API ABUSE

YASH PATEL (2310624)

ARMIN KHALEGI RAD (2310539)

RAKSHIT SAIN (2310453)

Outline

- Introduction
- What is API Abuse ?
- Impact
- Common Vulnerabilities Exploited
- Motivations Behind API Abuse
- Prevention Strategies
- Implementing API Security
- Demos (Attack & Prevention)
- Technology use
- Q&A

Introduction

- In our digital world, software talks to each other through something called APIs. These APIs help apps and programs work together smoothly.
- But sometimes, bad people try to take advantage of these APIs. They might try to steal information, crash systems, or cause trouble.
- APIs are like the doors to your digital house. If they're not secure, the bad guys can break in. So, it's really important to keep them safe.
- Today, we're going to learn about API abuse, what harm it can cause, and some simple steps we can take to stop the bad guys in their tracks.
- By the end, you'll know how to protect your digital 'house' and keep your data and systems safe from API troublemakers.

What is API Abuse?

- API abuse is when someone uses an application's interface in a harmful or unauthorized way. It's like a person trying to pick a lock to break into a house.
- People who abuse APIs might try to steal information, crash systems, or do things they're not supposed to.
- API abuse can lead to data leaks, system crashes, and even financial losses.
- Just like you lock your doors to keep your home safe, developers need to secure APIs to prevent abuse and protect data.

Type of API Abuse.

- Unauthorized Access: Attackers try to get into systems they shouldn't be in. It's like someone sneaking into a building without a key.
- Denial of Service (DoS): Imagine a traffic jam that stops everyone from moving. In the digital world, attackers flood a system, making it crash and unavailable.
- Data Scraping: This is like someone secretly collecting your personal information. Attackers use automated tools to steal data from systems.
- Brute Force Attacks: Just like trying all possible keys until one fits, hackers use this to guess passwords or codes.
- API Rate Limiting Violations: Imagine repeatedly ringing a doorbell to annoy. Attackers exceed allowed limits, disrupting services.
- API Token/Key Theft: This is like someone stealing your house keys. Hackers take API tokens or keys to pretend they're authorized.

Impacts

- Data Breaches: API abuse can lead to unauthorized access to sensitive information, resulting in data leaks. It's like someone breaking into a vault and stealing valuables.
- System Downtime: When attackers overload systems, they can crash services. This disrupts operations and frustrates users, just like a power outage in a neighborhood.
- Financial Losses: API abuse can cause companies to lose money through downtime, stolen data, and damage to reputation. It's like a shop losing customers due to bad service.
- Reputation Damage: If customers lose trust in a company's security, they might take their business elsewhere. It's like a house losing its value due to safety concerns in the neighborhood.
- Legal Consequences: Companies can face legal actions if user data is compromised. It's like being sued for not securing your house properly.
- Overall Chaos: API abuse doesn't just affect companies—it can disrupt entire online ecosystems. It's like one house in a row causing problems for the whole neighborhood.
- Mitigation: By understanding the potential impact, we can work to prevent API abuse and protect both data and reputation.

Common Vulnerabilities Exploited

- Lack of Input Validation: Imagine accepting any package without checking its contents. This vulnerability allows attackers to inject harmful code, leading to breaches.
- Insecure Data Transmission: Sending data without protection is like shouting sensitive information in public. Hackers can intercept and steal data during transmission.
- Insufficient Rate Limiting: Not limiting requests is like allowing someone to ring your doorbell non-stop. Attackers overload systems by sending too many requests.
- Insecure Storage of API Keys/Tokens: Storing keys where anyone can find them is like leaving your house keys in the open. Attackers steal keys to gain unauthorized access.
- Broken Authentication Flow: Think of this as someone tricking you into opening the door for them. Attackers exploit flaws in the login process.
- Zero-Day Vulnerabilities: These are like secret passages only attackers know about. They use undiscovered weaknesses to breach systems.
- Mitigation and Best Practices: Understanding these vulnerabilities helps us reinforce our APIs. Implement strong authentication, validate inputs, encrypt data, and secure keys.

Prevention Strategies for API Abuse

- Robust Authentication and Authorization:
  - Use strong locks for your digital doors. Implement methods like OAuth or JWT for secure access.
  - Assign specific access levels to users through Role-based access control (RBAC).
- Input Validation and Sanitization:
  - Inspect and clean what's coming in. Validate input data to prevent attacks like SQL injection and cross-site scripting (XSS).
- Secure Data Transmission:
  - Encrypt data like sending it in a locked box. Use HTTPS (TLS/SSL) to safeguard data during transit.
- Effective Rate Limiting and Throttling:
  - Prevent overcrowding at the door. Set sensible limits on how often requests can be made.
- CAPTCHA or reCAPTCHA Implementation:
  - Stop bots from knocking. Use CAPTCHA challenges to ensure requests are made by humans.
- Regular Security Audits and Penetration Testing:
  - Periodically check your locks. Conduct thorough tests to find and fix vulnerabilities.

Prevention Strategies for API Abuse

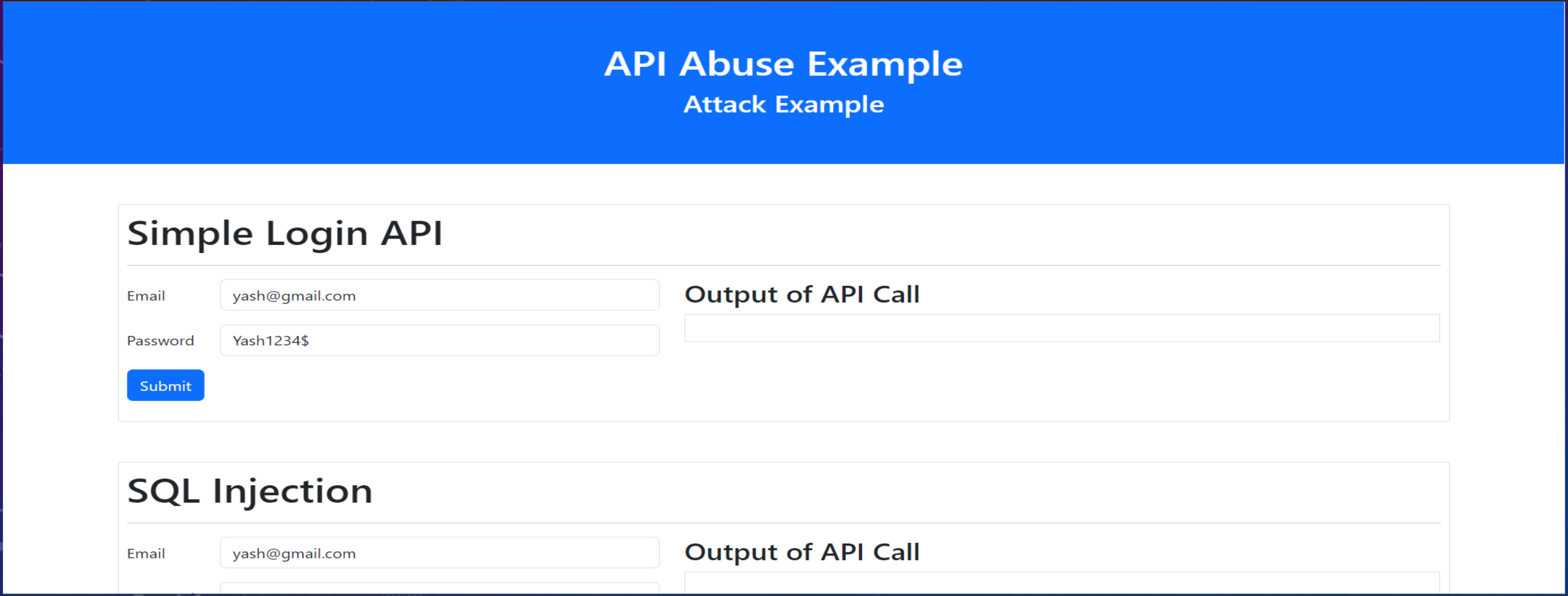
- API Security Tools:
  - Deploy guardians at your entrance. Tools like API Gateways, Web Application Firewalls (WAFs), and Intrusion Detection Systems (IDS) help protect APIs. Regular Updates and Patches:
  - Keep your locks up-to-date. Regularly update your APIs and associated software to fix security vulnerabilities.
- User Education:
  - Teach users about security. Educated users can help prevent breaches through responsible API usage.
- Real-time Monitoring and Logging:
  - Keep watch on who's knocking. Monitor traffic and maintain logs to detect and respond to suspicious activity.
- Collaboration with Security Experts:
  - Consult with the locksmiths. Work with security professionals to identify and implement best practices.

Implementing API Security

- API Gateway: Centralized Control:
  - Think of it as a security checkpoint. An API gateway manages access, enforces security policies, and filters traffic.
- Web Application Firewall (WAF):
  - Like a protective shield. A WAF blocks malicious traffic and filters out threats before they reach your APIs.
- Monitoring and Logging:
  - Digital security cameras. Monitor API traffic and keep detailed logs to track and analyze activity.
- Real-time Traffic Analysis:
  - Watching for unusual behavior. Analyze incoming and outgoing traffic patterns to identify anomalies.
- Encryption: Protecting Data in Transit and Storage:
  - Securing your data with a lock. Encrypt data as it moves between systems and when stored.
- Regular Security Audits and Penetration Testing:
  - Scheduled security checks. Test your APIs for vulnerabilities and fix them promptly.

What We have Implement?

Step 1 : Created a Simple Attack page from where we can test the attacks



API Abuse Example

Attack Example

Simple Login API

Email: yash@gmail.com

Password: Yash1234$

Output of API Call

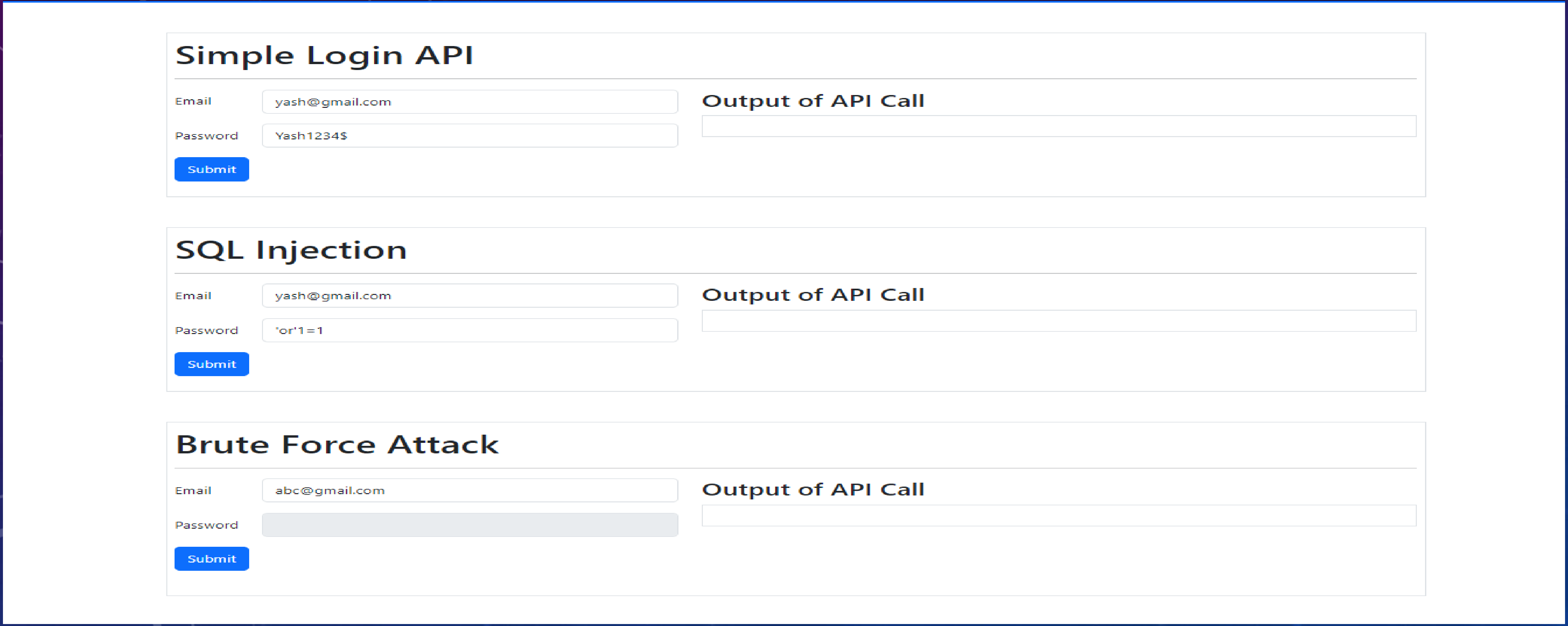
SQL Injection

Email: yash@gmail.com

Output of API Call

Step 2 : Added 3 Forms and created a Login API.

1. Simple Login API
2. SQL Injection
3. Brute force Attack



Simple Login API

Email	yash@gmail.com
Password	Yash1234$

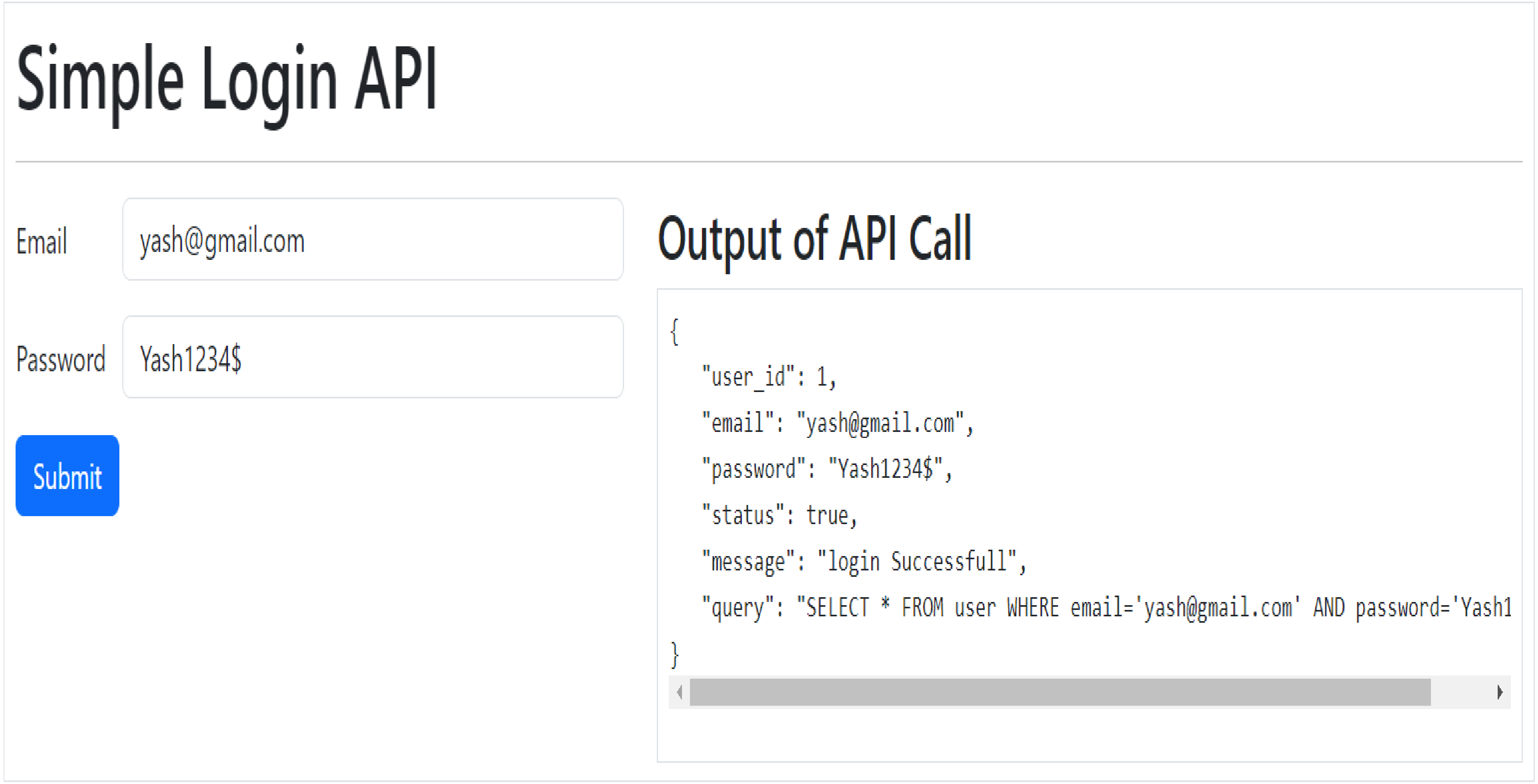
SQL Injection

Email	yash@gmail.com
Password	'or'1=1

Brute Force Attack

Email	abc@gmail.com
Password	

1. Simple Login API



Simple Login API

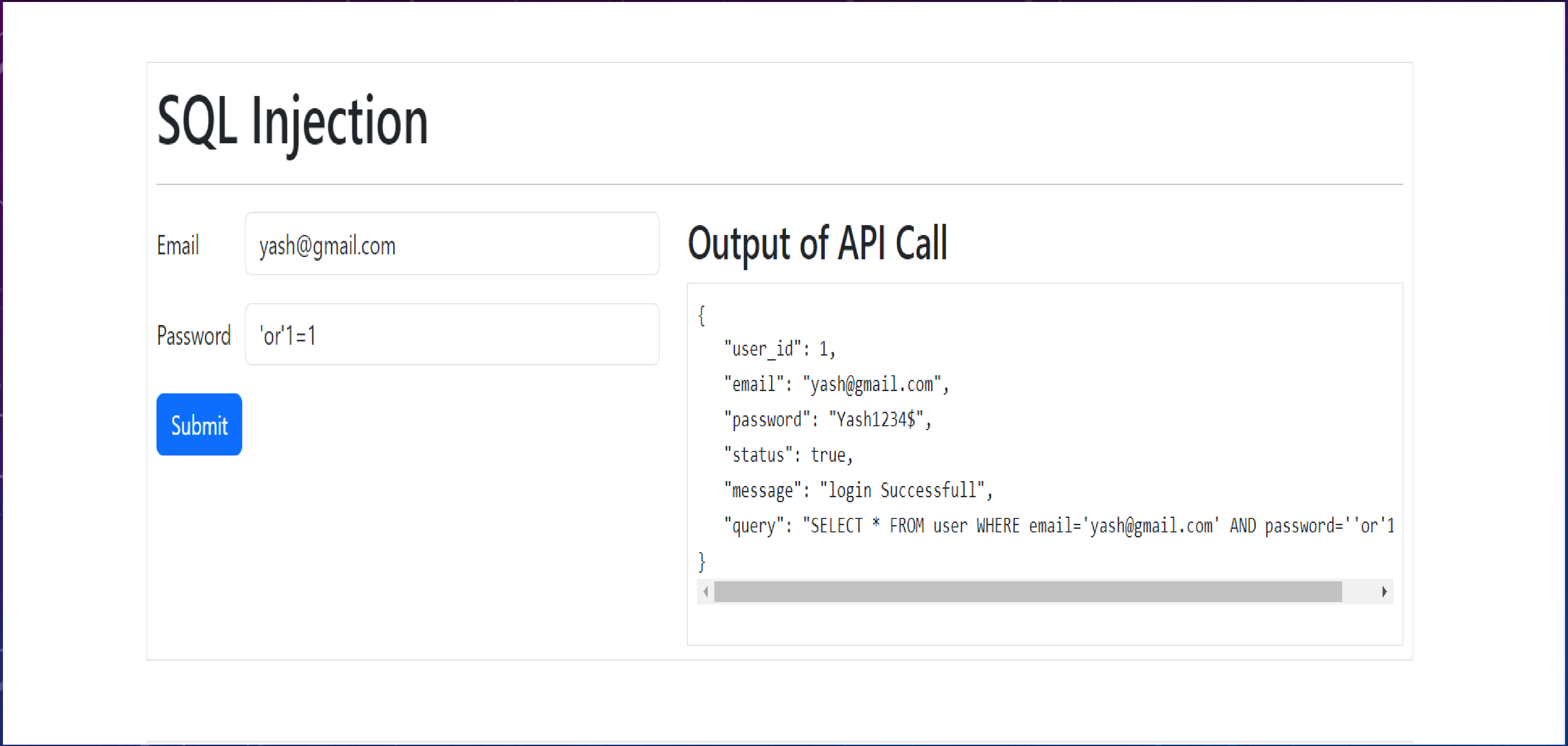
Email yash@gmail.com

Password Yash1234$

Output of API Call

```
{
   "user_id": 1,
   "email": "yash@gmail.com",
   "password": "Yash1234$",
   "status": true,
   "message": "login Successfull",
   "query": "SELECT * FROM user WHERE email='yash@gmail.com' AND password='Yash1
}
```

2. SQL Injection



SQL Injection

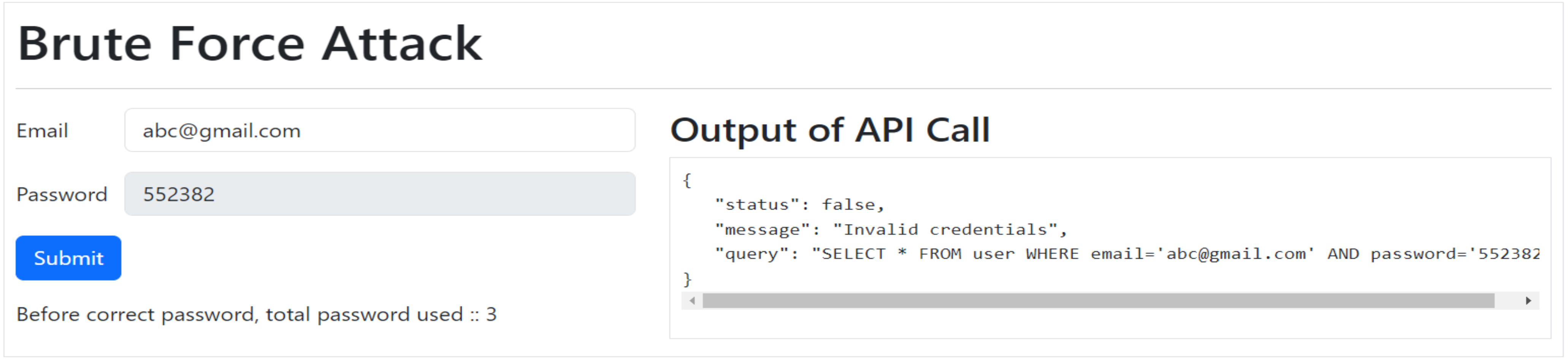
Email yash@gmail.com

Password 'or'1=1

Output of API Call

```
{
    "user_id": 1,
    "email": "yash@gmail.com",
    "password": "Yash1234$",
    "status": true,
    "message": "login Successfull",
    "query": "SELECT * FROM user WHERE email='yash@gmail.com' AND password=''or'1
}
```

3. Brute Force Attack



Brute Force Attack

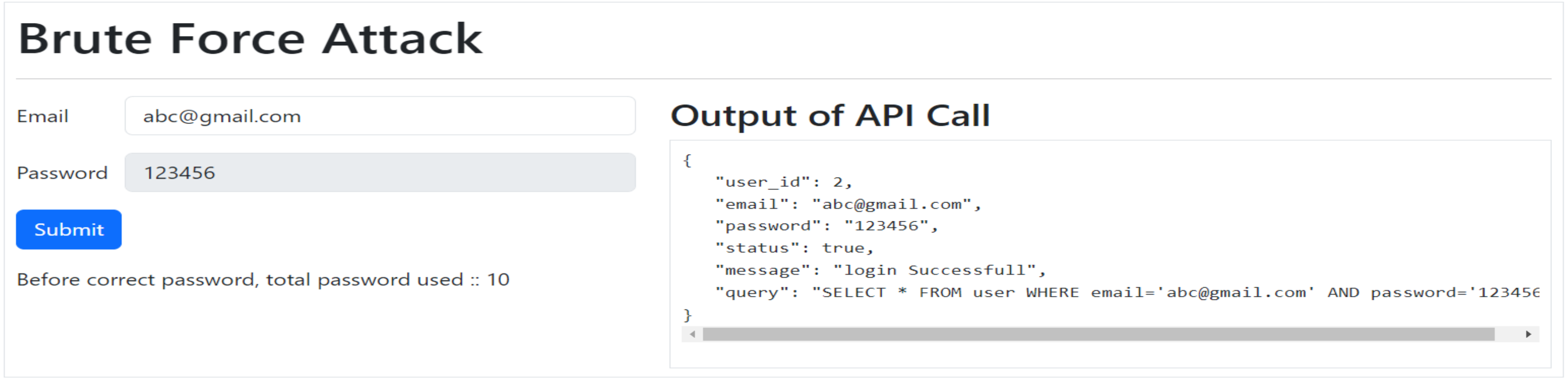
Email abc@gmail.com

Password 552382

Before correct password, total password used :: 3

Output of API Call

```
{
    "status": false,
    "message": "Invalid credentials",
    "query": "SELECT * FROM user WHERE email='abc@gmail.com' AND password='552382
}
```



Brute Force Attack

Email abc@gmail.com

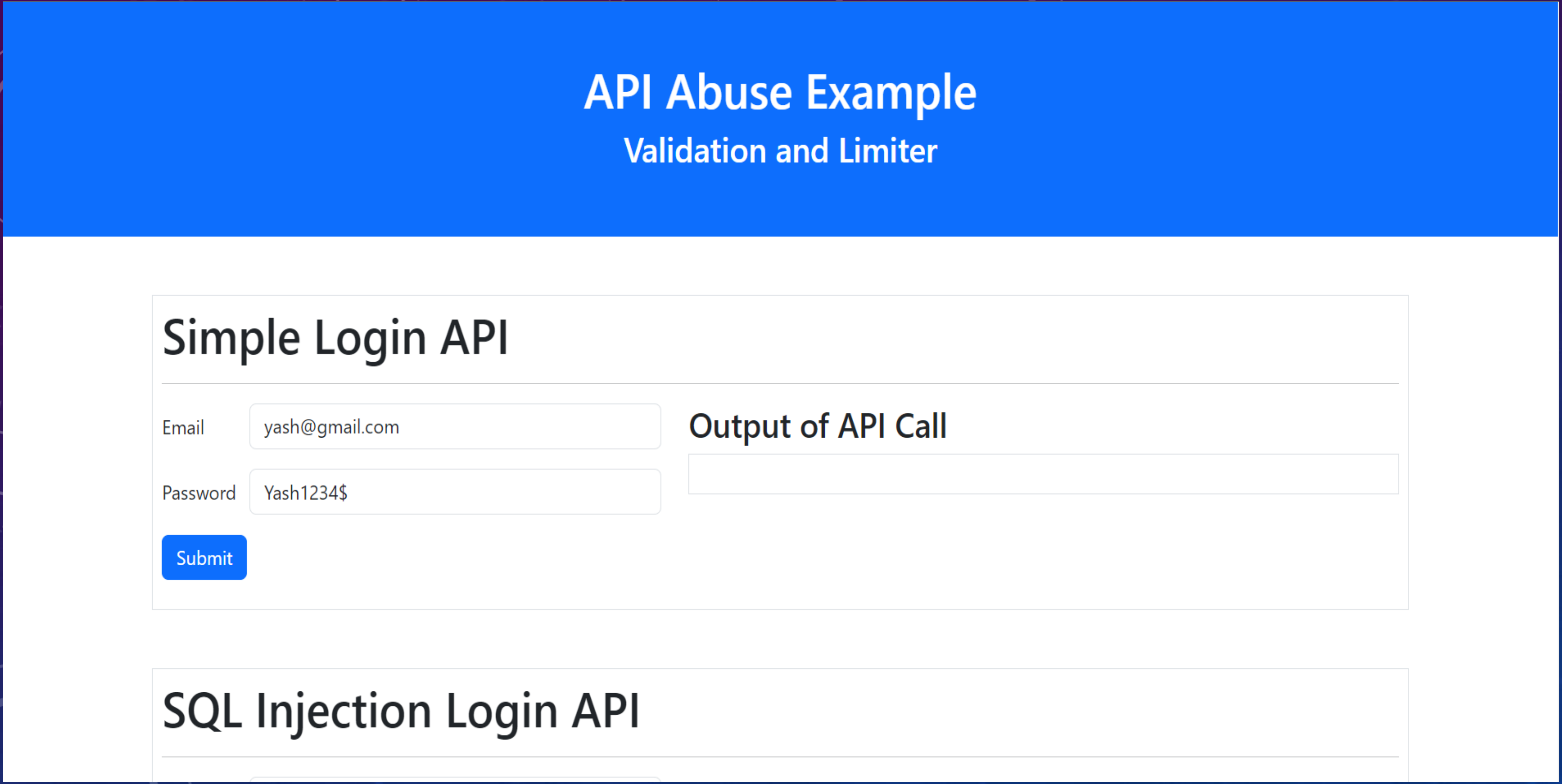
Password 123456

Before correct password, total password used :: 10

Output of API Call

```
{
    "user_id": 2,
    "email": "abc@gmail.com",
    "password": "123456",
    "status": true,
    "message": "login Successfull",
    "query": "SELECT * FROM user WHERE email='abc@gmail.com' AND password='123456
}
```

Step 3 : Created another page that prevent the attacks:



API Abuse Example

Validation and Limiter

Simple Login API

Email	yash@gmail.com
Password	Yash1234$

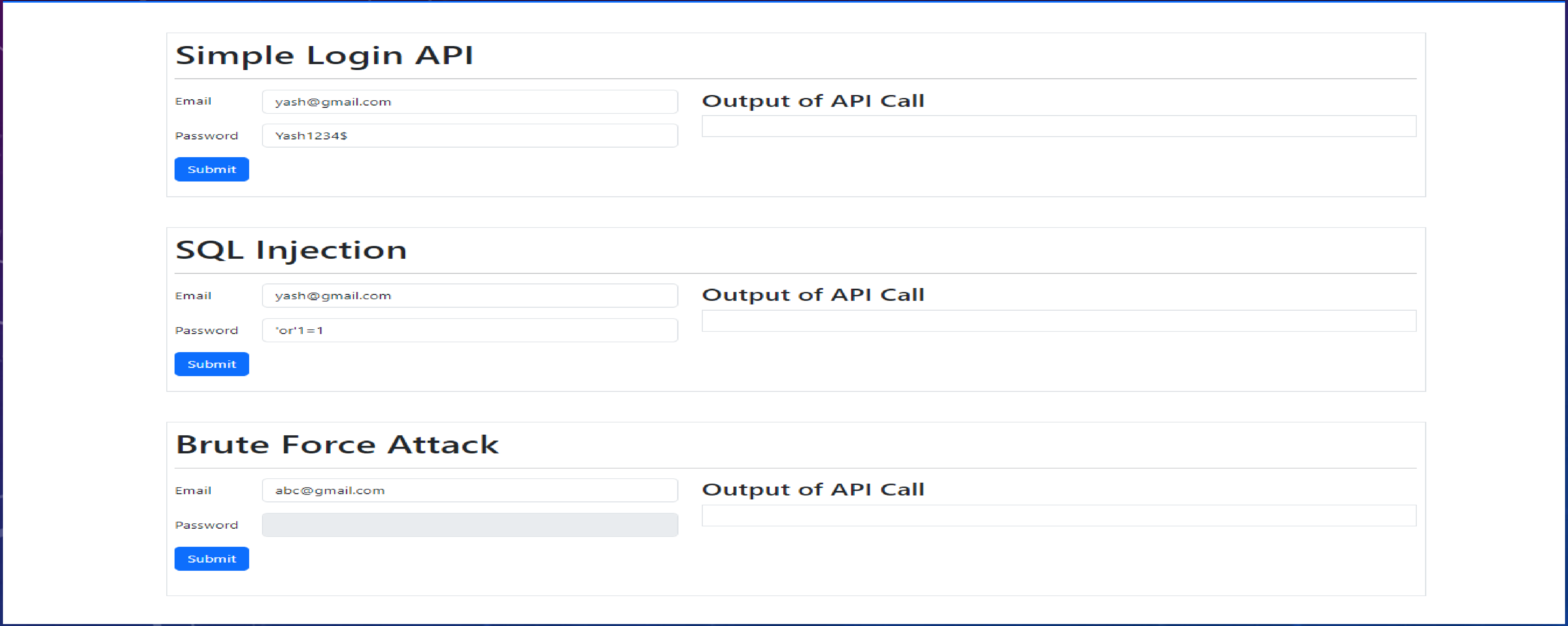
Submit

Output of API Call

SQL Injection Login API

Step 2 : Added 3 Forms with validators and A login API with rate limiter.

1. Simple Login API
2. SQL Injection
3. Brute force Attack



Simple Login API

Email	yash@gmail.com
Password	Yash1234$

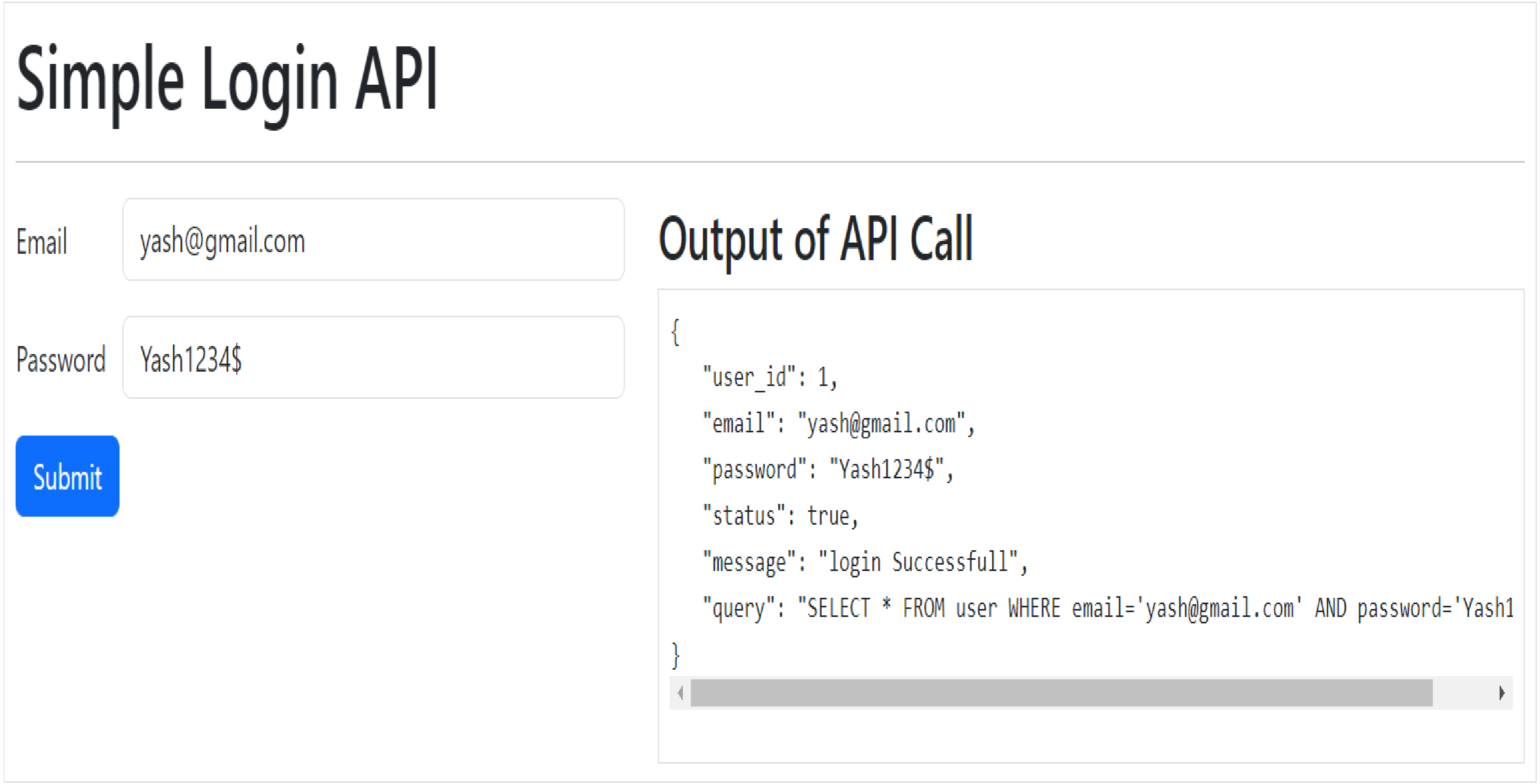
SQL Injection

Email	yash@gmail.com
Password	'or'1=1

Brute Force Attack

Email	abc@gmail.com
Password	

1. Simple Login API



Simple Login API

Email yash@gmail.com

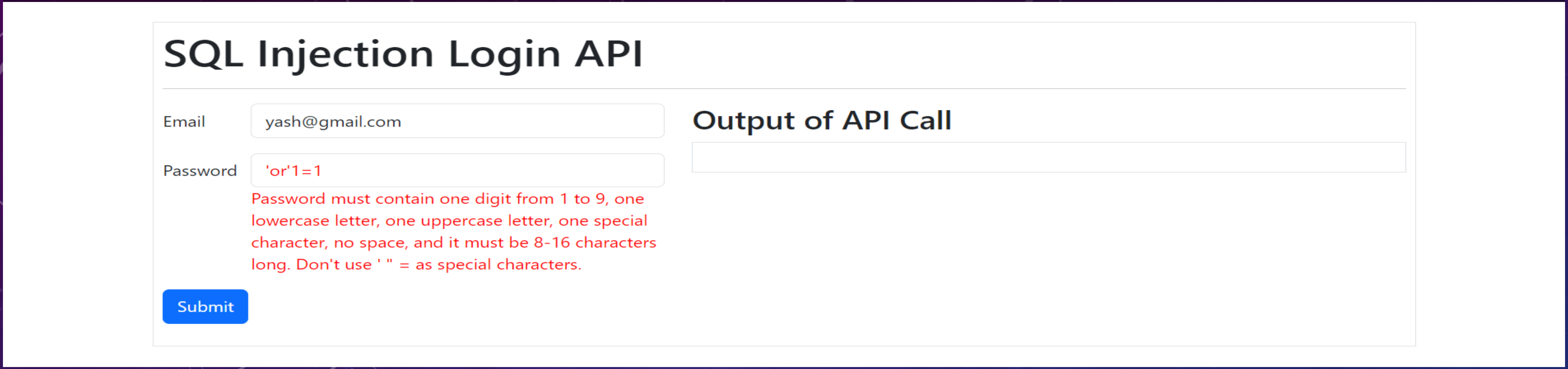
Password Yash1234$

Submit

Output of API Call

```
{
   "user_id": 1,
   "email": "yash@gmail.com",
   "password": "Yash1234$",
   "status": true,
   "message": "login Successfull",
   "query": "SELECT * FROM user WHERE email='yash@gmail.com' AND password='Yash1
}
```

2. SQL Injection Form with validators



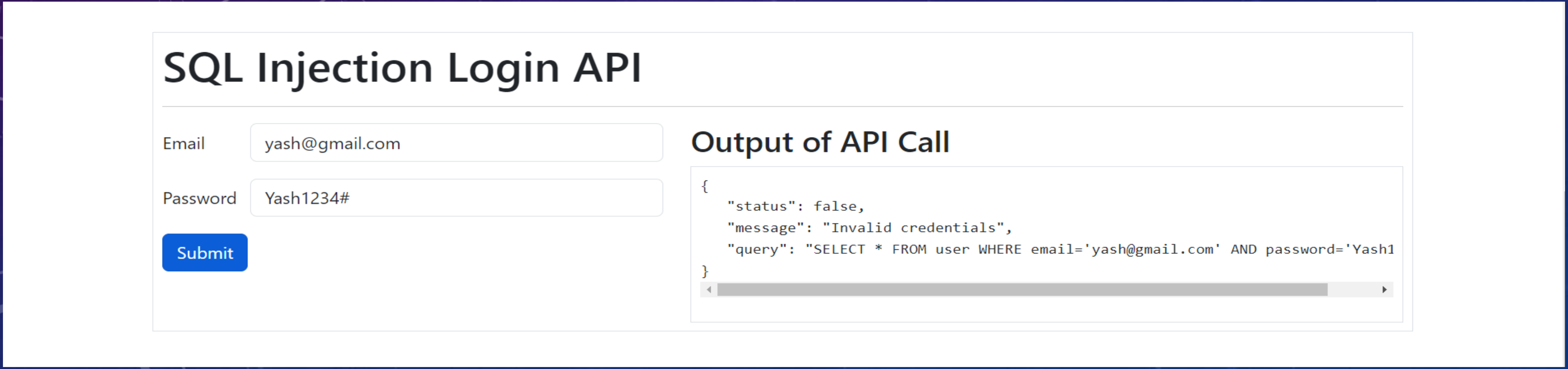
SQL Injection Login API

Email yash@gmail.com

Password 'or'1=1

Password must contain one digit from 1 to 9, one lowercase letter, one uppercase letter, one special character, no space, and it must be 8-16 characters long. Don't use ' " = as special characters.

Output of API Call



SQL Injection Login API

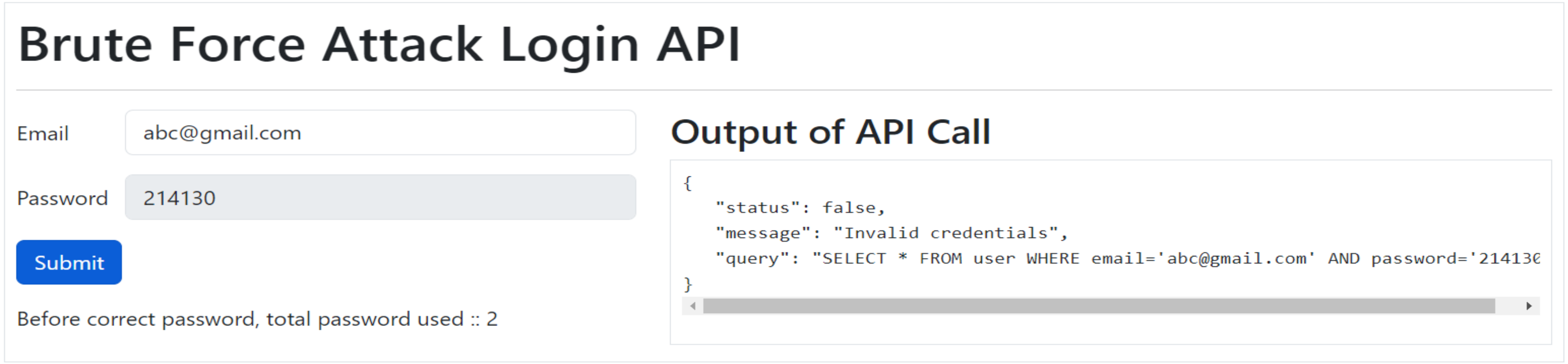
Email yash@gmail.com

Password Yash1234#

Output of API Call

```
{
   "status": false,
   "message": "Invalid credentials",
   "query": "SELECT * FROM user WHERE email='yash@gmail.com' AND password='Yash1
}
```

3. Brute Force Attack Form with Rate Limiter in backend:



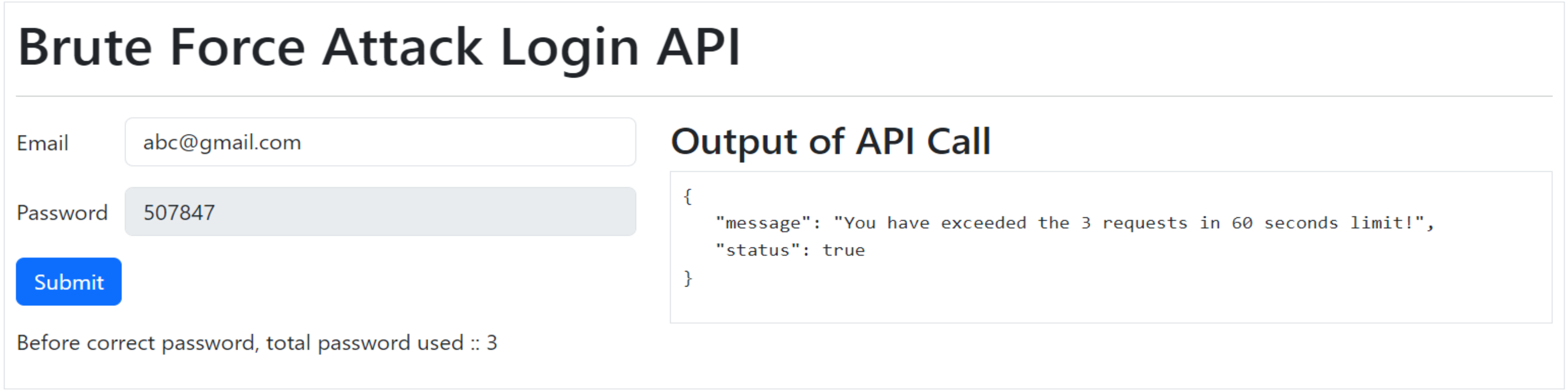
Brute Force Attack Login API

Email	abc@gmail.com
Password	214130

Before correct password, total password used :: 2

Output of API Call

```
{
   "status": false,
   "message": "Invalid credentials",
   "query": "SELECT * FROM user WHERE email='abc@gmail.com' AND password='214130
}
```



Brute Force Attack Login API

Email	abc@gmail.com
Password	507847

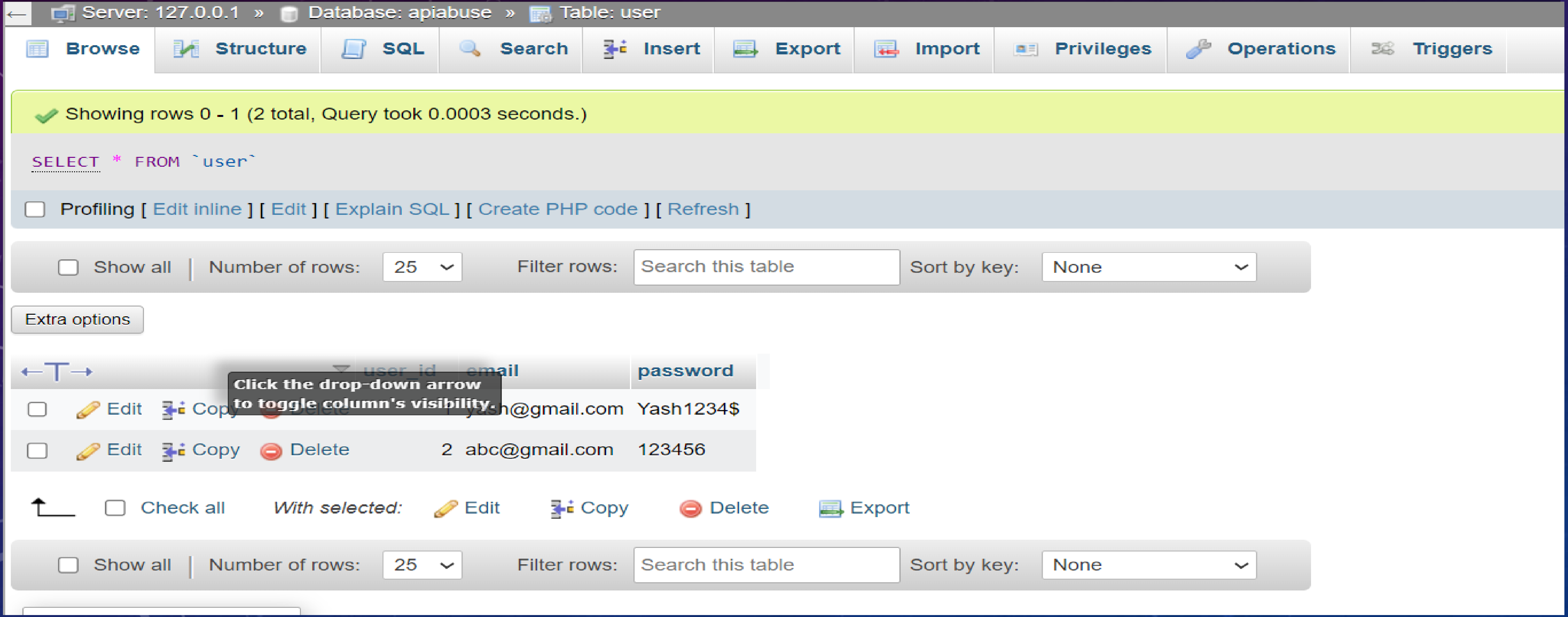
Before correct password, total password used :: 3

Output of API Call

```
{
   "message": "You have exceeded the 3 requests in 60 seconds limit!",
   "status": true
}
```

Database : MySQL

- Database Name : apiabuse → Table name : user



Server: 127.0.0.1 » Database: apiabuse » Table: user

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

✓ Showing rows 0 - 1 (2 total, Query took 0.0003 seconds.)

```
SELECT * FROM `user`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

Click the drop-down arrow to toggle column's visibility.

	user_id	email	password
Edit Copy Delete	1	yash@gmail.com	Yash1234$
Edit Copy Delete	2	abc@gmail.com	123456

Check all *With selected:* Edit Copy Delete Export

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

APIs:

We have created two APIs for this demos:

1. http://localhost:5000/api/users/login
   - Type : POST
   - Data :{
     Email: "",
     Password :""
     }

2. http://localhost:5000/api/users/bruteForceLogin (With rate Limiter)
   - Type : POST
   - Data :{
     Email: "",
     Password :""
     }

What technology we use ?

- Back-End :
  - NodeJS (Express JS)
  - MySQL (package)
  - Express rate limit (package)
- Server and Database :
  - Xampp (Apache)
  - MySQL
- Front-End :
  - Bootstrap
  - JQuery
  - Validate JS (Regex)

Resources

- Project URL : https://github.com/yashpatel521/API-Abuse
- Bootstrap :
  - https://getbootstrap.com/
- JQuery:
  - https://jquery.com/
- Validate JS:
  - https://validatejs.org/
- Express JS :
  - https://expressjs.com/
- Express rate limiter :
  - https://www.npmjs.com/package/express-rate-limit
- MySQL :
  - https://www.npmjs.com/package/mysql

Thank You ….

Any Questions