
```

```
import { ModuleWithProviders } from '@angular/core'
import { Routes, RouterModule } from '@angular/router'

import {
    AuthComponent,
    AnonymousGuard,
    AuthenticatedGuard,
    RecoverPasswordComponent
} from './authentication/index';

import {
    AchievementSearchComponent,
    AchievementAddComponent
} from './achievement/index';

import {
    CitizenAchievementAddComponent,
    CitizenAchievementPendingComponent,
    CitizenAchievementFriendComponent,
    CitizenAchievementRadiusComponent,
    CitizenAchievementWonComponent
} from './citizen-achievement/index';

import {
    ProfileComponent,
    ProfileCitizenFollowersComponent,
    ProfileCitizenFollowedComponent,
    CitizenSearchComponent
} from "./citizen/index";

import {
    CommunityProfileComponent,
    CommunitySearchComponent,
    CommunityAddComponent,
    CitizenProfileCommunityComponent
} from './community/index';
import { FeedbackComponent, PageHeaderComponent } from "./commons/index";

const appRoutes: Routes = [
    {
        path: '',
        component: AuthComponent,
        canActivate: [AuthenticatedGuard]
    },
    {
        path: 'recover-password',
        component: RecoverPasswordComponent,
```

```
        canActivate: [AuthenticatedGuard]
    },
    {
        path: 'achievement-search',
        component: AchievementSearchComponent,
        canActivate: [AnonymousGuard]

    },
    {
        path: 'achievement-add/:id',
        component: AchievementAddComponent,
        canActivate: [AnonymousGuard],
        data : {isEditMode : false}
    },
    {
        path: 'achievement-edit/:id',
        component: AchievementAddComponent,
        canActivate: [AnonymousGuard],
        data : {isEditMode : true}
    },
    {
        path: 'citizen-achievement-add/:id',
        component: CitizenAchievementAddComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen-achievement-pending',
        component: CitizenAchievementPendingComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen-achievement-won',
        component: CitizenAchievementWonComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen-achievement-friend',
        component: CitizenAchievementFriendComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen-achievement-radius',
        component: CitizenAchievementRadiusComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen/:id',
        component: ProfileComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen/followers/:id',
        component: ProfileCitizenFollowersComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen/followed/:id',
        component: ProfileCitizenFollowedComponent,
        canActivate: [AnonymousGuard]
    },
    {
```

```
        path: 'citizen-search',
        component: CitizenSearchComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'community-profile/:id',
        component: CommunityProfileComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'community-search',
        component: CommunitySearchComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'community',
        component: CommunityAddComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'community/:id',
        component: CommunityAddComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'citizen-profile-community/:id',
        component: CitizenProfileCommunityComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'feedback',
        component: FeedbackComponent,
        canActivate: [AnonymousGuard]
    },
    {
        path: 'header',
        component: PageHeaderComponent,
        canActivate: [AnonymousGuard]
    }
];

export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```