**Ebba Svensson** <sveb1500@student.ju.se>,  
**Pontus Anderö** <anpo16cc@student.ju.se>

A Project Work in *Internet and Cloud Services*

Jönköping University 2018

Your platform title here

*In this template, all italic text should be removed and replaced with your own text (which should not be italic); the italic text is just a placeholder letting you know what to write there.*

*On the cover page, change to your own platform title, your own names and your own JU email address.*

*You have a lot of freedom when it comes to writing this report. You do not have to use any part of this template, but the report you write should in the end somehow (in a good way) provide the same information as indicated in this template. Most students trying to do it in their own way usually fail, so if you try that, be sure to know what you are doing!*

*This page should be removed.*

Table of Contents

[Introduction 3](#_Toc528136262)

[Architecture 4](#_Toc528136263)

[Resources 6](#_Toc528136264)

[Backend Application 8](#_Toc528136265)

[REST API 9](#_Toc528136266)

# Introduction

CityForum is an application for phones, it’s a single-threaded forum where people can make a post to write about anything. The post will then pop up for all the users who live in the same city. They can then comment and like/dislike this post. This app is similar to another popular forum app that is called “Jodel”. The difference here is that in Jodel, every user is anonymous, no name and no profile. In CityForum users will have their own username and profile where they can have a picture and a textbox where they can write whatever they want for people who visits their profile to see. In this way, CityForum makes the experience more personal, users can recognize each other and make connections and that could lead to friendship in real life. The app is meant for entertainment, but can help solve real world problems too, for example can a user post a comment asking about which hairdresser is the best in the city, and people can then comment and recommend which one they think. If there’s a traffic stop somewhere, someone can make a post and warn others. It can also work as a source of news, if something interesting has happened or will happen, someone might make a post about this.

*Introduce your platform. Write text that* ***indirectly*** *answers questions like:*

* *Why does the platform exist/what is the problem it solves?*
* *How does the platform solve the problem?*
* *Why is your platform solution better than existing solutions (if any exists)?*
* *How will end users use the platform?*
* *...*

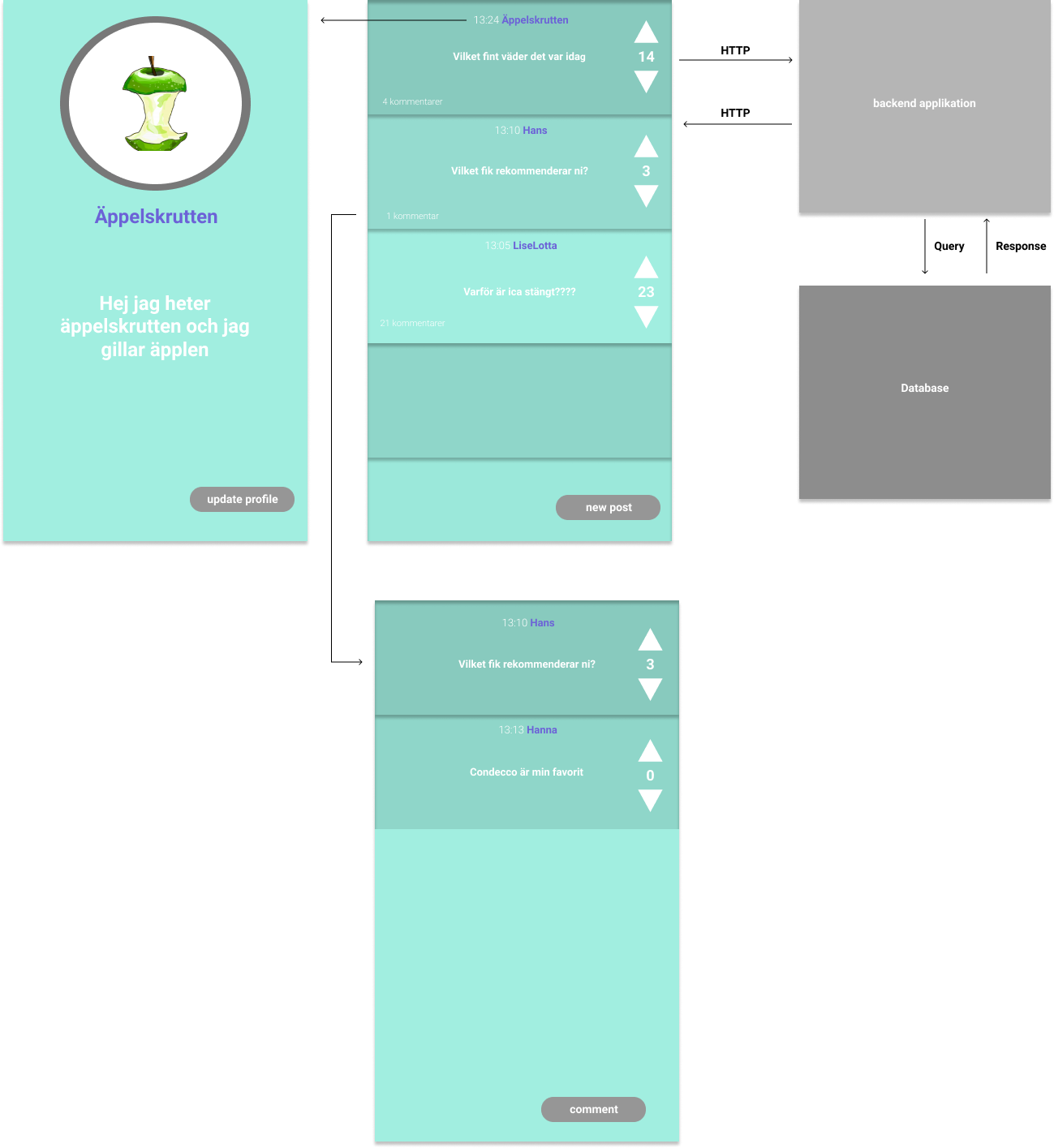
***Indirectly*** *means that you should avoid using the words* ***problem*** *and* ***solution****. For example, instead of writing "A problem with laptops is that they run on batteries which don't last forever" you can write "Batteries in laptop don't last forever, so often people can't use them as much as they want to".*

*After having read this chapter, those that have never heard of the platform before should have a good understanding of what it is. If they would like to learn how it has been implemented, they just need to continue reading the rest of the report.*

*If possibly, provide pictures/figures of some kind. Maybe a use-case diagram?*

*This chapter can to some extent be seen as a pitch text: imagine the reader is an investor, and you should convince the reader that your platform solves an existing problem in an excellent way and that it is worth investing money in it.*

# Architecture



CityForum is part of a platform that consists of three different components*.*

***The frontend application***

The blue pictures represent the frontend application; it handles the graphical interface*.*

***The backend application***

The backend application makes it possible to create or delete posts, comments, changing profile picture, liking/disliking posts and creating account/logging in. It handles HTTP requests from the frontend application and then communicates with the database to store/delete/change or fetch data to send back to the frontend application.

***The database***stores all posts, comments, account values and pictures*.*

*Give an overview of the platform. Which components does it consist of (backend application, frontend applications, frontend devices, databases, etc.)? Visualize this using a figure and show how the different parts make use of/communicate with each other.*

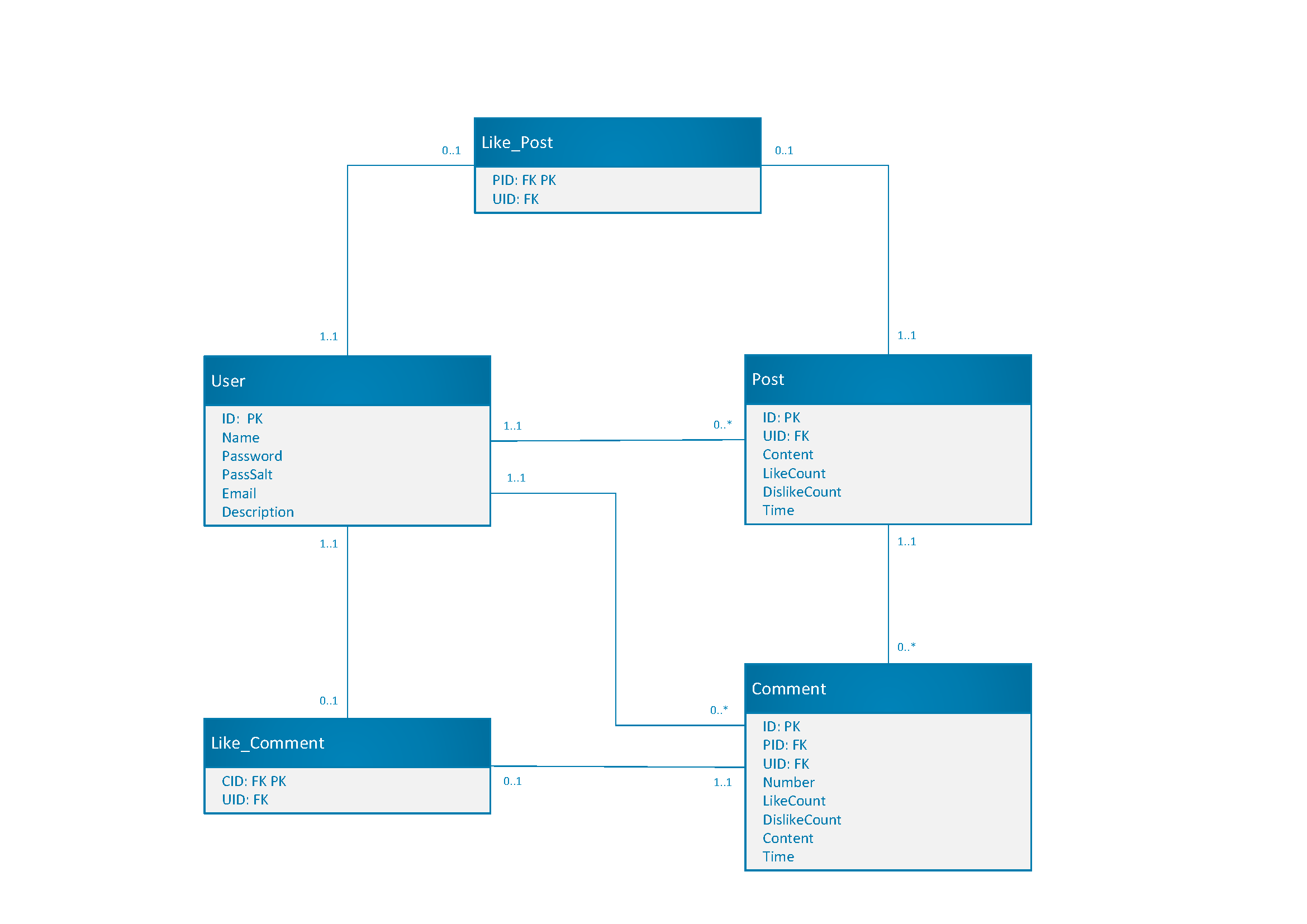
*For each component, provide a sub-chapter with more information about how that component works. You will not implement any frontend application in this course, so only give a brief description of how these works. The backend application should be described in detail in the chapter Backend Application, so only describe that one briefly here as well.*

*After having read this chapter, the reader should have a broad (but shallow) understanding of the entire platform.*

# Resources

There are four different resources on the database as User, Post, Reply, Like\_Post, and Like\_Comment.

* The ‘**User**’ entity contains everything needed to have many unique users. The attributes would be ID, Name, Password (hashed), Salt for the password, Email, Description for the profile, and the registration date.
* The ‘**Post**’ entity contains everything needed to have many unique posts. The entity has a reply count to know the amount of replies below it. The attributes would be ID, User ID, Content, Like Count, Dislike Count, and the time the post was created.
* The ‘**Comment**’ entity contains everything needed to have unique comments connected to a post. The entity need a post to be created. The attributes would be ID, Post ID, User ID, Reply Number (position in an array for example), Content, and the time the comment was created.
* The ‘**Like\_Post**’ entity is used so that a user can only like/dislike a post once. With the attributes Post ID, and User ID.
* The ‘**Like\_Comment**’ entity is used so that a user can only like/dislike a comment once. With the attributes Comment ID, and User ID.



**Figure 2** Solid line indicating relationship. FK meaning Foreign Key, and PK meaning Primary Key.

*Describe the resources in detail. What attributes do they consist of? Showing an ER diagram might be a good way to visualize the resources?*

*After having read this chapter, the reader should know how the data stored on the platform is structured. If the reader is a new programmer that should start working on the platform, she should now know what she needs to know if she wants to change the resources or add more type of resources (e.g. know how to add a new table to the database with a relation to an existing table in the database).*

# Backend Application

*Describe how you've implemented the backend application. Which language have you used? How has the code been structured? How does one start the application? Etc.*

*After having read this chapter, the reader should know how the backend application has been implemented. If the reader wants to add a new type of resource to the platform and implement CRUD operations for that one, the reader should now know precisely which files that should be created/extended to contain the new code dealing with the new resource.*

# REST API

What you should be able to do:

POSTS

* Get all posts
* Get specific post
* Create a post
* Delete a post

you should not be able to update a post

COMMENTS

* Get all comments
* Get specific comment
* Create a comment
* Delete a comment

you should not be able to update a comment

USER

* Get user content (user-name and description)
* Update user content (description and profie picture)
* Create user
* Delete user

For later:

Add profile-picture

Log in

# OPERATIONS

POSTS

Get list of all posts

|  |
| --- |
| GET /posts |

Response

If successfully fetched, response contains status 200 and an array with all the posts

|  |
| --- |
| Status: 200 OK  content-type: application/json  [{post-id: 1, user-id: 67, content: ‘here is some content’, likecount: 7, dislikecount; 9, time: 467546546}, …] |

Get specific post

|  |
| --- |
| GET /posts/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| post-id | Int | id of the post sent in the uri |

Response

If successfully fetched post, the response contains status 200 and the fetched post

|  |
| --- |
| Status: 200 OK  content-type: application/json  post-id: 123  user-id: 55  content: “some content”  likeCount: 4  dislikeCount: 3  time: 657654756757 |

Response

If post does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

Create new post

|  |
| --- |
| POST /posts |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| user-id | Int | Id of user that created the post, sent in the body |
| time | Int | Time of post creation, sent in the body |
| content | String | Content of the post, sent in the body |

Body of request

|  |
| --- |
| content-type: application/json  {  “user-id”: 123  “time”: 2342465436  “content”: “Content of post”  } |

Response

If user exist and content is not an empty string, the response contains status code 201 and the new created post.

|  |
| --- |
| Status: 201 CREATED  Location: /posts/567 |

Response

If user does not exist or content is an empty string

|  |
| --- |
| Status: 400 BAD REQUEST |

Delete specific post

|  |
| --- |
| DELETE /posts/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| post-id | Int | id of the post sent in the uri |

Response

If successfully deleted

|  |
| --- |
| Status: 204 DELETED |

Response

if post does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

COMMENTS

Get list of all comments to a specific post

|  |
| --- |
| GET /posts/123/comments |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| post-id | Int | id of the post sent in the uri |

Response

If successfully fetched, the response contains status 200 and an array with all the comments to that post

|  |
| --- |
| Status: 200 OK  content-type: application/json  [{comment-id: 43, user-id: 37, post-id: 123, content: ‘here is some content’, likecount: 7, dislikecount; 9, time: 467546546}, …] |

Get specific comment

|  |
| --- |
| GET /comments/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| comment-id | Int | Id of the comment sent in the uri |

Response

If successfully fetched the comment, response contains status 200 and the comment

|  |
| --- |
| Status: 200 OK  content-type: application/json  comment-id: 123  user-id: 55  post-id: 12  content: “some content”  likeCount: 2  dislikeCount: 10  time: 657654432467533 |

Response

If post or comment does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

Create a comment

|  |
| --- |
| POST /comments |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| user-id | Int | id of the user creating the comment, sent in the body |
| post-id | Int | id of the post the comment is made on, sent in the body |
| content | String | Content of the comment, sent in the body |

Body of request

|  |
| --- |
| content-type: application/json  {  “user-id”: 123  “post-id”: 456  “content”: “content of comment”  } |

Response

If successfully created a comment, the response contains status 201 and the new comment

|  |
| --- |
| Status: 201 CREATED  Location: /comments/123 |

Response

If post does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

Delete a comment

|  |
| --- |
| DELETE /comments/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| comment-id | Int | Id of the comment sent in the uri |

Response

If successfully deleted the comment

|  |
| --- |
| Status: 204 DELETED |

Response

If post or comment does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

USER

Get user content (name and description)

|  |
| --- |
| GET /users/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| user-id | Int | id of the user sent in the uri |

Response

If user exist, the response contains status 200 and the user name plus description

|  |
| --- |
| Status: 200 OK  content-type: application/json  name: “Sten”  description: “Hejsan” |

Response

If user does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

Update user description

|  |
| --- |
| UPDATE /users/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| user-id | Int | id of the user sent in the uri |
| Content | String | Content of description, sent in the body |

Body of request

|  |
| --- |
| content-type: application/json  {  “content”: “content of description”  } |

Response

If successfully updated user, response contains status 200

|  |
| --- |
| Status: 200 OK  content-type: application/json |

Response

If user does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

Create user

|  |
| --- |
| POST /users |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| user-name | String | Name of the user, sent in the body |
| password | String | Password in hash-form, set in the body |
| password-salt | String | Salt for the password, sent in the body |
| email | String | Email of the user, sent in the body |

Body of request

|  |
| --- |
| content-type: application/json  {  “user-name”: “Alice”  “password”: “HJEFKSF5454LKJFLS2543254”  “Password-salt”: “545hggfh67575”  “email”: “alice123@gmail.com”  } |

Response

If successfully created user, response contains status 201 and the new user

|  |
| --- |
| Status: 201 CREATED  content-type: application/json  Location: users/345 |

Response

If user-name or email is already taken

|  |
| --- |
| Status: 400 BAD REQUEST |

Delete user

|  |
| --- |
| DELETE /users/123 |

Input

|  |  |  |
| --- | --- | --- |
| **name** | **Type** | **Description** |
| user-id | Int | id of the user sent in the uri |

Response

If successfully deleted user

|  |
| --- |
| Status: 204 DELETED |

Response

If user does not exist

|  |
| --- |
| Status: 400 BAD REQUEST |

*This chapter should contain the specification for the REST API on your platform. It should contain enough information for a new programmer to start using the REST API without reading through the code on the backend application implementing the REST API.*

*If you prefer, feel free to write this chapter in a separate document, and just provide a reference to that document in this chapter.*

*You are recommended to look at how others have specified how their REST APIs work and then choose a way to describe your own REST API that you think is good. You can for example look at:*

* *Facebook:* [*https://developers.facebook.com/docs/graph-api/reference/v3.2/album*](https://developers.facebook.com/docs/graph-api/reference/v3.2/album)
* *Google Calendar:* [*https://developers.google.com/calendar/v3/reference/calendars*](https://developers.google.com/calendar/v3/reference/calendars)
* *GitHub Project:* [*https://developer.github.com/v3/projects/*](https://developer.github.com/v3/projects/)