



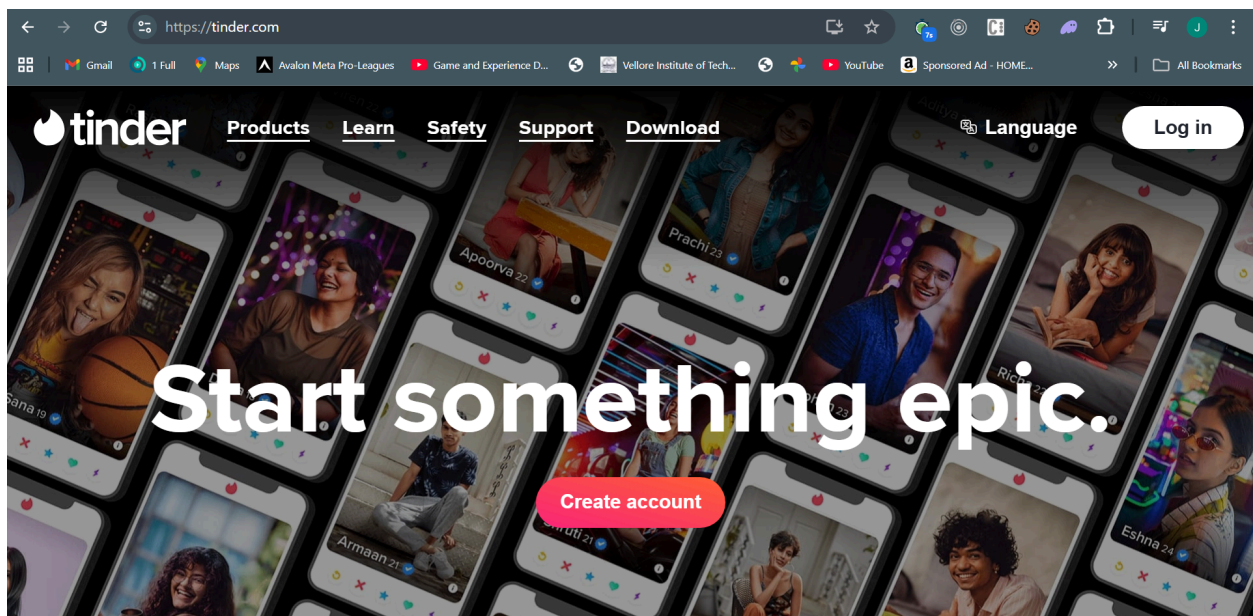
# Episode-02 | Features, HLD, LLD & Planning

this is the job of product manager but here we are the one who si doing job of everyone



so what exactly the devtinder is

people who dont know about the tinder platfromd lets see



when u enter its will ask u to create your profile ask u for your firstname gender intrestes etc

in tinder when u swipe right it accepted the connection request when u left swipe means rejected

we will also have implemented all the connections

messages tab is also there where u can chat with people

so features we are building

1. create an account
2. login
3. update your profile
4. Feed Page - explore (see other developers)
5. send connect Request
6. see our matches
7. see the request we have sent
8. update your profile

u can add many more features in it like bloating the profile make chat groups etc

in company generally product manager work with ui engineers how login app will look like how buttons will look like so we don't have designers we will build ui of our own

after these features decided

now the role of tech teams comes in picture

how u design db apis ui technical things

tech planning

we will create 2 microservices

1. frontend
2. backend

next we decide tech stack

backedn

-nodejs

-express

-mongodb

frotntend

-react

now the backend engineers sde comes in picture to write code but bfore wriitng code they do lld nd hld

if u spent lot of time in palnning than writing code is very very easy

in planning u decide lld hld

the first two imp things in lld is

1. db design
2. api design

1 db desing

in this we think about the collection how we store the data when using mongo we use term collection rather than tbale

so we have seprte collection

1. user collection

inside it we have

1. fristName
2. lastName
3. emailld

4. passowrd
5. age
6. gender
7. e.tc

now when we create account we put the data into user cllctn

when we login we user collection

in feed page we again need user cllctn to shwo the data

in send connection request

now agai i need the new collection

how do we store conenctions

here comest he tricky part have u heard about rlthnship diagrams

in dbms subjt

i cannot store connections rlthsnhp into the user collection i need seprte colleciton for that

so i careate aanotehr collection

## 2. connectionRequest

suppose a sends request to b so we need who sending to whom  
connect status

- from UsrId
- to userId
- status = pending

if ur using sql then u have to be mor emore consius becasue schema changes in sql is big pain as comapare to mongodb  
that is why palnning is very very imporant

so till now we have created two collections

now i ahve small task for u think about what are the status will have here like pending rejecting accepted but there can be multiple status will have depdnding on yr prjct

suppose

$A \rightarrow B$

pending from pending it can be move to accepted or rejected

can it have more status?

when u open tinder u can right or left sqipe so u can ignre the prlfe also so status can be ignored also ther can be status blocked also (more compelx)

so when u right sqipe sttus goes to pending than accepted or rjected

but when u left swipe status goes to ignored

Now lets move to the API design

we are going to use rest api

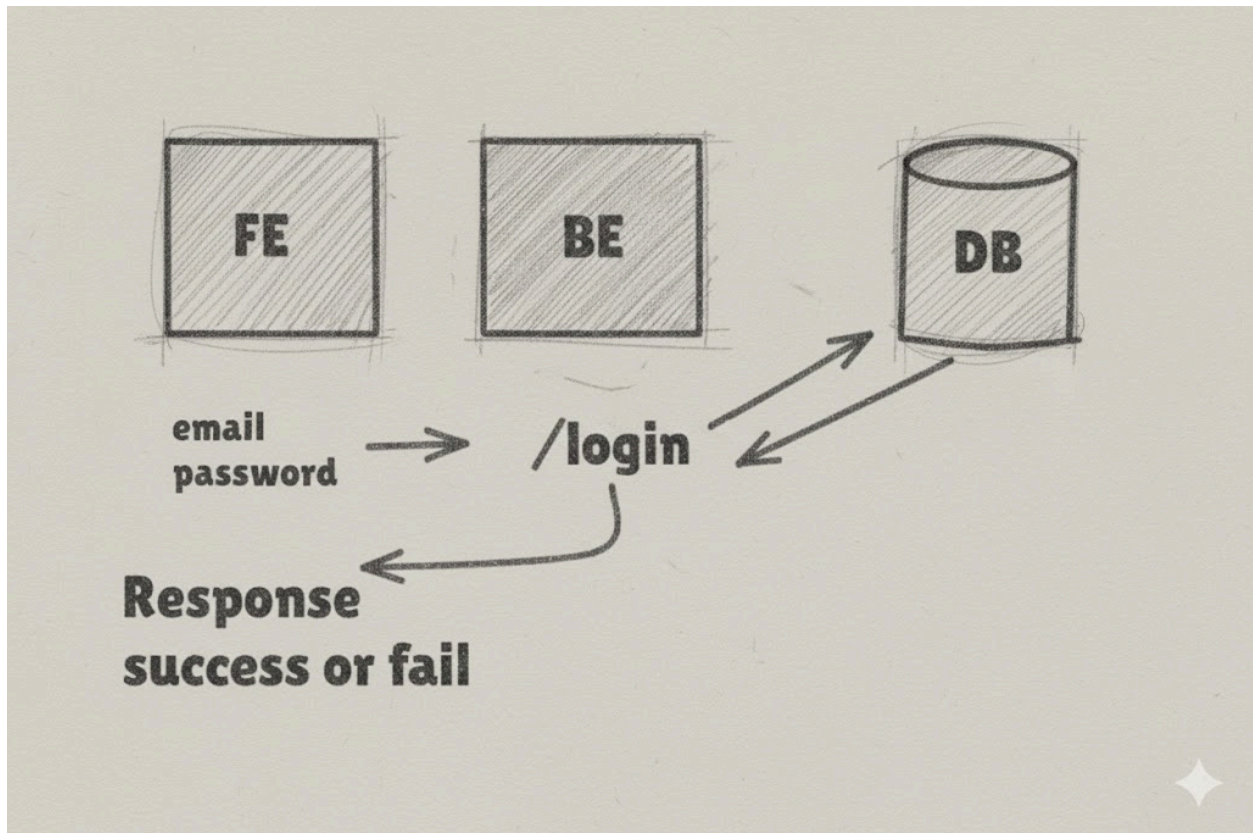
what is Rest API?

**A REST API (Representational State Transfer Application Programming Interface)** is a standardized way for different software systems to communicate with each other over the internet using the **HTTP protocol**. It defines a set of rules and conventions that allow clients (like a frontend application or mobile app) to send requests to a server (backend) and receive responses in a predictable format — usually **JSON**.

In simple terms, a REST API acts as a **bridge** between the client and the server. The client doesn't directly interact with the database or internal logic; instead, it sends HTTP requests (like **GET**, **POST**, **PUT**, or **DELETE**) to specific endpoints (like `/login`, `/users`, `/products`), and the server responds with data or a confirmation message. Each endpoint in a REST API represents a specific resource or functionality, and each HTTP method represents an operation on that resource — for example, `GET /users` fetches data, `POST /users` creates a new user, `PUT /users/:id` updates existing data, and `DELETE /users/:id` removes it.

A key principle of REST is **statelessness**, meaning every request from the client must contain all the information the server needs to process it — the server doesn't store session data between requests. This makes REST APIs lightweight, scalable, and ideal for distributed systems like modern web and mobile apps.

In practice, when you log into a website or fetch data from an app, you're likely interacting with a REST API — where your request travels from the **frontend (FE)** to the **backend (BE)**, which then communicates with the **database (DB)**, processes the request, and sends back a response indicating **success or failure**.



## 🧩 Components in the Diagram

- **FE (Frontend)** → The client or user interface (like a React app or mobile app).
- **BE (Backend)** → The server that exposes REST APIs and handles requests.
- **DB (Database)** → The system where user data (like email, password, profile info, etc.) is stored.

## 🔄 REST API Flow (using `/login` example)

### 1. User Action (Frontend → Backend)

- A user enters their **email** and **password** in the frontend (FE).
- When they click **Login**, the frontend sends an HTTP **POST request** to the backend endpoint:

```
POST /login
```

- This request body contains `{ email, password }`.
- 

### 1. Backend Processing (BE → DB)

- The backend receives the `/login` request and checks if the provided email and password exist in the database.
  - It communicates with the **DB** by querying user records (e.g., `SELECT * FROM users WHERE email = ?`).
- 

### 1. Database Response (DB → BE)

- The database returns the user data (if found) or indicates no match.
  - The backend compares the stored password hash with the one provided.
- 

### 1. Backend → Frontend Response

- If the credentials are correct → backend sends a **success response** (e.g., `{ status: "success", token: "abc123" }`).
  - If not → it sends a **failure response** (e.g., `{ status: "fail", message: "Invalid credentials" }`).
- 

### 1. Frontend Handling

- The frontend receives the backend's response.
  - If it's successful, it may:
    - Store the authentication token (e.g., JWT) in localStorage or cookies.
    - Redirect the user to a dashboard.
  - If failed, it may:
    - Show an error message like "Wrong email or password."
- 

## Why It's a REST API

- REST APIs use **HTTP methods** (GET, POST, PUT, DELETE) for communication.
- Data is exchanged in a **stateless** way (each request is independent).
- Responses are typically in **JSON format**.

---

The frontend calls a REST API endpoint (/login) on the backend, the backend verifies data with the database, and returns a **success or failure response** — completing one RESTful interaction.

---

what are the different types of REST APIs

1. Get - fetch

used to get the data from the service

1. post

used to post the data

2. put

used to put the data

4. DELETE

used to delete the data

all of these are HTTP methods

there is a subtle difference between post and put

it's your task to read about this in your interview questions also

now let's discuss the APIs we need to build in our project

all these operations known as CRUD operations

1. POST /SIGNUP

2. POST /LOGIN

3. GET /profile

4. POST /profile

5. PATCH / profile
6. DELETE /profile
7. POST / Send Request (ignore, intrested)
8. POST / Review Request (accept , reject )
9. GET /request
10. GET /connections

now writing code will be very easy for us  
see u in the enxt episode keep learning