Code Fellows

Class 08
Express Routing and Connected API

seattle-javascript-401n14

Lab 07 Review

Code Challenge 07 Review

Vocab Review!

What is a server?

What is a server?

A server is an application whose purpose is to provide data or expose APIs to another application, called the **client**. A single server can have multiple clients, and a single client can use multiple servers. We created a MongoDB server in Lab 05.

What is the client-side?

What is the client-side?

The client-side portion of an application requests data from a server and deals with user input / action. When we think of web applications, we want our client-side to be user-focused; only knowing, showing and dealing with thing the end-user needs.

What is an API?

What is an API?

API stands for Application Programming Interface, and it mainly means a function that one application exposes, and another application uses. Within a single application, the server exposes endpoints that the client can use, and thus a server creates an API.

What is an endpoint?

What is an endpoint?

An endpoint is a route in the server that clients can request data from. Each endpoint should return a response if there is a properly formatted request. An endpoint is the end of a communication channel between the client and server.

What is a development environment?

What is a development environment?

A development environment is a version of an application that is meant to be used while developing the code. Everything that an application depends on typically has a development version which can be freely modified without breaking the deployed version of the application.

What is a production environment?

What is a production environment?

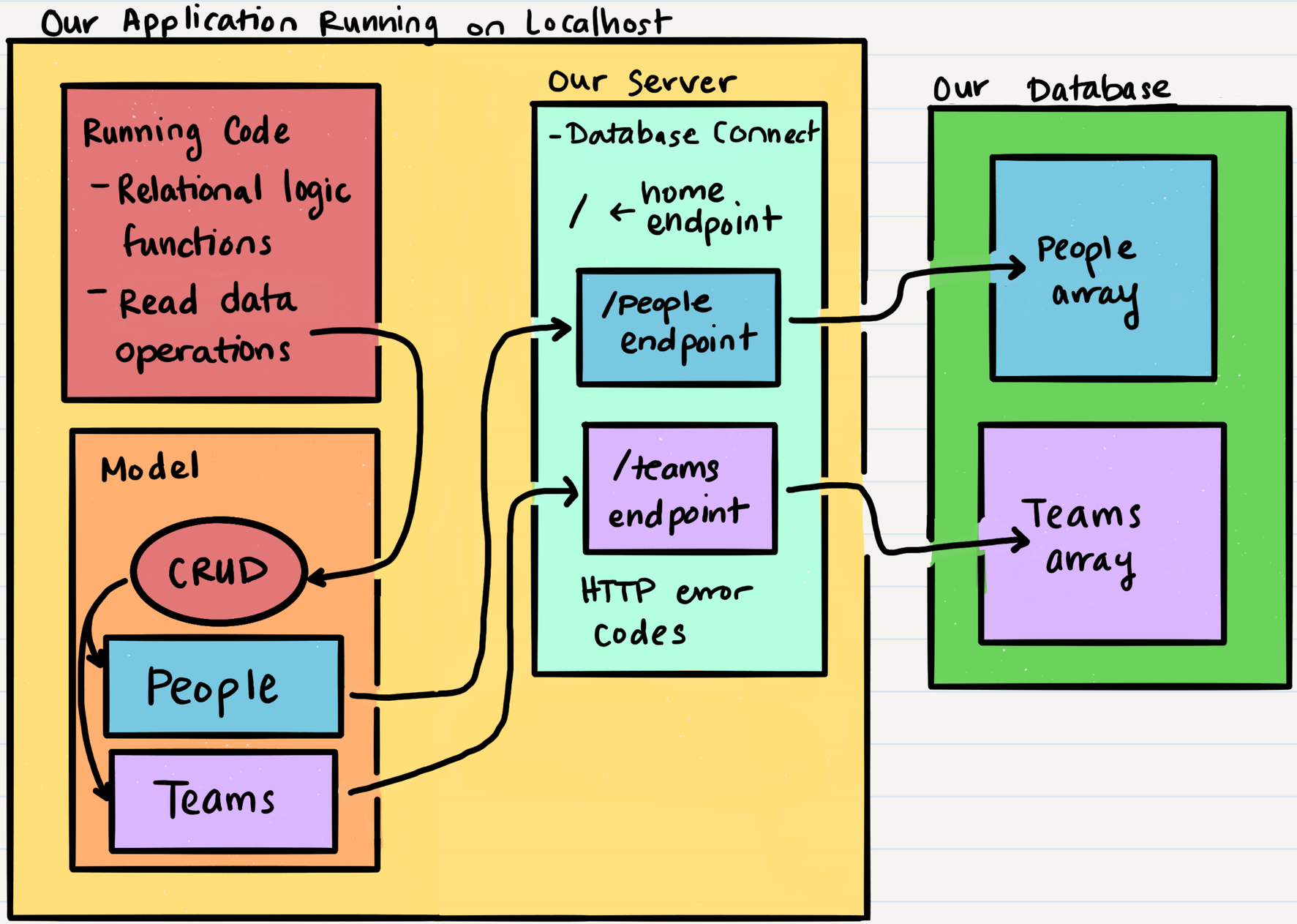
A production environment is the live version of your application that other users may access. This is typically a deployed application, with all the dependencies for the application (such as databases, external assets, etc) also deployed.

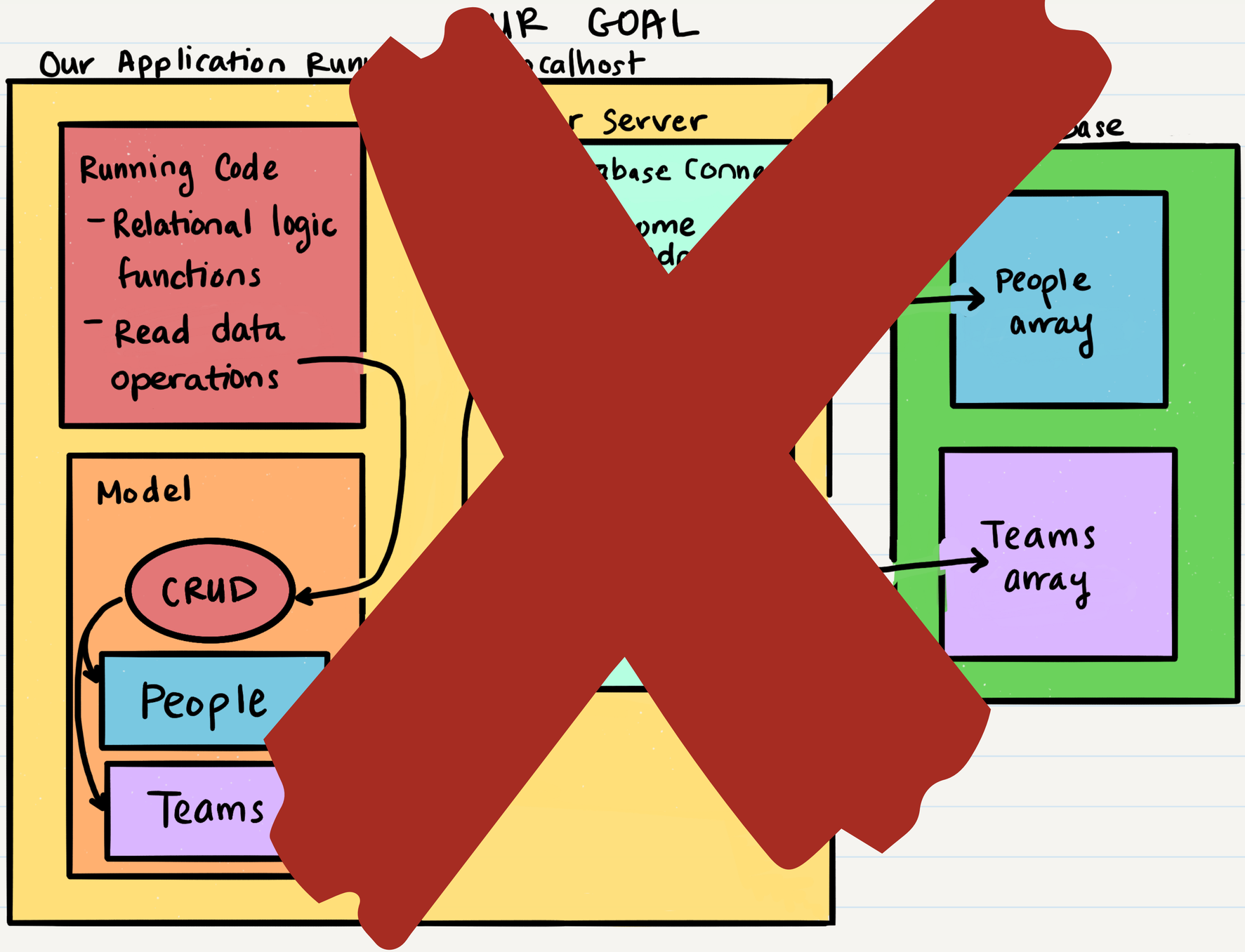
What is mongoose?

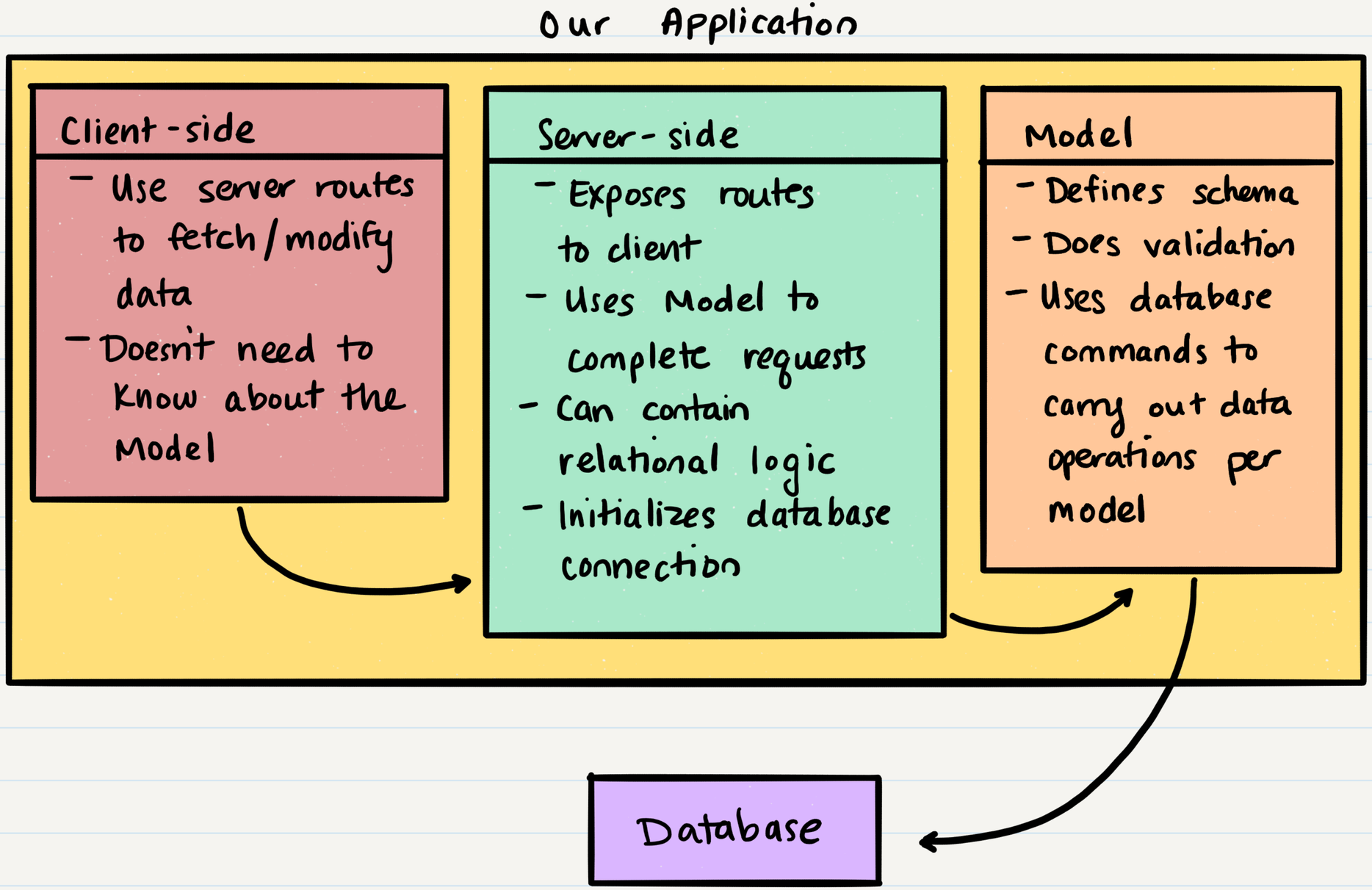
What is mongoose?

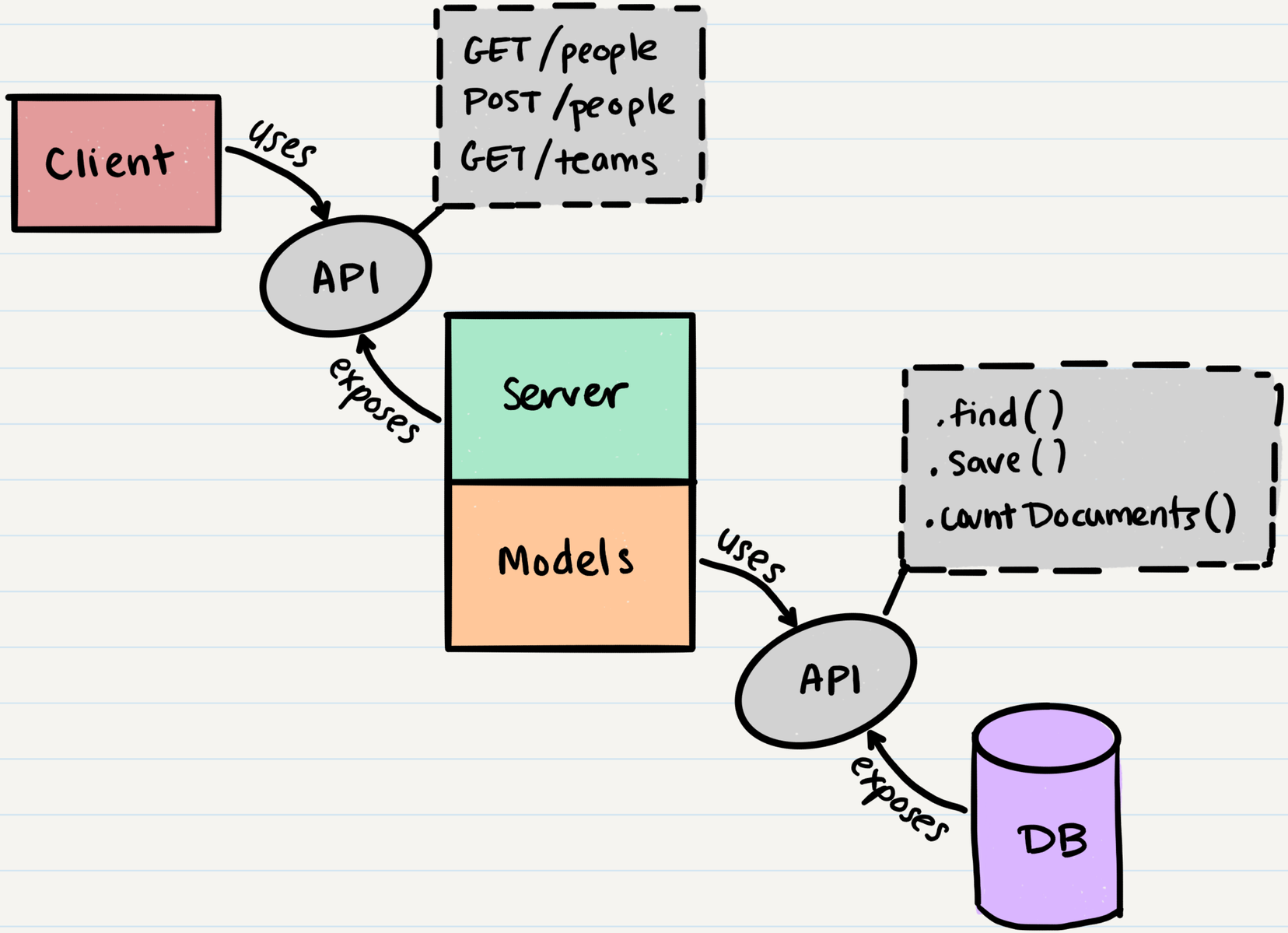
Mongoose is a middleman (aka middleware) between our Node.js applications and a MongoDB database. Mongoose helps us impose schema validation upon our database, and helps us properly connect to MongoDB.

OUR GOAL


Our Application Running on Localhost
Running Code
- Relational logic functions
- Read data operations
Model
CRUD
People
Teams
Our Server
-Database Connect
/ ← home endpoint
/people endpoint
/teams endpoint
HTTP error codes
Our Database
People array
Teams array


UR GOAL
Our Application Run
calhost
r Server
abase Conne
ome
Running Code
- Relational logic functions
- Read data operations
Model
CRUD
People
Teams
base
People array
Teams array


Our Application
Client-side
- Use server routes to fetch/modify data
- Doesn't need to know about the Model
Server-side
- Exposes routes to client
- Uses Model to complete requests
- Can contain relational logic
- Initializes database connection
Model
- Defines schema
- Does validation
- Uses database commands to carry out data operations per model
Database


GET /people
POST /people
GET /teams
Client
uses
API
exposes
Server
Models
uses
.find()
.save()
.countDocuments()
API
exposes
DB

From Last Time...

Function	Description
`app.use()` or `router.use()`	Applies middleware to every request made to this application or route
`app.all()` or `router.all()`	Applies middleware to each HTTP method endpoint on this appication or route
`app.get()` or `router.get()`	Applies middleware to each `GET` endpoint on this application or route
`app.post()` or `router.post()`	Applies middleware to each `POST` endpoint on this application or route
`app.put()` or `router.put()`	Applies middleware to each `PUT` endpoint on this application or route
`app.delete()` or `router.delete()`	Applies middleware to each `DELETE` endpoint on this application or route
`app.param()` or `router.param()`	Applies middleware to each request on this application or route that has a specific `req.param` key-value set

From Last Time...

Function	Description
`app.use()` or `router.use()`	Applies middleware to every request made to this application or route
`app.all()` or `router.all()`	Applies middleware to each HTTP method endpoint on this appication or route
`app.get()` or `router.get()`	Applies middleware to each `GET` endpoint on this application or route
`app.post()` or `router.post()`	Applies middleware to each `POST` endpoint on this application or route
`app.put()` or `router.put()`	Applies middleware to each `PUT` endpoint on this application or route
`app.delete()` or `router.delete()`	Applies middleware to each `DELETE` endpoint on this application or route
`app.param()` or `router.param()`	Applies middleware to each request on this application or route that has a specific `req.param` key-value set

New!

New!

```
9  const app = express();
10
11 // Actual Routes
12 app.get('/', (req, res, next) => {
13   res.send('Routing Demo');
14 });
15
16 app.use(myRoutes);
17 app.use('/your', yourRoutes);
18
19 app.get('/foo', (req, res, next) => {
20   console.log(req.param);
21   console.log(req.query);
22   console.log(typeof req.param.id, req.param.id);
23   res.send('ok');
24 });
25
26 app.get('/foo/:id', (req, res, next) => {
27   console.log(req.params);
28   console.log(req.query);
29   console.log(typeof req.params.id, req.params.id);
30   res.send('ok');
31 });
32
33 module.exports = {
34   server: app,
35   start: port => {
36     let PORT = port || process.env.PORT || 3000;
37     app.listen(PORT, () => console.log(`Listening on ${PORT}`));
38   }
39 };
```

Demo

demo/routing

We can use either req.query or req.param to see details about our path from the request, and to return more filtered data from that.

Building upon Routes

- We can access query parameters in our routes using `request.query`
- Better yet, we can structure our routes in a more complex way so we don't have to use queries!
    - Queries are more work on the client-side, which we don't want!
- By using r`equest.params`, we have more flexibility and can even create `param` middleware!

```
let db = require('./db.js');

app.use(express.json());

// Default Route
app.get('/', (req, res, next) => {
  res.send('Homepage');
});

// Route to Get all People
app.get('/people', (req, res, next) => {
  let count = db.people.length;
  let results = db.people;
  res.json({ count, results });
});

// Route to Get a person
app.get('/people/:id', (req, res, next) => {
  let id = req.params.id;
  let record = db.people.filter(record => record.id === parseInt(id));
  res.json(record[0]);
});

// Route to Create a person
app.post('/people', (req, res, next) => {
  let record = req.body;
  record.id = Math.random();
  db.people.push(record);
  res.json(record);
});
```

Lab

lab/starter-code

Let's walk through getting set up with our starter code for Lab 08.

We'll connect to a simple MongoDB database in our server, and set up a quick modular route.

What's Next:

- Due by Midnight Tomorrow: **Learning Journal 08**
- Due by Midnight Thursday: **Code Challenge 08**
- Due by 9am Saturday:
    - **Lab 08**
    - **Read: Class 09**
    - **Read: DSA 02**
- Next Class:
    - **Class 09 - API Server**

Questions?