Auth and Migrations

CSC309

Kianoosh Abbasi

UNIVERSITY OF TORONTO

Fall 2024

So far

- Next.js API handlers
- MVC and model design
- Prisma ORM
- CRUD

Next session

- Authentication and authorization
- Tokens and sessions
- Detailed discussion about migrations

Authentication vs Authorization

- Authentication:
  - + Who's calling?
  - - This is Daniel Liu
  - + Is it really Daniel Liu?
- Obtains user information from user/pass, session, API key, fingerprints, etc.

- Authorization:
  - Does Daniel Liu have enough access and permissions (aka authorized) to make this request?
- Checks user's properties and permissions

Authentication

- Client should tell us who they are
- Through request headers
- Several authentication methods
  - Basic auth
  - Session auth
  - Token auth

Basic auth

- Simply sends credentials at every request
  - User/pass, fingerprints, face ID, etc.
- No concept of login and logout
- So insecure: transfers raw sensitive data many times
  - If compromised, huge damage is incurred
- Not used in modern applications

Session auth

- Client sends user/pass at login
  - Or fingerprints, face ID, etc.
- If successful, server creates and stores a session id
  - Mapped to user
- Session id returned in the response
  - Browser saves it in cookies
- Browsers sends the same session id at next requests
  - Incognito tab: browser does not send the same session id

Token auth

Visit: https://www.javainuse.com/jwtgenerator

- Instead of a random session id, the token can contain information about the user

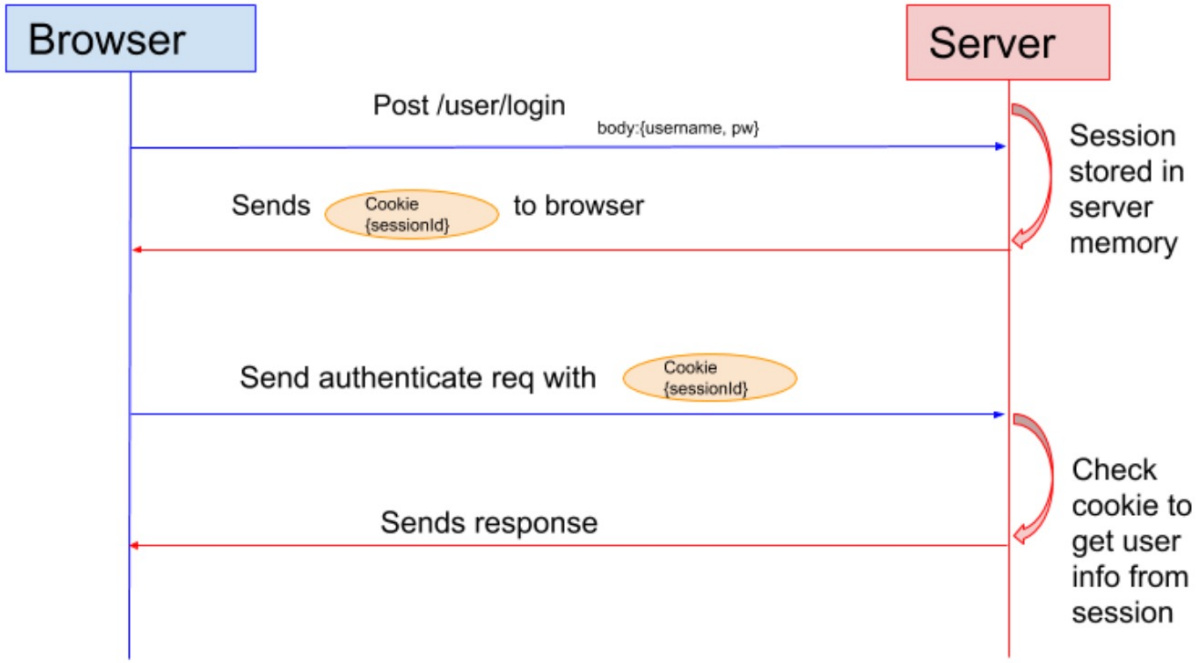
- It can be a JSON string

```
{ "userId": "134234", "expiresAt": 1722720863 }
```

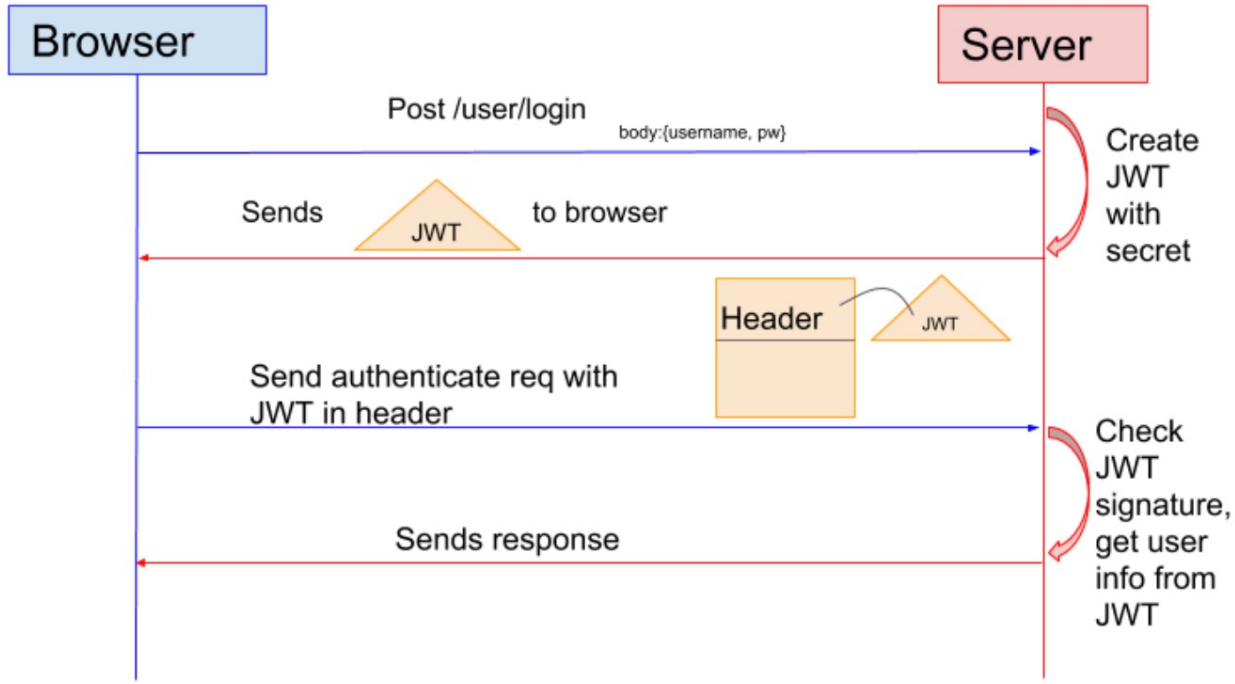
- Must be signed by the server to avoid attacks
  - Turned into a seemingly random string

```
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiIxMjM0IiwiZXhwaXJlc0F0IjoiMTcyMjcyMDg2MyJ9.UsTi2eDC5hrI1uqv-JzUf384g0QznPZomPfzJbdnMtY
```

Session auth



Token auth



Source: https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4

Session vs token auth

Session auth

- Less scalable
  - server stores all sessions
  - An additional database query per request
- More control
  - server can revoke a session

Token auth

- Simpler
  - No database interaction
- More scalable
  - Client in charge of storing the token
- Less control
  - Not revocable. True logout is not possible

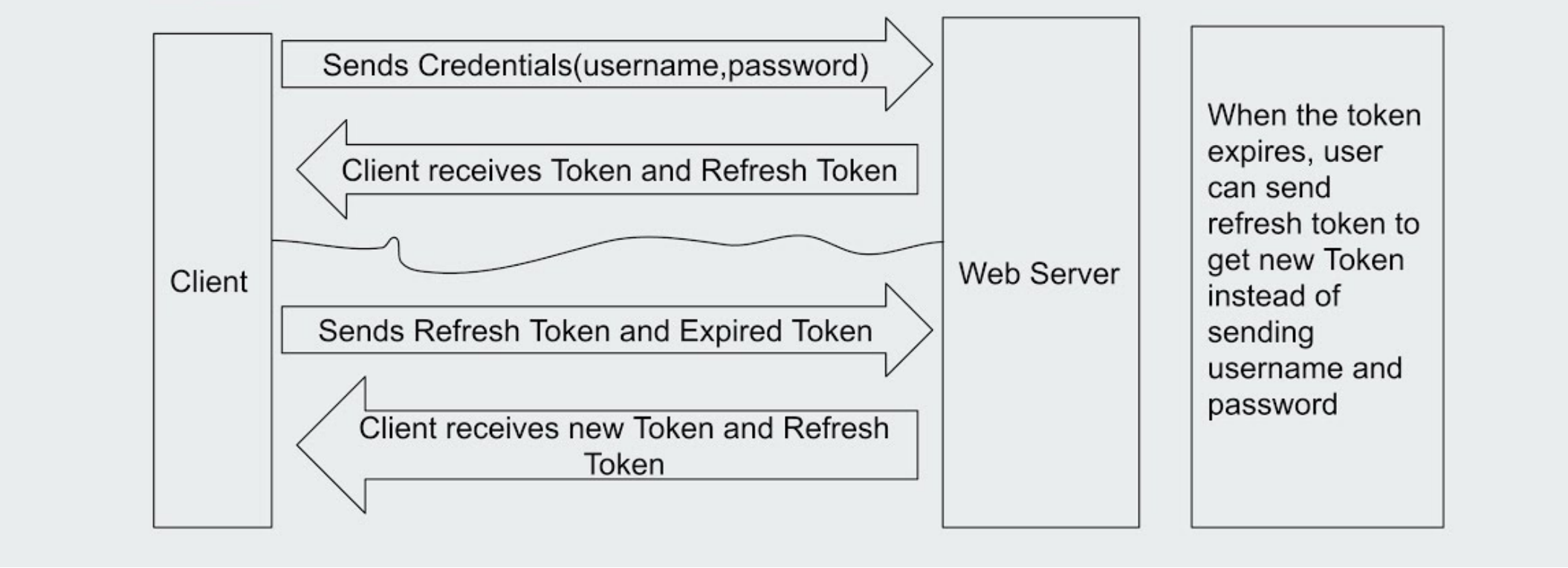
Best practices

- Token auth is preferred in modern web apps
  - Because of simplicity and scalability
- Known as JSON Web Token (JWT)
- They are generally very safe
  - Constructing a counterfeit token is almost impossible
- The main risk: compromised tokens
  - Tokens are not revocable: They should not be sent over and over

Best practices

- Short-lived tokens
  - Access tokens should expire within 15 minutes to an hour
- Having user authenticate every hour is a very bad UX
- Refresh tokens
  - Signed using a different secret
  - Can only be used to generate a new access token

Refresh tokens



Source: https://www.youtube.com/watch?v=yadjfgDBSiM&themeRefresh=1

Refresh tokens

- Refresh tokens last much longer
  - From hours to days or even weeks

- Upon receiving a 401 Unauthorized response:
  - Try refreshing the token first
  - Resend the request with the new access token

- Session continuity
  - User only re-authenticates when refresh token expires

Exercise: JWT auth in Next.js

Authorization

- Often, you should check several conditions before executing the API handler logic
  - Is the user authenticated?
  - Does the user have enough access?
    - e.g., being the owner of the store, or a follower of the author
- Return a 403 Unauthorized in those cases
- Should be re-usable logic, ideally separate from the handler logic
  - Often in middlewares

Migrations

The great assumption

- The state of database tables is the same as what defined in model schema

- But these two are totally independent things
  - Prisma models vs database tables

- ORM's job to apply application's schema to database
  - Via DDL queries

- Changes to schema's state:
  - Creation or removal of a table/model
  - Creation or removal of a column/field
  - Modification of field option/attributes

- Whenever the state changes, database should migrate to the new state

- Prisma does not do it automatically. WHY?

- You simply get a database exception if ORM's and database's schema do not match

Migration workflow

- Think about it as a git commit
  - Talks about what has changed since the last migration

- History of changes needs to be stored somewhere
  - The migrations folder

Migration workflow

```
migrations/
  └─ 20210313140442_init/
    └─ migration.sql
  └─ 20210313140442_added_job_title/
    └─ migration.sql
```

```sql
-- CreateTable
CREATE TABLE "Person" (
    "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    "name" TEXT NOT NULL,
    "email" TEXT NOT NULL,
    "age" INTEGER NOT NULL
);

-- CreateIndex
CREATE UNIQUE INDEX "Person_email_key" ON "Person"("email");
```

New migration

- Think about it as a new commit:
  - Includes what has changed since the last commit (i.e., migration)
- Builds the old database state from previous migrations
  - Does not contact the database
- Iterates over all models to find out differences
- Creates a new folder inside the migrations directory
  - Containing the DDL queries

New migration

- Migrations are created and applied via

```
npx prisma migrate dev
```

- But a migration should not be applied twice!
  - The same `CREATE TABLE` will not work again!
  - How is Prisma to know?

- Migrations themselves are stored in database

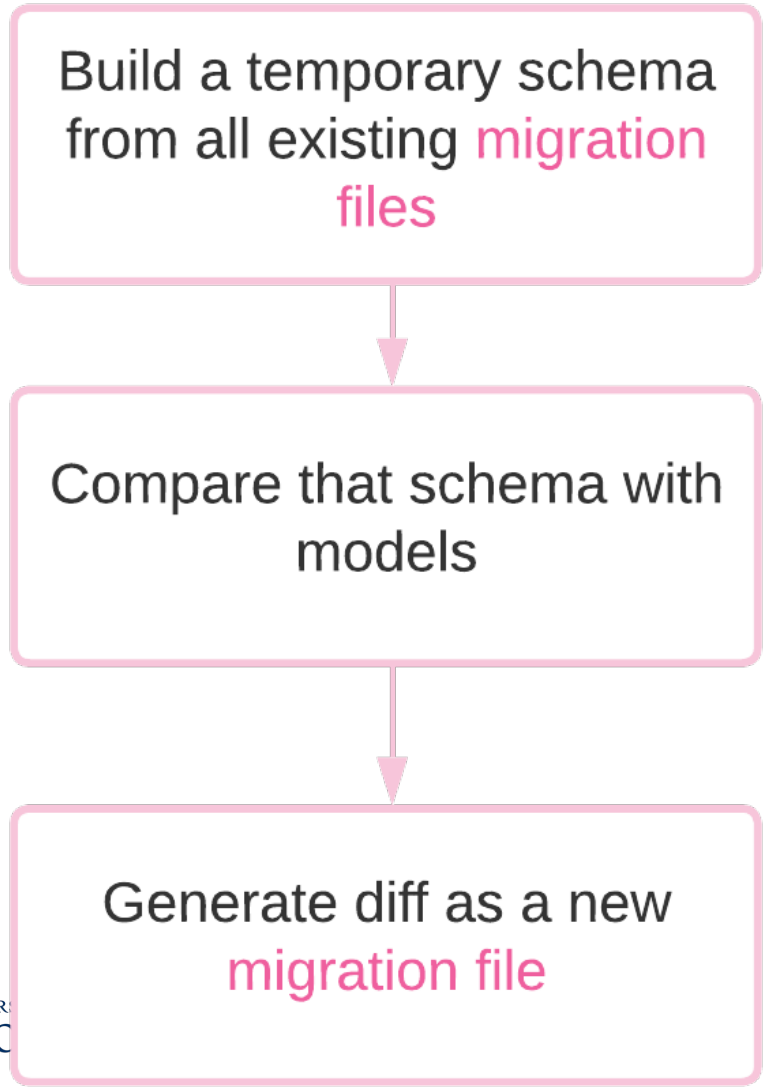
Migrations table

- Migrations are stored in `_prisma_migrations` table
- Stores the migrations' metadata
  - Content is only stored in the migration file
- `checksum` ensures migrations are not edited

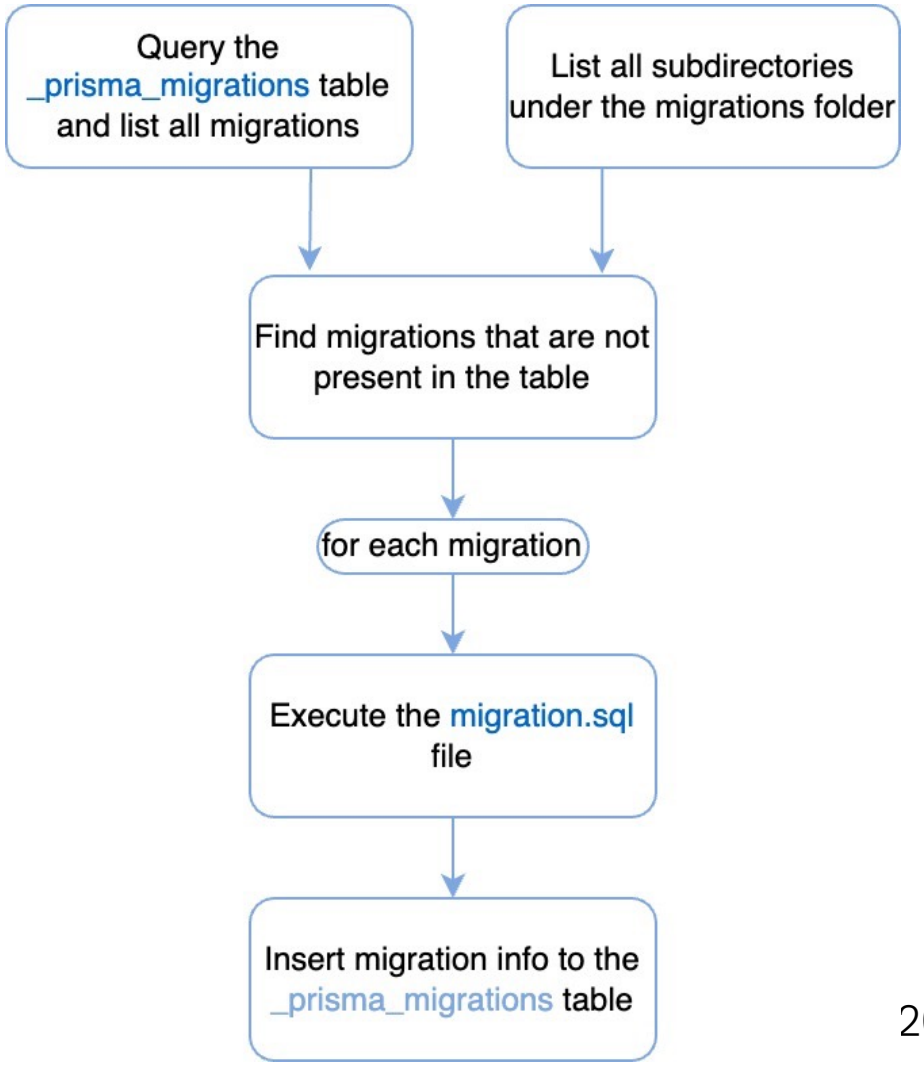
```
sqlite> PRAGMA table_info(_prisma_migrations);
0|id|TEXT|1||1
1|checksum|TEXT|1||0
2|finished_at|DATETIME|0||0
3|migration_name|TEXT|1||0
4|logs|TEXT|0||0
5|rolled_back_at|DATETIME|0||0
6|started_at|DATETIME|1|current_timestamp|0
7|applied_steps_count|INTEGER UNSIGNED|1|0|0
```

Migration workflow

Generate a new migration file



Apply the migrations to the database



Migration assumptions

- For this system to work, you must
  - Never directly change the database schema
    - e.g., manually running an `ALTER TABLE …`
  - Never edit/delete a migration file

- Migration files must be the same everywhere
  - Always push the migration files into git

- Migration errors can take hours to resolve!
  - Be cautious!

Migration commands

`npx prisma generate`

- Generates JavaScript code of the schema

`npx prisma migrate dev`

- Identifies schema changes since last migration
- Generates a new migration
- Applies unapplied migrations
- Should only be used in development (WHY?)

`npx prisma migrate deploy`

- Applies unapplied migrations (without creating new ones)
- Suitable for production

Migration errors

- Common scenarios:
  - You and your teammate added same or conflicting migrations independently
  - Someone manually updated the database schema
  - Someone created an failing migration
    - e.g., marking a column with `NULL` values as `NOT NULL`
  - Someone edited a migration file

- Very tricky:
  - Potential for data loss is high. This should be avoided at all costs!

Migration error solutions

Visit https://www.prisma.io/docs/orm/prisma-migrate/workflows/patching-and-hotfixing

- Resolve a migration

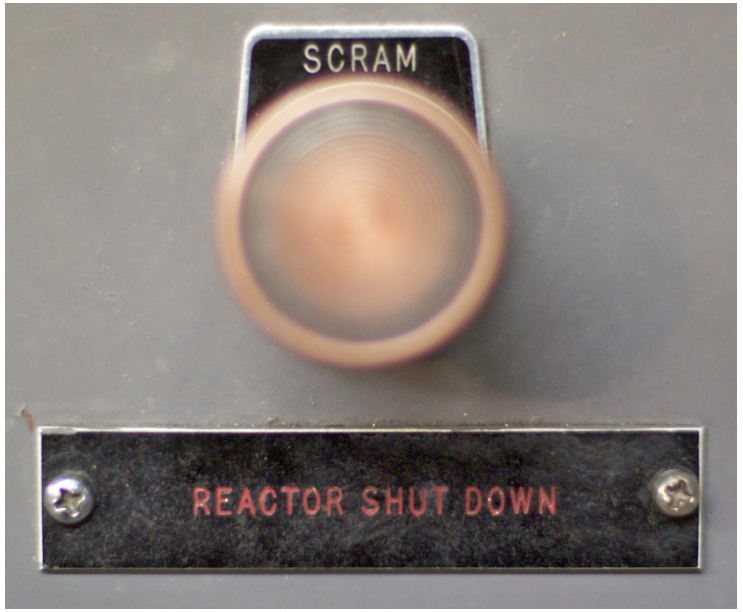
```
npx prisma migrate resolve --applied "migration_name"
npx prisma migrate resolve --rolled-back "migration_name"
```

- Will only update the migrations table, without executing the queries

- Manually sync database schema with migrations

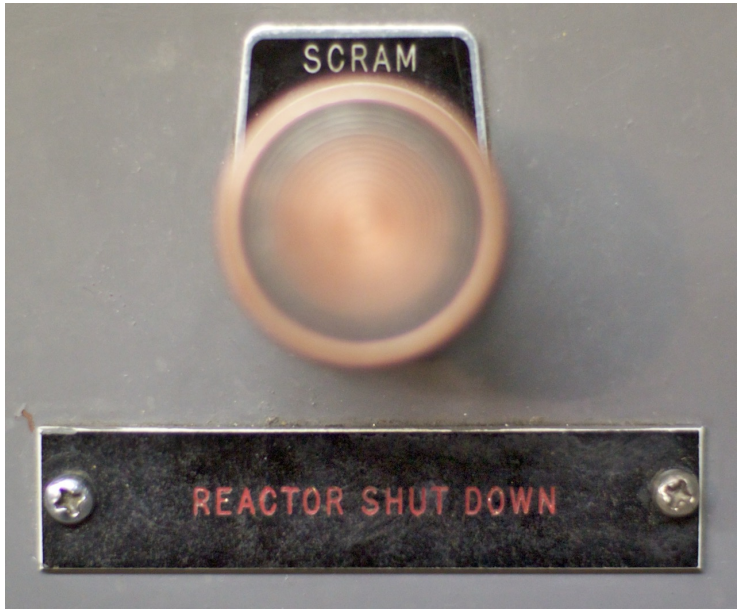
The last resort

- Reset the entire database
  ```
  npx prisma db reset
  ```
- Deletes all table's data
  - Applies the migrations on an empty database
- Definitely NOT an option in production
  - So be careful about migrations



The very last resort

- Delete the entire database
  - Just delete the dev.db file!
- Delete the migrations directory afterwards
- Restart with a fresh schema and generate new migrations!
- Definitely NOT an option in production
  - So be careful about migrations



Next session

- Begins (or resumes) our front-end journey
- Modern client-side JavaScript
  - React, JSX
- React application
  - Props
  - Events
  - State