Member-only story

A basic question in a security Interview: How do you store passwords in the database?

LORY · Follow
7 min read · May 12, 2024

Explained in 3 mins.

Last week's interview story

Position: junior sec engineer. fresh grad.

Let's start the interview.

Me: "Here you mentioned having a good understanding of data security. Could you please give me some samples of which part of the data securities?"

A: "Sure. For example, when we build a system there will be a user module, and when we store passwords in the database user table, we need to encrypt it before storing"

Me: "Are you sure it is encryption but not hashing?"

A: "Yes."

Me: "Then where do you store the keys"

A: “What key?”

Me: “The encryption key which you used to encrypt the password. and it is asymmetric or symmetric encryption? also is it one user per key or the key is shared?”

A: “Hmm. We are not using all these. Then it must be hashing”

Me: “No problem. could you explain why we need to hash it before storing, it instead of storing the plain text?”

A: “Yes. because we want to archive safety. When we validate the password, the password can not be sent as plaintext from UI to the server to do validation”.

Me: “When you register a user, do you need to send the password and confirmation password as plaintext?”

A: “Yes. but that’s the only use case we send the plaintext”

Me: “OK I got your point. so the reason you think storing hash value in database is to minimize the chance to send sensitive data as plaintext during API calls?”

A: “Exactly.”

Me: “Then do you think it will make sense to store hash value when the database gets compromised by someone?”

A: “Oh yes. then the original password will be shown to hackers. the hashing value can prevent this.”

Me: “Yes but not enough. it’s still possible for someone using a rainbow table to attack the table and get the original value through hash collision. do you know about salt?”

A: “Heard of it. the value should be a random string, which will make password storage safer”

Me: “Good. Do you know how it works?”

A: “Not really”.

Me: “So in case someone got the database through SQL injection or some other way. then he uses the Rainbow table to attack, let’s say the hash collision happened and he got the original text, but the value he got is the original password blended with a random string, so in a way, we can prevent the original password from being stolen. which means we stopped some wrong hand from getting your ‘one for all’ password to login your mail, social media, and even bank account.”

A: “I see”.

Me: “So which hashing algorithm you are using in your current system?”

A: “So as we know md5 is not safe. so we used some other options, like sha3”

Me: “Could you brief me bit why md5 is not safe? and why choose sha3 to hash the password? because AFAIK the sha3 KDF like hmac-sha3 is not designed to hash passwords. but for other purposes like JWT token signatures.”

A: “md5 is proven as not safe. there is already some news. but what is KDF?”

Me: “KDF is a key derivate function. which means the hashing algorithm you use takes a password as input to generate a hash value. yes, there is news saying md5 is not safe. but do you know why?”

A: “Hmm not really.”

Me: “In short it is due to weak hash collision resistance. and is possible to be attacked by using rainbow table.”

A: “What is rainbow table?”

Me: “A rainbow table is a precomputed table that contains chains of hash values derived from plaintext passwords. These chains are generated by iteratively

applying a hash function to an initial plaintext password, and then reducing the resulting hash value back to a plaintext password through a series of reduction functions. Rainbow tables are typically used in password cracking attacks, where an attacker compares hashed passwords stored in a compromised database against entries in the rainbow table to find matching plaintext passwords."

A: "I see".

Me: "And why are you using sha3 to generate a password?"

A: "It's safer than MD5 I think."

Me: "Yes it is. however, there could be better hashing options for passwords. like argon2 which won the prize in 2015, compared to sha3 which requires more iterations and higher RAM which could mean a higher time complexity and resource consumption in order to generate one single hash. and which makes it harder for the attacker to brute-force. but sha3 is usually used for other purposes, e.g. digital signature, balance the security and performance".

...

Let's address the details one by one from this interview.

Password hash

We all know that passwords should never be saved as plaintext in the 21st century.

- However, It is not about reducing the chance of sending password value as plaintext during some API calls.
- The main concern is when databases fall into the wrong hands. make it impossible to get back the original text.
- What will happen next is the database will be "Credential-Stuffing". The same credentials will be tried again and again on different accounts with the same user name: email, bank cards, social media, school, even gov account.

Why not use encryption but hashing?

- The key difference lies in reversibility. Encryption is a reversible process, meaning that with the right key, the original plaintext password can be recovered. This poses a security risk, even if encryption keys are securely managed.

- Additionally, managing encryption keys, such as building a key chain, can incur significant costs and complexity.

And why need salt

- Salt serves to add randomness to each hashed password, making it computationally expensive for attackers to crack passwords using precomputed tables like rainbow tables (will explain below).

- Without salt, attackers could efficiently guess passwords by comparing hash values against a large database of precomputed hashes.

- Additionally, salt must be unique per password, not just per user. This ensures that even if two users have the same password, their hashed values will be different due to the unique salt, preventing attackers from easily identifying duplicate passwords.

Different purposes for using hash

Sha3 is safer than md5, why not use sha3? Hashing is not just used in password value, there are different use cases:

1. Data integrity (e.g., SHA3): Hashing algorithms like SHA3 are commonly used to fingerprint files, ensuring their integrity remains intact. These algorithms need to strike a balance between speed and hash collision resistance. While SHA3 offers good collision resistance, it also maintains reasonable performance for file fingerprinting tasks.

2. Checksum (e.g., CRC32): When speed is of utmost importance, as in verifying data integrity during network transfers, algorithms like CRC32 excel. CRC32 is often used to quickly identify if a file has been modified during transmission, prioritizing efficiency over collision resistance.

3. Key generation (e.g., Argon2, bcrypt, PBKDF2): For securely storing passwords or generating cryptographic keys, specialized key derivation functions (KDFs) like Argon2, bcrypt, or PBKDF2 are preferred. These algorithms intentionally introduce a "slow" hashing process, adding layers of computational complexity to fight against a hash collision brute-force.

While SHA3 is suitable for tasks like HTTPS connections or JWT token signature generation, it lacks the necessary slowness to provide the heightened security required for password hashing.

Why KDF matters in hashing password

KDFs play a crucial role in password hashing by intentionally slowing down the hashing process. This deliberate slowdown is achieved through multiple iterations of hash value computations. While the technical details of these iterations can be complex, they essentially involve repetitive operations like shifting, inverting, and XOR-ing the hash values, performed over thousands of rounds.

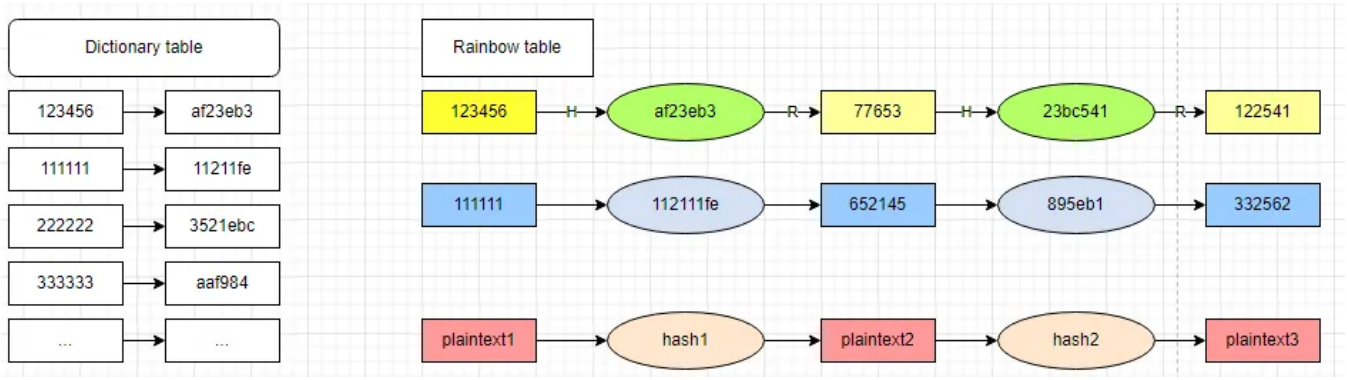
The primary goals of using KDFs in passwords are:

1. Slowing down the process: By introducing computational overhead, KDFs make it significantly harder for attackers to brute-force passwords. The increased time and computational resources required to compute hash values act as a deterrent against rapid dictionary or rainbow table attacks.

2. Resource consumption: KDFs also consume significant CPU and memory resources during the hashing process. This resource-intensive nature further impedes attackers, as they must invest substantial computational power and time to crack hashed passwords.

What is the rainbow table attack?

A quick explanation.



- Dictionary table. before the rainbow table, attackers precompute all the hash values and enumerate all the hash values in a table called “dictionary table” Then after getting the target database table, do hash collision and get the linked original password value (again, that is why salt is so important)
- However, a dictionary table could grow very fast and end up with a huge number of rows made it unusable.
- The rainbow table could be considered “an efficient version of a dictionary table”
- In a rainbow table, 2 functions. H: to get hash value from plaintext, R: to get plaintext from hash value
- Stores multiple rows. each row is a “hashing chain”, every row starts with a plain text and then does H and R multiple times.
- Doing hash collision with a target database table, once the hash value matches, do the R to get the plaintext.

Last

why md5 (or sha1) can not be used in password value hash? if yes are there any other use cases for md5?

- The quick answer: Weak hash collision resistance.
- md5 has been proven weak hash collision resistance in 2004

- The same happened to sha1 in 2005 and announced by Google in 2017.
- How about we use md5 for other purposes? like fingerprint or checksum? no. it is proven, that 2 different files can be generated with the same md5; if used for checksum, crc32 has a much faster performance.
- there are no used cases for md5, you should never use it.

Thanks for reading and see you in the next post.

Security Interview Interview Questions Programming Hashing

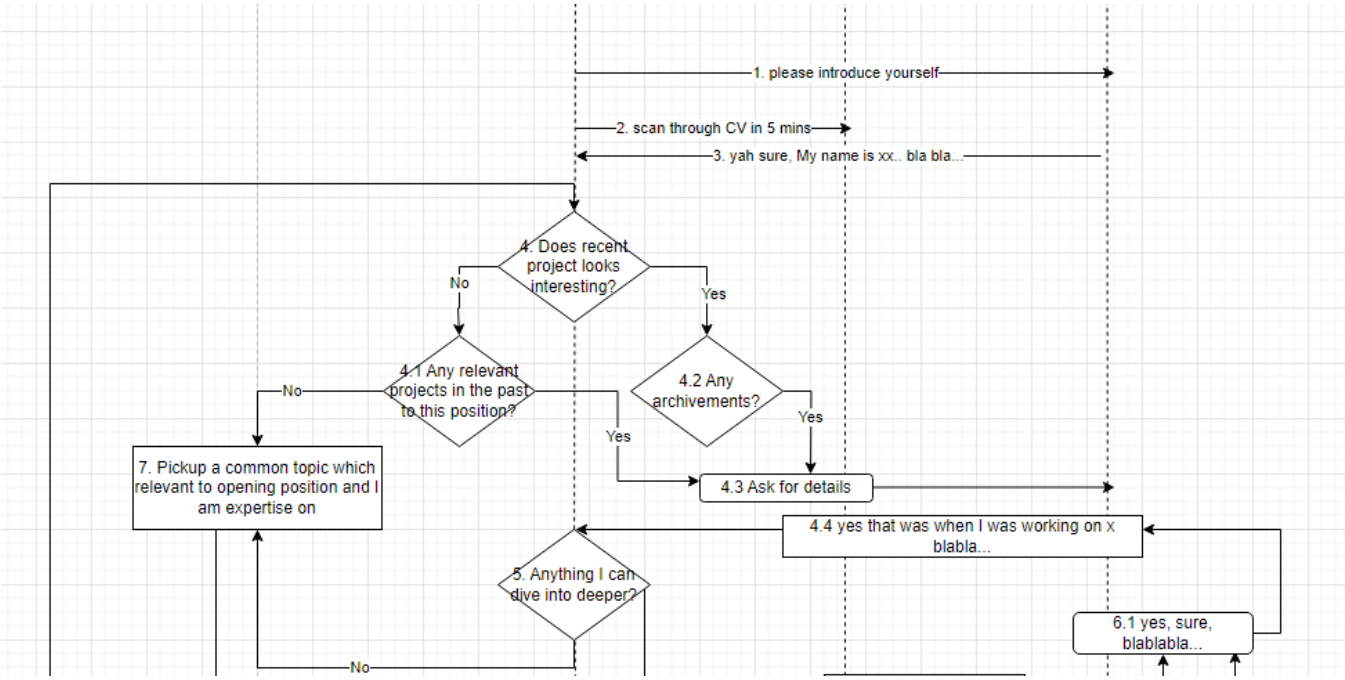
Follow

Written by LORY

12.5K Followers

A channel which focusing on developer growth and self improvement

More from LORY

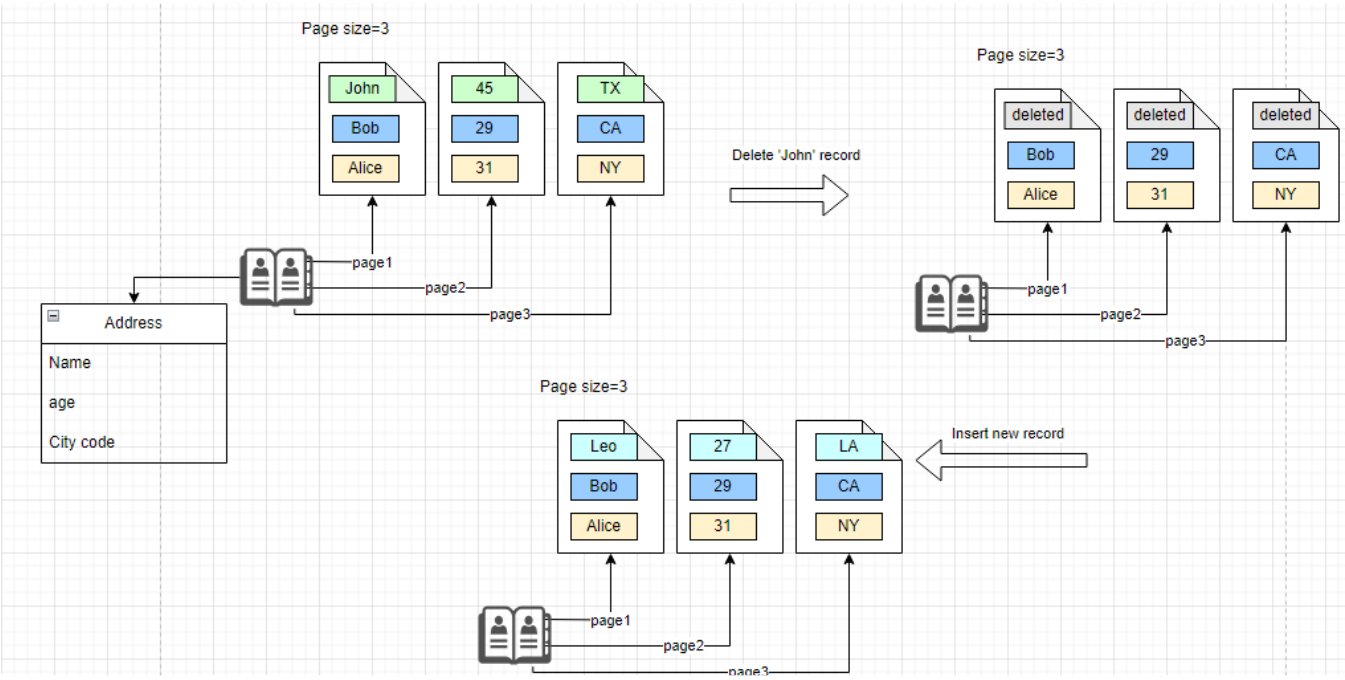


LORY

Today I Interviewed for a Lead Front-End Role

And They Asked Me a Couple of Tough Questions

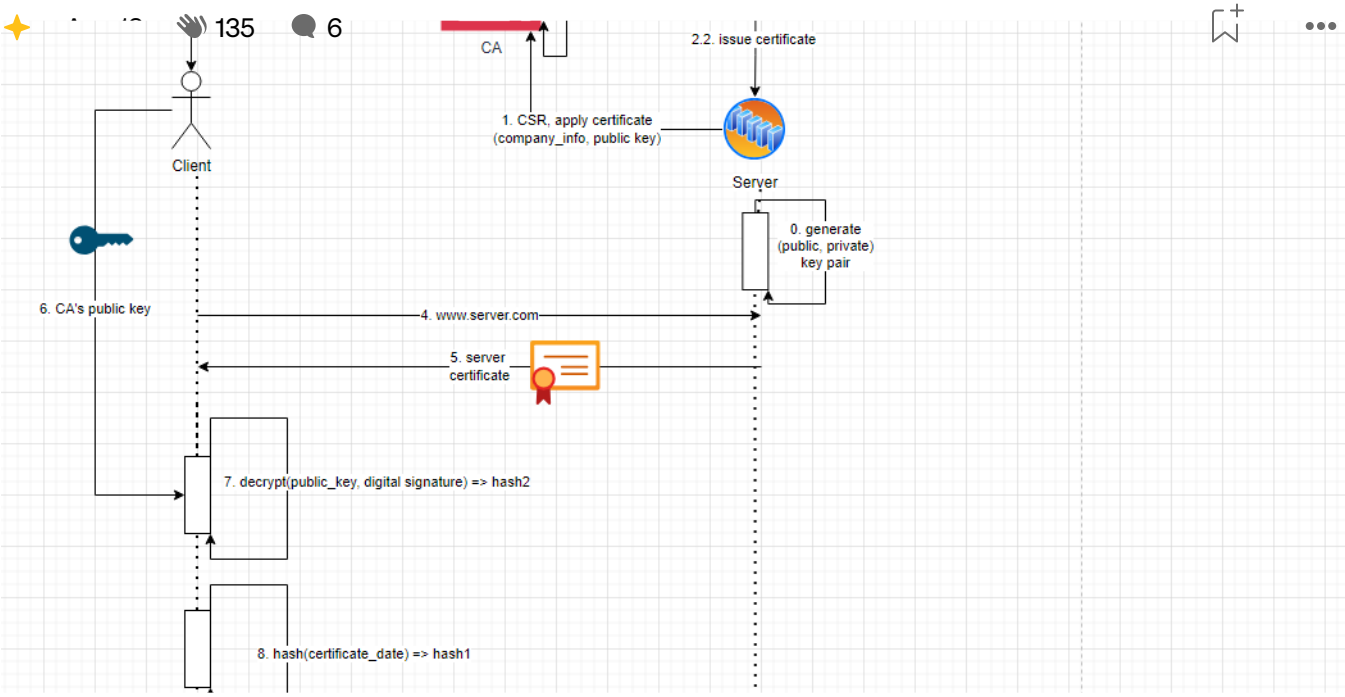
Aug 4 824 14



LORY

My Friend Asked These Questions to his interviewers but Couldn't Get an Answer

So let me try to help them, bro.



LORY

Interviewer: Why self-sign is not safe?

SSL certificate verification process explained in 3 minutes

Jun 5 476 7

LORY

Before You Pick Up the Offer, Make Sure Read This

Don’t fall into the same traps again.

Sep 1 67

See all from LORY

Recommended from Medium

Amazon.com Seattle, WA

Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)

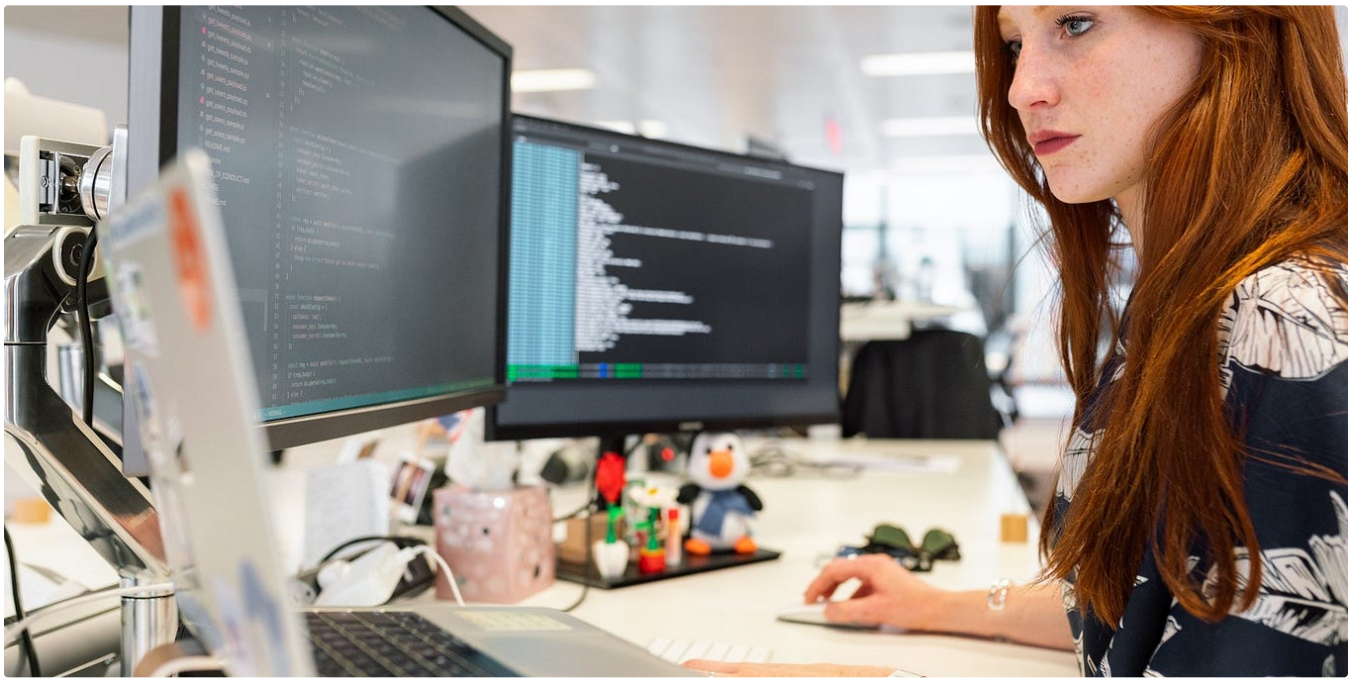
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 22K 444



The Coding Diaries in The Coding Diaries

Why Experienced Programmers Fail Coding Interviews

A friend of mine recently joined a FAANG company as an engineering manager, and found themselves in the position of recruiting for...

Nov 2, 2022 9.7K 182

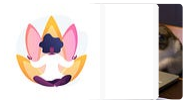
Lists

General Coding Knowledge

20 stories · 1598 saves

Coding & Development

11 stories · 820 saves

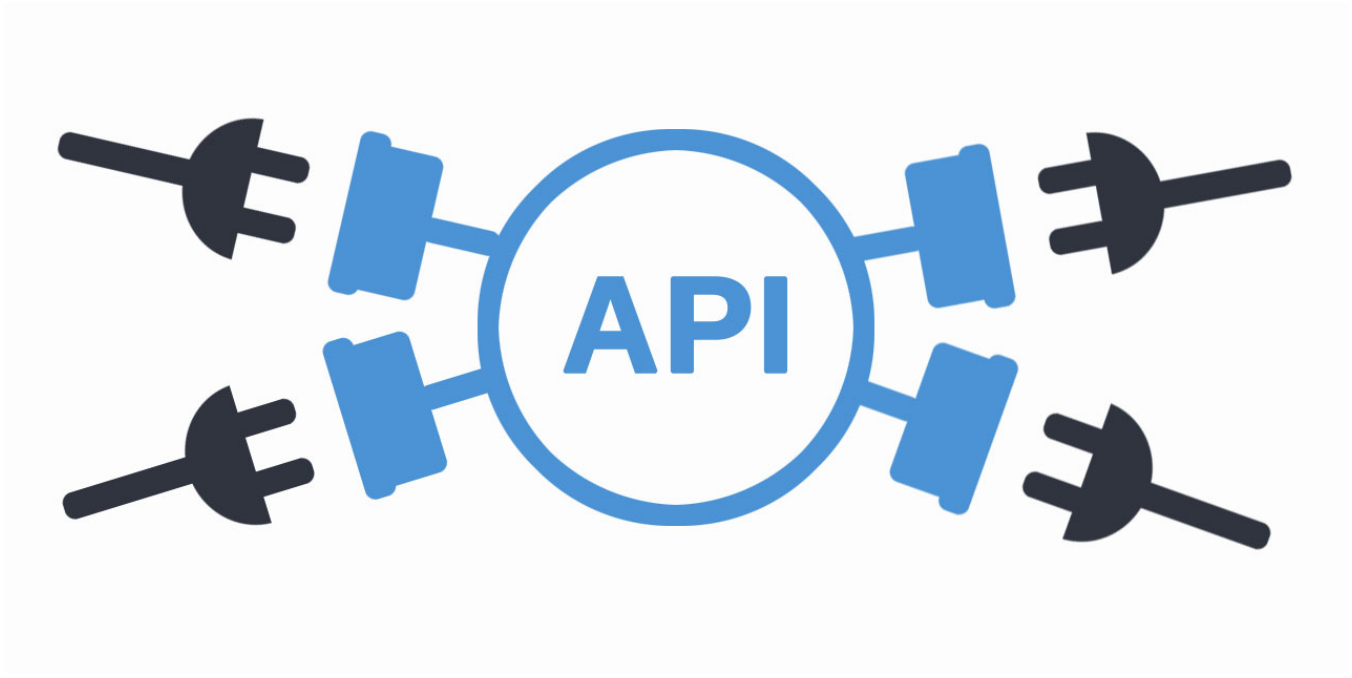


Stories to Help You Grow as a Software Developer

19 stories · 1378 saves

ChatGPT

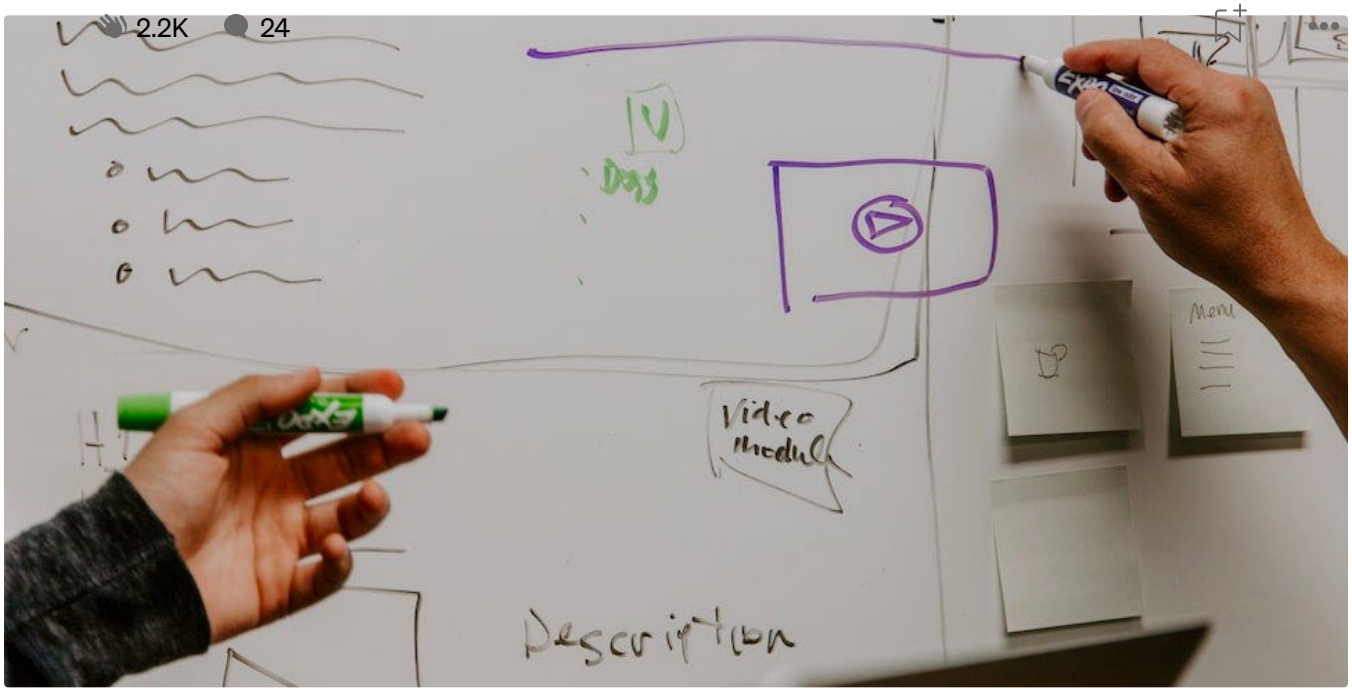
21 stories · 815 saves



Seliesh Jacob in Dev Genius

API Design: From Basics to Best Practices

Introduction



Carlos Arguelles

My favorite coding question to give candidates

A coding question, from the viewpoint of an Google/Amazon/Microsoft interviewer

Nov 12, 2023 9.4K 95

Grant Piper

Why Can’t Robots Click The “I’m Not a Robot” Box On Websites?

Clicking a tiny box tells Google all they need to know about your humanity


```
const promise1 = new Promise((resolve, reject) => {
  console.log(1)
  resolve(2)
})

promise1.then(res => {
  console.log(res)
})

console.log('end');
```

start

Shuai Li in Programming Domain

Can You Answer This Senior Level JavaScript Promise Interview Question?

Most interviewees failed on it.

Aug 12 2.4K 22

See more recommendations