The Good, The 400 And The Ugly of APIS

Christopher Miller - Lead Full Stack Developer - Treblle

Presented By treblle

Who Am I?

Name: Christopher Miller

Position: Lead Full Stack Developer

Socials: ccmiller2018 - on everything

Time In Industry: 23 years

The Good

Three Thoughts About The Good

A Horror Story: Why Good Apis Matter

A Beautiful Option: The Characteristics Of Good Apis

The Right Way: Best Practices For Designing Apis

Presented By treblle

The Bad

Three Thoughts About The Bad

Users Are Stupid: Why Error Management Matters

Users Are Impatient: Why Speed Matters

Users Are Inconsistent: Why Stateless Matters

The Ugly

Three Thoughts About The Ugly

Hackers Are Opportunistic: Why Authentication Matters

Hackers Are Consistent: Why Non-Sequential Matters

Hackers Are Observant: Why Data Management Matters

Presented By treblle

The Good: A Horror Story: Why Good Apis Matter

Honda Made A Mistake

What happened?

Security researcher Eaton Zveare discovered two vulnerabilities in Honda’s e-commerce platform These vulnerabilities allowed anyone to reset the passwords of any account on the platform, and to access the admin panels of all Honda dealers.

Presented By treblle

The Good: A Horror Story: Why Good Apis Matter

Honda Made A Mistake

What data was exposed?

The password reset vulnerabilities exposed customer data including names, addresses, email addresses, and phone numbers. The admin panel vulnerabilities exposed even more sensitive data, including dealer sales figures, inventory levels, and customer purchase history.

Presented By treblle

The Good: A Horror Story: Why Good Apis Matter

Honda Made A Mistake

How did it happen?

The vulnerabilities were caused by a lack of access controls on the platform. The password reset API did not require a token or the previous password, and the admin panels were accessible to anyone who knew the user ID of a dealer.

Presented By treblle

The Good: A Horror Story: Why Good Apis Matter

Honda Responded Badly

The vulnerabilities were discovered in January 2023
Honda did not disclose them until June 2023

Presented By treblle

The Good: A Beautiful Option: The Characteristics Of Good Apis

Three Main Sections

- Design
  - Well Documented Endpoints - don’t just document good responses
  - Multiple HTTP Methods - don’t use get for everything
  - API Versioning - versioning allows users to use the right version
  - JSON Support Consistently - json makes integration easy
  - Singular Wording For Urls - makes discoverability easy
  - Nouns not Verbs For Urls - makes discoverability easy

The Good: A Beautiful Option: The Characteristics Of Good Apis

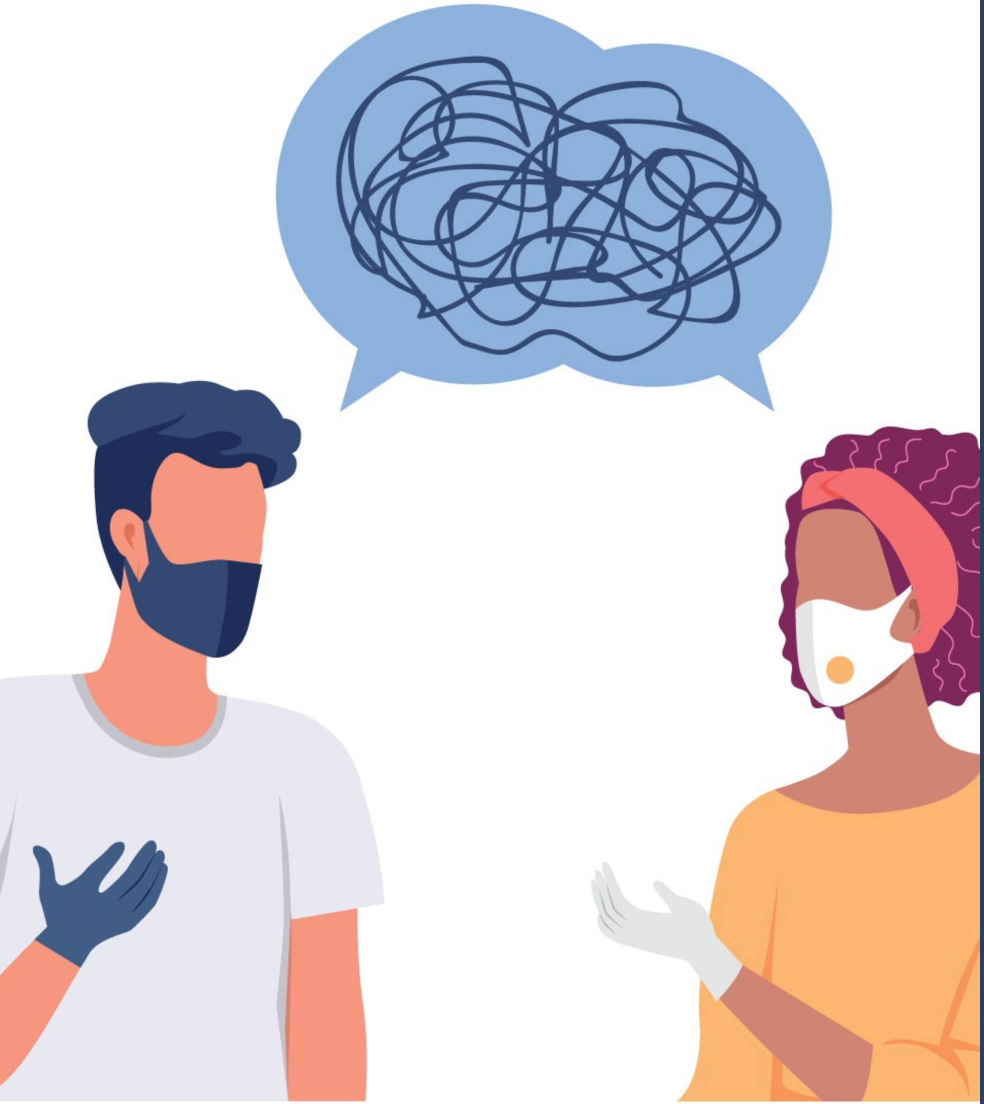
Three Main Sections

- Security
  - Authorization - you can trace who did what easily
  - HTTPS not HTTP - it's encrypted in transit
  - Rate Limiting - you can't just constantly hit the route
  - Content Type Control - you control the type of content allowed
  - Forcing A Secure Connection - ensures in transit encryption

The Good: A Beautiful Option: The Characteristics Of Good Apis

Three Main Sections

- Performance
    - Low Load Times
    - Small Response Sizes
    - CDN Usage
    - HTTP/2 or HTTP/3
    - Caching Support
    - Compression Support

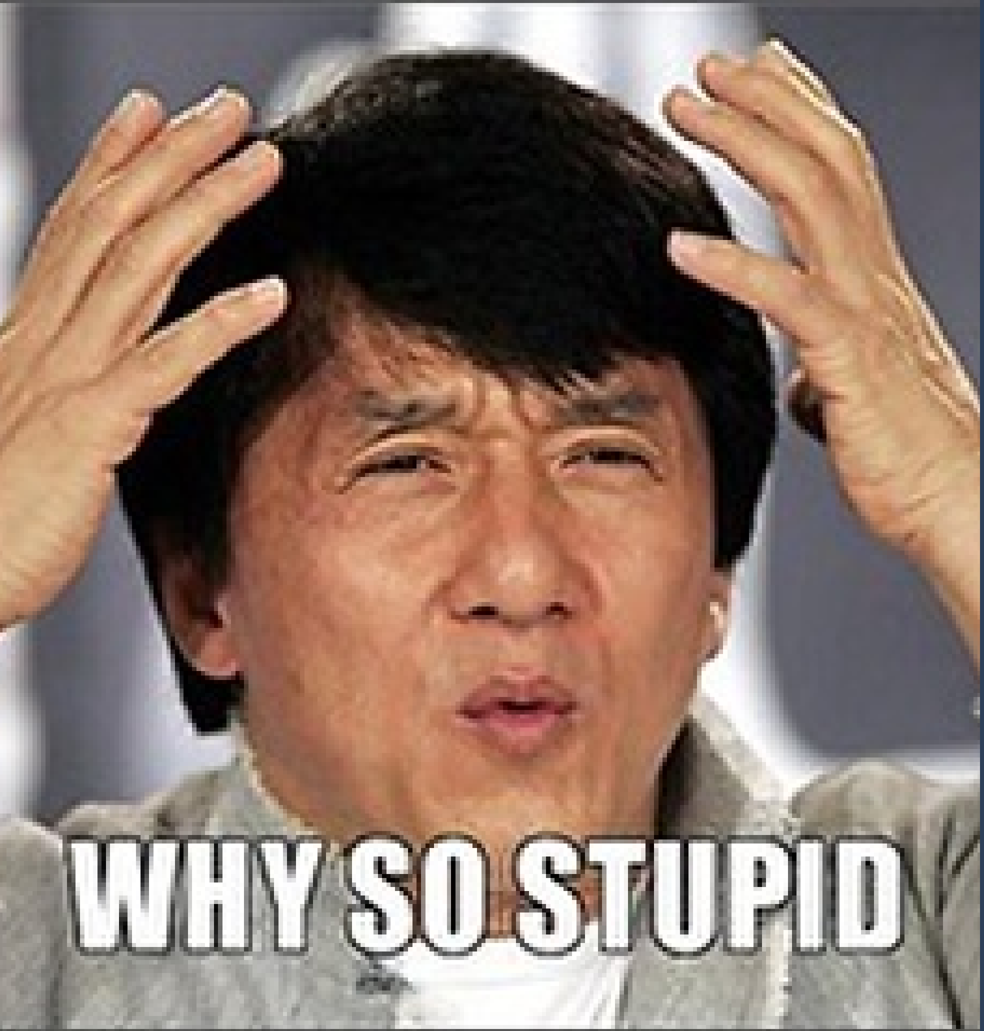


The Good: The Right Way: Best Practices For Designing Apis

Apart from the characteristics:

- Talk to your users
- TALK to your users
- TALK TO YOUR USERS

Presented By treblle



The Bad: Users Are Stupid: Why Validation Matters

- User puts Telephone Number In Email Field
    - Can't Log In
    - Can't Reset Password
- User Puts Email In Telephone Field
    - 2FA Fails
    - Can't Get Texts

Presented By treblle

The Bad: Users Are Stupid: Why Validation Matters

Handling It Is A Case Of Thinking

imagine the following typescript definition:

```typescript
interface User {
  first_name: string;
  last_name: string;
  other_names: string;
  email: string;
  telephone: string;
  password: string;
  roles: array;
}
```

Is this enough?

The Bad: Users Are Stupid: Why Validation Matters

The Answer Is NO - Its Not

so how do we fix it? enter class-validator

```
export class User {
    @isString()
    @length(3, 255)
    first_name: string;
    // removed other_names and last_name
    @isEmail()
    email: string;
    @isTelephoneNumber()
    telephone: string;
    @isSecurePassword()
    password: string;
}
```

The Bad: Users Are Stupid: Why Validation Matters

The Answer Is NO - Its Not

And use it in our function:

```
app.post('/api/users', async (req, res) => {
  try {
    const userData: User = req.body;
    const errors = await validate(userData);
    if (errors.length > 0) {
      return res.status(400).json({ errors });
    }
    res.status(201).json({ message: 'User created successfully!' });
  } catch (err) {
    res.status(500).json({ error: 'Internal server error' });
  }
});
```

The Bad: Users Are Impatient: Why Speed Matters

Your users will give up - quick!

Generally, APIs that are considered high-performing have an average response time between 0.1 and one second. At this speed, end users will likely not experience any interruption. At around one to two seconds, users begin to notice some delay. Around five seconds, users will feel a significant delay and may abandon the application or website as a result.

Api Response Time, Explained in 1000 Words Or Less
Jamie Juviler
June 24th 2022
https://blog.hubspot.com/website/api-response-time

Presented By treblle

The Bad: Users Are Impatient: Why Speed Matters

our database takes 1 millisecond per record to collect

our database has 20 million records

our loop takes 1 millisecond per record to add up

that’s now 4,000 seconds (or 1 hour 6 minutes 40 seconds)

```
app.get('/posts', async (req, res) => {
        let z = 0;
        const data = await postsRepository.getAll();

        for (let i = 0; i < data.count; i++) {
            z += data[i].commentCount;
        }

        res.send({
            allCommentsCount: z,
            posts: data,
        });
});
```

Presented By treblle

The Bad: Users Are Impatient: Why Speed Matters

What's The Solution?

our database takes 1 millisecond per record to collect

our database has 20 million records

but we only get 50 records

so that's only 50 ms

```
app.get('/posts', async (req, res) => {
    // this would be set via config - example only
    const limit = 50;
    const page = req.params.page ?? 0;

    const data = await postsRepository.paginated(
        page,
        limit
    ).getAll();

    res.send({
      posts: data,
      page: page,
      limit: limit
    });
});
```

Presented By treblle

The Bad: Users Are Inconsistent: Why Stateless Matters

Users’ behavior can be inconsistent when interacting with applications or systems.

Inconsistent user behavior can lead to varying requests and data needs.

Stateless APIs treat each user request independently, ignoring past interactions.

This approach ensures predictable and reliable responses, regardless of user history.

Avoids potential conflicts and confusion caused by relying on user state information.

Simplifies application design and maintenance by not storing user-specific data.

Enables easier horizontal scaling as requests can be distributed without considering user state.

The Ugly: Hackers Are…

The examples here are all from real security issues I’ve seen

The Ugly: Hackers Are Opportunistic: Why Authentication Matters

An API i was using wanted me to pass the API key as an X-API-Key Header

- I called /user/abc123/persona
- I got the full details back

No Issue here - I was using the X-API-Key Header

EXCEPT...

The key was 'TEST-KEY'

Their authentication was broken. abc123 was my user. but xyz987 wasnt!

I had access to everything! every. single. users details

The Ugly: Hackers Are Opportunistic: Why Authentication Matters

```
app.use(passport.initialize());

passport.use(
  new BearerStrategy((token, done) => {
    if (!userRepository.hasToken(token)) {
      return done(null, false);
    }
    return done(null, true);
  })
);

app.get(
  '/protected-route',
  passport.authenticate('bearer', { session: false }),
  (req, res) => {
    res.json({ message: 'Access granted. This is a protected route.' });
  }
);
```

Presented By treblle

The Ugly: Hackers Are Consistent: Why Non-Sequential Matters

how do we count to three?

1

2

3

how does a uuid count to three?

d21a739a-1028-4441-987f-e16af308565a

adecb885-1630-478c-af92-e6e754a97e10

d5d89e52-7d36-46f3-b1d6-3b5ec001cfec

The Ugly: Hackers Are Consistent: Why Non-Sequential Matters

let's make a vulnerable route for api.example.com

```
app.get('/users', (req, res) => usersRepository.getById(req.userId).toJsonResponse();
```

and a program to exploit sequential numbers

```
let userId = 1;
while(true) {
    try {
        const response = await axios.get(`https://api.example.com/users/${userId}`);
        if (response.status !== 200) { break; }
        fs.writeFileSync(`${userId}.json`, JSON.stringify(response.data, null, 2));
    } catch (error) { break; }
    userId++;
}
```

Presented By treblle

The Ugly: Hackers Are Observant: Why Data Management Matters

```
app.get('/users', (req, res) => usersRepository.getById(req.userId).toJsonResponse();
```

But it also has another problem: the response

```
{
  ...,
  "email": "chris@treblle.com",
  "password": "5f4dcc3b5aa765d61d8327deb882cf99",
  "roles": [
    "Administrator"
  ],
  ...
}
```

The Ugly: Hackers Are Observant: Why Data Management Matters

```
async function readJsonFilesSequentially(fileNumber = 1) {
  const data = [];
  while (true) {
    try {
      const fileData = await fs.promises.readFile(`${fileNumber}.json`);
      const jsonData = JSON.parse(fileData);
      const decryptedPassword = crypto.createHash('md5').update(jsonData.password).digest('hex');
      data.push({ user_id: fileNumber, password: decryptedPassword });
      fileNumber++;
    } catch (error) {
      break;
    }
  }
  await csvWriter.writeRecords(data);
}
```

Presented By treblle

WHAT???

Our hacker from one route now has

- Every User
- Every Users Email
- Every Users Password
- Every Users Roles

The Good, The Bad and The Ugly Of APIS

Let’s sum up everything we’ve said

- APIs are a daily tool for the web
- When writing APIs, consider how it will be used
- When writing APIs, consider that users can get it wrong
- When writing APIs, consider that hackers will try it out if its public