Task 1 – Research Document

SocialHive API

By Elisea Mizzi

1. Introduction

SocialHive is a PHP and MySQL-based social media platform designed to let users connect, share content, and build communities. In order to unlock its potential, it needs an API—a way for mobile apps, third-party tools, and other websites to interact with it.

With a secure and well-designed API, SocialHive can:

- Work smoothly with mobile apps (Ionic, Flutter, React Native).
- Allow third-party integrations (AI chatbots, analytics tools, social media cross-posting).
- Provide real-time notifications and messaging across multiple devices.

2. Why Does SocialHive Need an API?

- **More Than Just a website**

Not everyone wants to be tied to a desktop browser. A mobile app or even a desktop app can greatly improve user engagement. The API makes this possible.

- **Better User Experience**

With an API, SocialHive can provide real-time updates for notifications, messages, and post interactions—just like other major social networks.

3. What Will the API Do?

The SocialHive API will cover everything a user can do on the platform—but now, they can do it from anywhere.

Key Features & Endpoints

Feature	Endpoint	Method	What It Does
User Authentication	/api/register	POST	Create a new user account
	/api/login	POST	Log in a user
	/api/logout	POST	Log out a user
User Profiles	/api/profile/{id}	GET	Get user profile info
	/api/profile/update	PUT	Update profile details
Posts & Content	/api/posts	GET	Get all posts

	/api/posts/create	POST	Create a new post
	/api/posts/{id}	GET	View a specific post
	/api/posts/{id}/delete	DELETE	Remove a post
Comments & Likes	/api/comments/{post_id}	GET	Get comments for a post
	/api/comments/create	POST	Add a comment
	/api/comments/{id}/delete	DELETE	Delete a comment
Messaging	/api/messages	GET	Fetch all user messages
	/api/messages/send	POST	Send a private message
Groups & Communities	/api/groups	GET	Get a list of groups
	/api/groups/create	POST	Create a new group
	/api/groups/{id}/members	GET	Get group members
Follow System	/api/follow/{id}	POST	Follow a user
	/api/unfollow/{id}	DELETE	Unfollow a user
Notifications	/api/notifications	GET	Get user notifications
	/api/notifications/read/{id}	PUT	Mark a notification as read

4. How Will This Benefit Users?

Scenario 1: Access via Web or Mobile App

By using the PHP API, users can interact with SocialHive from any device. Some of the key benefits include:

- User Authentication & Post Creation: Users will be able to sign in and create posts through both web and mobile applications.
- Push Notifications: The API can send push notifications for new messages, likes, and comments, so users stay updated on the latest interactions.
- Real-Time Messaging: The API allows real-time messaging, meaning users don't need to refresh the page or app to see new messages, making communication smoother and faster.

Scenario 2: Integration with Other Platforms

With the API, users can take advantage of powerful integrations with external tools and platforms:

- Cross-Posting to Other Social Media Platforms: Users can create content on SocialHive and easily share it on other platforms like Twitter or LinkedIn, expanding the reach of their posts.

- External Analytics Tools: By integrating with external analytics tools, businesses can track user engagement and interactions across the platform, helping them improve their strategies and content performance.

Scenario 3: Testing and Debugging with Postman

Using Postman, I can test and debug the API efficiently, ensuring it works properly and is ready for deployment:

- Testing API Endpoints: Postman makes it easier to test each endpoint (like creating posts, logging in, or messaging), allowing for quick identification of any bugs or issues before they affect users.
- Ensuring Smooth Integration: Postman allows me to check how well the API interacts with both the front-end app and the database, ensuring a seamless user experience across devices.

5. How Will It Be Built?

The API will follow RESTful principles, ensuring it is scalable, secure, and easy to use.

- Tech Stack

  Backend: PHP
  Database: MySQL
  Authentication: JWT (JSON Web Token)
  Security: HTTPS, OAuth2, Rate Limiting
  Tools: API, Postman, Bootstrap, GitHub

6. Security & Performance Considerations

When building the API, security is my top priority. The API will handle user logins, private messages, and personal data, so I need to ensure it's safe to use.

- JWT Authentication: This helps make sure only authorised users can access their accounts.
- Rate Limiting: Stops spamming and protects the API from being overloaded with too many requests at once.
- HTTPS Encryption: All data sent between the user and the server will be encrypted to keep everything secure.
- Input Validation & Sanitisation: This protects the API from common attacks, like SQL injection or cross-site scripting (XSS).

Performance is just as important. I want the API to be fast and responsive, even when there's a lot of data.

- Caching: This will speed up loading times by storing frequently used data, like posts and profiles.

- Pagination: Instead of loading everything at once, the API will break down large datasets into smaller, manageable chunks.
- Database Optimisation: I'll make sure the database queries are efficient, so the response time stays low.

7. Why This API is a Game-Changer

The SocialHive API isn't just a technical addition – it's the key to making the platform truly versatile and connected.

With this API, SocialHive will be able to:

- Work seamlessly with mobile apps, desktop apps, and other external tools.
- Automate social media tasks and track analytics.
- Offer features like real-time messaging, push notifications, and a smooth user experience.

Part 1 – RESTful APIs (KU1)

1. What is a RESTful API?

- A REST API is defined as an application programming interface, which follow the design principles of the REST architectural framework. REST in itself is defined as a representational state transfer; these are a set of rules and guidelines that should be followed when building an API. A collection of guidelines and procedures for creating and integrating application software is known as an API. Sometimes also referred to as a contract between an information provider and an information user, making it a point to establish the content required from both the consumer and the producer. (Red Hat, 2020)

RESTful APIs enable communication between clients (like web and mobile apps) and servers using **HTTP**. They allow clients to send requests (GET, POST, etc.) and receive structured responses, making data exchange smooth and efficient.

2. Core Principles of REST

- **Client-Server Architecture** – A client-server architecture in which HTTP is used to manage requests between clients, servers, and resources.
- **Statelessness** – Stateless client-server communication means that each request is independent and unrelated to the others, and no client data is saved in between get requests.
- **Cacheability** – Cacheable information that simplifies communication between clients and servers.

- **Layered System** – By making use of layered systems, organizers are able to organize each type of server that is used to retrieve any requested information into hierarchies, this would be invisible to the client.
- **Uniform Interface** – Clients can interpret self-descriptive messages, manage resources, and navigate activities through hypermedia thanks to a standardised interface that standardises data transport.
- **Code on Demand (optional)** – This gives APIs the ability to send executable code to the client when requested, allowing clients to have extended functionality.

REST follows a set of guidelines that can be used as needed, allowing for the REST APIs to be faster and more lightweight, with increased scalability which is ideal for internet of things (IoT) as well as mobile development. (Red Hat, 2020)

3. How API Requests Work

- R**equest/response process**:
    1. An HTTP request is sent by a client (such as a web application or mobile app) when it requests data from an API.
    2. In order to perform this request, the API sometimes communicates with a database to retrieve or update data.
    3. After that, the API provides the client with the data it requires by returning a response, typically in XML or JSON format.
- An example using **Postman**.
    1. Open Postman and start a new request.
    2. Choose **GET** as the method and enter https://api.example.com/users/1 as the URL.
    3. Hit **Send**, and you'll see the response data appear in JSON format.

4. HTTP Methods

For an API request to go through a request method is needed. This is also known as the HTTP method, which focuses on the specified type of action to be performed based off one of the following functions.

- GET – The **GET** method is used to request and retrieve data from the server. It does not modify any existing data; it simply fetches information. This is commonly used for:
    - Fetching user profiles
    - Viewing posts or comments
    - Searching for specific data
- POST – The **POST** method is used to send data to the server to create a new resource. Unlike GET, this method **modifies** the server by adding new data. **Common use cases:**
    - Creating a new user account
    - Posting a new status update
    - Uploading an image

- PUT – The **PUT** method is used to update an **entire** existing resource. If a resource exists, it will be replaced; if it doesn't, the server may create it (depending on implementation). **Common use cases:**
    - Updating a user's profile information
    - Replacing an entire post with new content
- PATCH – The **PATCH** method is used when only part of a resource needs to be updated, rather than replacing the entire object like **PUT** does. **Common use cases:**
    - Changing only a user's profile picture
    - Editing a single field in a post (e.g., correcting a typo)
- DELETE – The **DELETE** method is used to remove a resource from the server. Once deleted, the resource is no longer available. **Common use cases:**
    - Deleting a user's post
    - Removing an account
    - Unliking a post

Each HTTP request is sent to a specific **endpoint**, which is a **unique URL** that serves as the access point for a resource. Endpoints define what data can be accessed and what actions can be performed.

For example, in SocialHive:

- GET /api/users → Retrieves all users
- GET /api/users/123 → Retrieves user with ID **123**
- POST /api/posts → Creates a new post
- DELETE /api/posts/456 → Deletes post with ID **456**
- **How an API Request Works (Step-by-Step)**

A client (browser, mobile app, or external system) sends an HTTP request to an API endpoint with the appropriate method.

The server processes the request, checking permissions, validating data, and executing the requested operation.

A response is returned to the client, typically in JSON format, containing the requested data or confirmation of the action performed.

(sendbird, 2025).

5. Query String Parameters vs. Request Body

- **Query Parameters** – Query Strings are a set of characters that are placed at the end of a URL, positioned after the question mark which may also have one or more parameters. These parameters are represented by a unique key-value pairs or a set of two linked data items for example:

  https://www.tech.com/search?query=database+tools&star_rating=4&order=alphabetical

Query strings are important as they pass on information to the web server, informing it of what content needs to be passed on or if any action needs to be taken. Allowing the page content to be modified according to what the user is searching for giving them a personalized experience. (claravine, n.d.)

The Request Body is used in POST, PUT, and PATCH requests to send structured data (usually in JSON format) to the server. It allows the client to create (POST), update (PUT), or modify (PATCH) data.

Part 2 – Authorization (KU2)

1. What is OAuth 2.0?

- **OAuth 2.0** stands for "Open Authorization", is a standard created to let an application or website access resources hosted by other web apps on a user's behalf. Without ever exposing the user's login information, OAuth 2.0 gives permissioned access and limits the actions that the client app can take on the user's behalf (Okta Inc, 2025).
- **Principles**
    - OAuth 2.0 is an authorization protocol meaning that it is designed as a mean of granting access for example remote APIs or user data (Okta Inc, 2025).
- **Benefits**:
    - OAuth 2.0 version makes use of **Access Tokens**, these are pieces of data that are a representation of the authorization required to access resources on behalf of the end user. It is also important to note that for security reasons these Access Tokens may have an expiration date (Okta Inc, 2025).
    - **Scopes** are also an important concept that are used as they specify the exact reason why the resource may be granted (Okta Inc, 2025).

2. OAuth 2.0 Components

- **Scopes** – Scopes specify what the customers are authorized to access in the user's name. On the other hand, permissions such as RBAC and ACL define what the user is allowed to do. Scopes allows users to control the level of access that their clients must protected resources. Permissions make sure a user can only access their own resources, not those of other users (Ory Corp, 2025).
- **Access Tokens-** An **access token** is a temporary credential that allows users to securely access an API without repeatedly logging in. Instead of sending login details with every request, the system issues a token that verifies the user's identity.(Ory Corp, 2025).

  Security Best Practices
    - **Use HTTPS** to encrypt token transmission.
    - **Set Expiry Times** to reduce security risks.
    - **Store Securely** (e.g., HTTP-only cookies).

- **Implement Revocation** to allow users to log out and revoke tokens.

Access tokens ensure **secure, efficient authentication**, making SocialHive's API safer and more user-friendly.

- **Client ID and Client Secret** – After users register their application, they would receive a Client ID and optionally a client secret from the authorization server. These identifiers allow that the authorization server identifies your application and allow it to access protected resources (Ory Corp, 2025).

3. How OAuth 2.0 Works

- Explain the **Authorization Flow**:
    1. The user requests access.
    2. The client redirects the user to the authorization server.
    3. The user logs in and grants permissions.
    4. The authorization server sends an **access token**.
    5. The client uses this token to access protected resources.

Implementation Plan

- Since this is my first time working with APIs, I want to keep things simple but effective. One of the most important things I need to implement is user authentication, and rather than making users create yet another password, I've decided to integrate **Google OAuth 2.0** for login. This will allow users to sign in using their Google accounts, making the process smoother and more secure.

  Adding Google OAuth 2.0 will make SocialHive more secure, more user-friendly, and easier to manage. Plus, it saves me from handling password security, which is one less thing to worry about! Since this is my first time working with an API,

<u>Part 3 – Security Considerations (KU3)</u>

1. **What is the OWASP API Security Top 10?**
    - **Broken Object Level Authorization**- When APIs don't properly check who's allowed to access certain data, attackers can sneak in and steal or change information they shouldn't have access to. (OWASP API, 2023)
    - **Broken Authentication**- Weak or poorly implemented login systems make it easy for attackers to break in, steal session data, or impersonate other users (OWASP API, 2023).
    - **Broken Object Property Level Authorization**- Even if a user is allowed to access an object, they shouldn't necessarily see **everything** in it. When APIs fail to limit access at a deeper level, users might end up seeing or modifying data they shouldn't (OWASP API, 2023).
    - **Unrestricted Resource Consumption**- Some APIs don't limit how much data or processing power a user can request, which means attackers can

flood the system and slow it down—or even crash it completely (OWASP API, 2023).

- **Broken Function Level Authorization**- APIs often have different levels of access for different users, but when these rules aren't properly enforced, people can end up accessing functions they shouldn't (OWASP API, 2023).
- **Unrestricted access to sensitive Business Flows**- If an API exposes critical business logic without proper security checks, attackers can exploit it to manipulate transactions, access financial data, or disrupt important workflows (OWASP API, 2023).
- **Server Side Request Forgery**- If an API blindly fetches data from URLs provided by users, attackers can trick it into accessing internal servers or sensitive information (OWASP API, 2023).
- **Security Misconfiguration**- Poorly configured security settings (like leaving debug mode on or using default passwords) can leave APIs wide open to attacks (OWASP API, 2023).
- **Improper Inventory Management**- If developers don't properly track different API versions and endpoints, outdated or insecure versions may remain accessible, creating security holes (OWASP API, 2023).
- **Unsafe Consumption of APIs** - Relying on third-party APIs without verifying the security of the data they provide can introduce vulnerabilities, especially if those APIs are compromised (OWASP API, 2023).

2. **Overview of API Security Threats**
   APIs are constantly under attack, and these are some of the most common ways hackers try to exploit them:

   - **Injection attacks-** Malicious code sneaks into API inputs, potentially leading to data leaks, database manipulation, or system takeovers (Akamai Technologies, 2025).
   - **Broken Authentication and session management-** If an API doesn't properly handle user logins, passwords, and tokens, attackers can hijack accounts and access private data (Akamai Technologies, 2025).
   - **Insecure communication-** If data isn't encrypted while being transmitted, hackers can intercept sensitive information like passwords or payment details (Akamai Technologies, 2025).
   - **DDoS attacks-** Without rate limits, attackers can flood an API with requests, overwhelming servers and shutting down services (Akamai Technologies, 2025).
   - **Misuse of API keys-** If API keys are exposed or not properly secured, attackers can use them to gain unauthorized access to systems and data (Akamai Technologies, 2025).
   - **Lack of Input Validation-** If an API blindly trusts user inputs, it opens the door to all kinds of attacks, from SQL injection to remote code execution (Akamai Technologies, 2025).
   - **Unvalidated redirects and forwards**- If APIs allow users to be redirected without checking the destination, attackers can send people to phishing or malware sites (Akamai Technologies, 2025).

- **Lack of access control-** If there aren't strong rules in place about who can do what, unauthorized users might be able to access or change sensitive data (Akamai Technologies, 2025).
- **Lack of monitoring and logging-**Without proper tracking of API activity, attacks can go unnoticed until it's too late (Akamai Technologies, 2025).

3. Select Two Threats & Mitigation Strategies

1. Broken Authentication

- **What is the threat?**

This is when my API doesn't properly check if a user is who they say they are. If authentication (like a password check) isn't done right, someone could trick the system into giving them access to private data.

- **How I will fix it:**

- **Use OAuth 2.0**: I plan to use OAuth 2.0, which is a secure way to check if a user is logged in, without them needing to enter a password every time. It's like using Google to check if someone's identity is verified.
- **Require strong passwords**: I'll make sure users create passwords that are hard to guess by requiring a mix of letters, numbers, and symbols.
- **Enable Multi-Factor Authentication (MFA)**: I will add an extra layer of security. After the user logs in with a password, they'll also have to verify their identity with another method, like a code sent to their phone.

2. SQL Injection

- **What is the threat?**

SQL injection happens when someone sends harmful code to my API to mess with or steal data from my database. It's a serious risk, but I can prevent it by taking a few simple steps.

- **How I will fix it:**

- **Use Prepared Statements**: I'll make sure that when my API talks to the database, it uses prepared statements. This will ensure that user input is treated as just data and not part of the code, preventing harmful commands from being executed in the database.
- **Validate user input**: I'll always check that the information coming into my API (like names or emails) is exactly what I expect. This way, if someone tries to input something harmful, it will get blocked.
- **Use ORM (Object-Relational Mapping)**: I'll use an ORM, which will handle most of the database communication for me. This tool will help make sure that user inputs are safe and won't cause problems in my database.

References

Akamai Technologies. (2025). *What Are API Security Risks?* Retrieved from Akamai: https://www.akamai.com/glossary/what-are-api-security-risks

claravine. (n.d.). *A Query on Using Query Strings/Parameters*. Retrieved from claravine: https://www.claravine.com/a-query-on-using-query-strings-parameters/#:~:text=A%20query%20string%20is%20a,separates%20each%20key%20and%20value.

Okta Inc. (2025). *What is OAuth 2.0?* Retrieved from auth0: https://auth0.com/intro-to-iam/what-is-oauth-2

Ory Corp. (2025). *oauth2-concept*. Retrieved from ory: https://www.ory.sh/docs/oauth2-oidc/overview/oauth2-concepts

OWASP API. (2023). *OWASP Top 10 API Security Risks*. Retrieved from OWASP: https://owasp.org/API-Security/editions/2023/en/0x11-t10/

Red Hat. (2020, May 8). *What is a REST API?* Retrieved from Red Hat: https://www.redhat.com/en/topics/api/what-is-a-rest-api

sendbird. (2025). *What is an API request?* Retrieved from sendbird: https://sendbird.com/learn/what-is-an-api-request