



Databases & ORMs

Object Relational Mapper

- Acts as a “bridge” between your code and the RDBMS.
- Using ORM, data can be easily stored and retrieved from a database without writing SQL statements directly.

Sequelize

- **Sequelize is an Object-Relational Mapper (ORM)**
- **Access SQL databases from Node.js**
 - Using JS objects and methods instead of SQL statements
- Represents tables as “classes” and rows as objects (instances)

Without ORM

```
client.query(`select * from dogs`)
```

```
client.query(`select * from cats`)
```

```
client.query(`select * from hippos`)
```

With ORM

`Dog.findAll()`

`Cat.findAll()`

`Hippo.findAll()`



Tables

Models

+

=

+

Rows

Instances

Basic Workflow

Sequelize Basics: Workflow

- **Instantiate Sequelize**

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```


Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- Define your **Model(s)**

```
const User = db.define('user', {
  name: Sequelize.STRING,
  pictureUrl: Sequelize.STRING
});
```

Sequelize Basics: Workflow

- ◎ **Instantiate Sequelize**

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- ◎ **Define your **Model(s)****

- Add options to **Model** fields (validations, default values & more)

```
const User = db.define('user', {
  name: Sequelize.STRING,
  pictureUrl: Sequelize.STRING
});
```

Sequelize Basics: Workflow

- ⦿ **Instantiate Sequelize**

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- ⦿ **Define your **Model(s)****

- Add options to **Model** fields (validations, default values & more)

```
const User = db.define('user', {
  name: {
    type: Sequelize.STRING,
    allowNull: false
  },
  pictureUrl: Sequelize.STRING
});
```

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- Define your **Model(s)**

- Add options to **Model** fields (validations, default values & more)

```
const User = db.define('user', {
  name: {
    type: Sequelize.STRING,
    allowNull: false
  },
  pictureUrl: Sequelize.STRING
});
```

- Connect/sync the **Model** to an *actual* table in the database

```
await User.sync()
```

Sequelize Basics: Workflow

- Use the **Model** (Table) to find/create **Instances** (row)

```
const users = await User.findAll();
```

Sequelize Basics: Workflow

- Use the **Model** (Table) to find/create **Instances** (row)

```
const user = await User.findById(6);
```

Sequelize Basics: Workflow

- Use the **Model** (Table) to find/create **Instances** (row)

```
const person = new User({  
  name: "Kate",  
  pictureUrl: "http://fillmurray.com/10/10"  
});
```

Sequelize Basics: Workflow

- Use the **Model** (Table) to find/create **Instances** (row)
- Use the **Instances** to save / update / delete

```
const person = new User({  
  name: "Kate",  
  pictureUrl: "http://fillmurray.com/10/10"  
});
```

```
await person.save();
```


Additional Model Options

- Sequelize models can be extended **Hooks, Class & Instance Methods, Getter & Setters, Virtuals**, etc.

Hooks

Hooks

- When you perform various operations in Sequelize (creating, updating, destroying, etc), various “events” occur. These are called “lifecycle events”
- Hooks are like adding an event listener to these events
 - *“Every time a journal entry is created or updated, escape any dangerous sequences that could result in an XSS attack”*
 - *“Every time a user is updated with a new password, hash it so that the plaintext password doesn’t get saved in the database”*



What happens when we do this?

```
const pug = await User.create({  
  name: "Cody",  
  pictureUrl: "http://fillmurray.com/10/10"  
});
```

beforeValidate

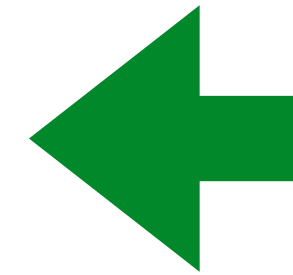
validation

afterValidate

beforeCreate

creation

afterCreate



```
User.beforeValidate((user) => {  
  })
```

beforeValidate

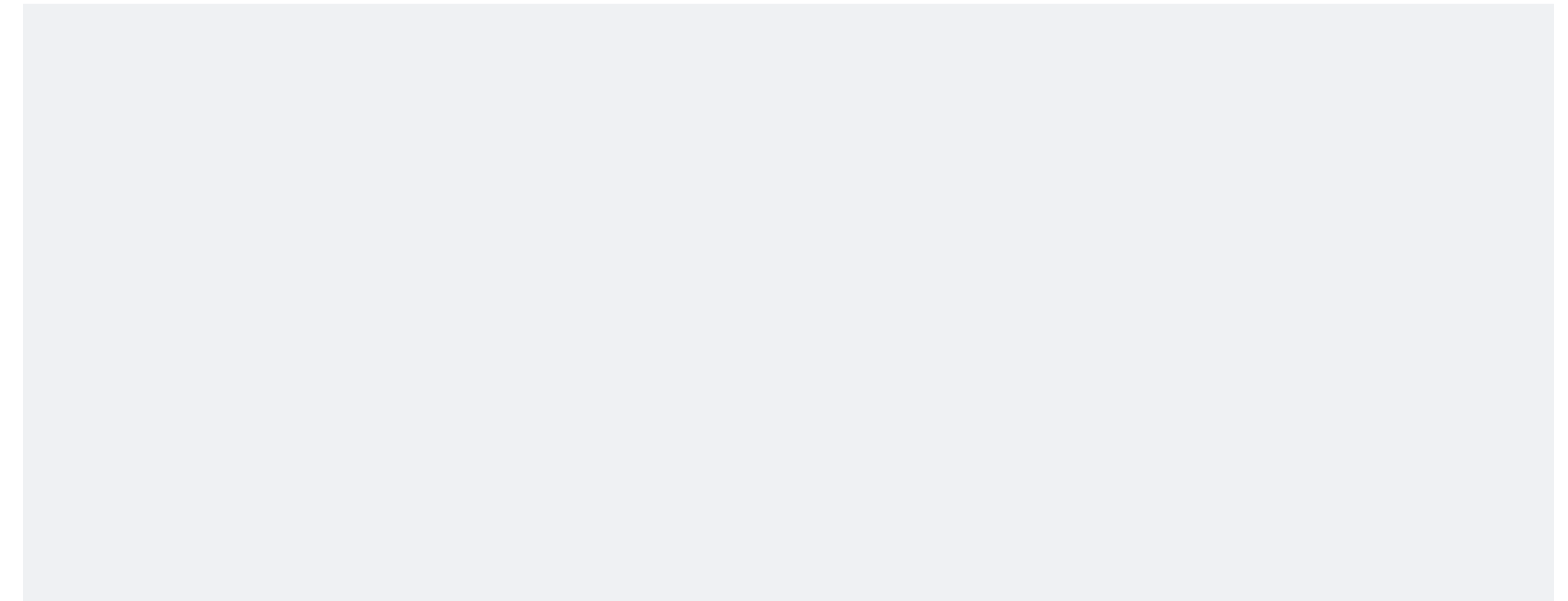
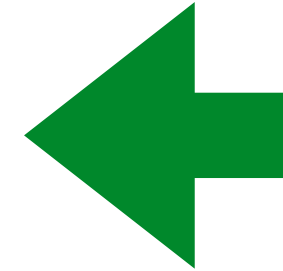
validation

afterValidate

beforeCreate

creation

afterCreate



beforeValidate

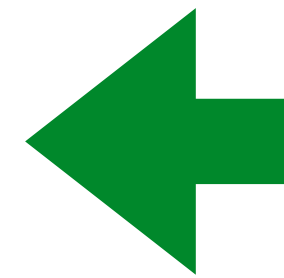
validation

afterValidate

beforeCreate

creation

afterCreate



```
User.afterValidate((user) => {  
  })
```

beforeValidate

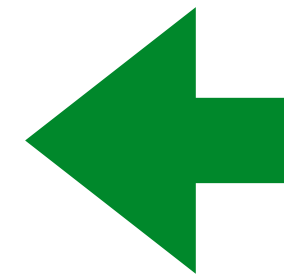
validation

afterValidate

beforeCreate

creation

afterCreate



```
User.beforeCreate((user) => {  
  })
```


beforeValidate

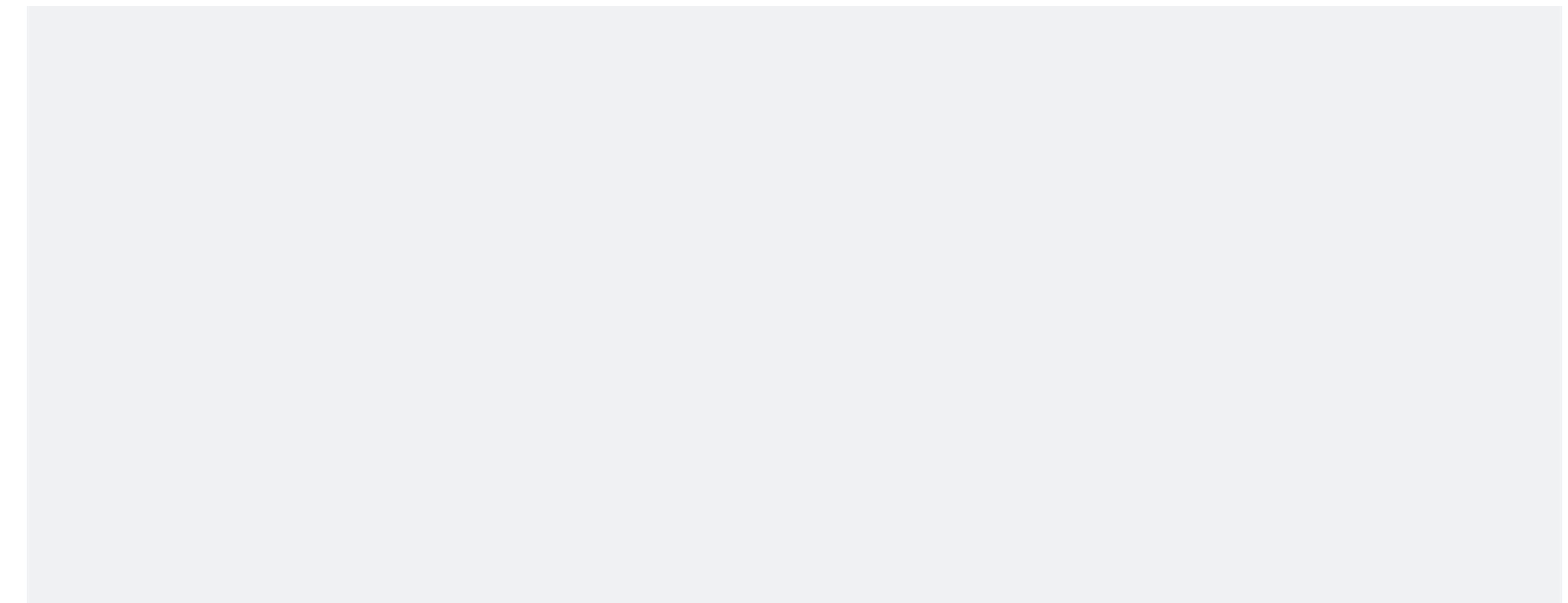
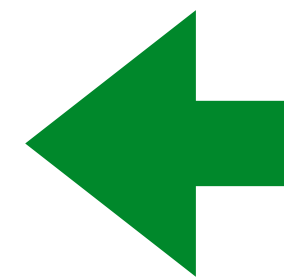
validation

afterValidate

beforeCreate

creation

afterCreate



beforeValidate

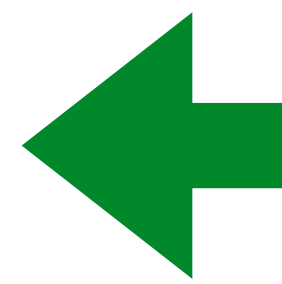
validation

afterValidate

beforeCreate

creation

afterCreate



```
User.afterCreate((user) => {  
  })
```

Associations

Associations

- Establishes a **relationship** between two tables (using a foreign-key or a join-table)
- Creates several special **instance methods** (like `getAssociation` & `setAssociation`), that an instance can use to search for the instances that they are related to.
- And more... (eager loading, etc)



Associations

```
const User = db.define("user", {...})  
const Pet  = db.define("pet", {...})
```

```
Pet.belongsTo(User)  
User.hasMany(Pet)
```



Associations

```
const User = db.define("user", {...})  
const Pet  = db.define("pet", {...})
```

```
Pet.belongsTo(User)  
User.hasMany(Pet)
```

```
const someUser  = await User.findById(12)  
const andHisPet = await someUser.getPets()
```

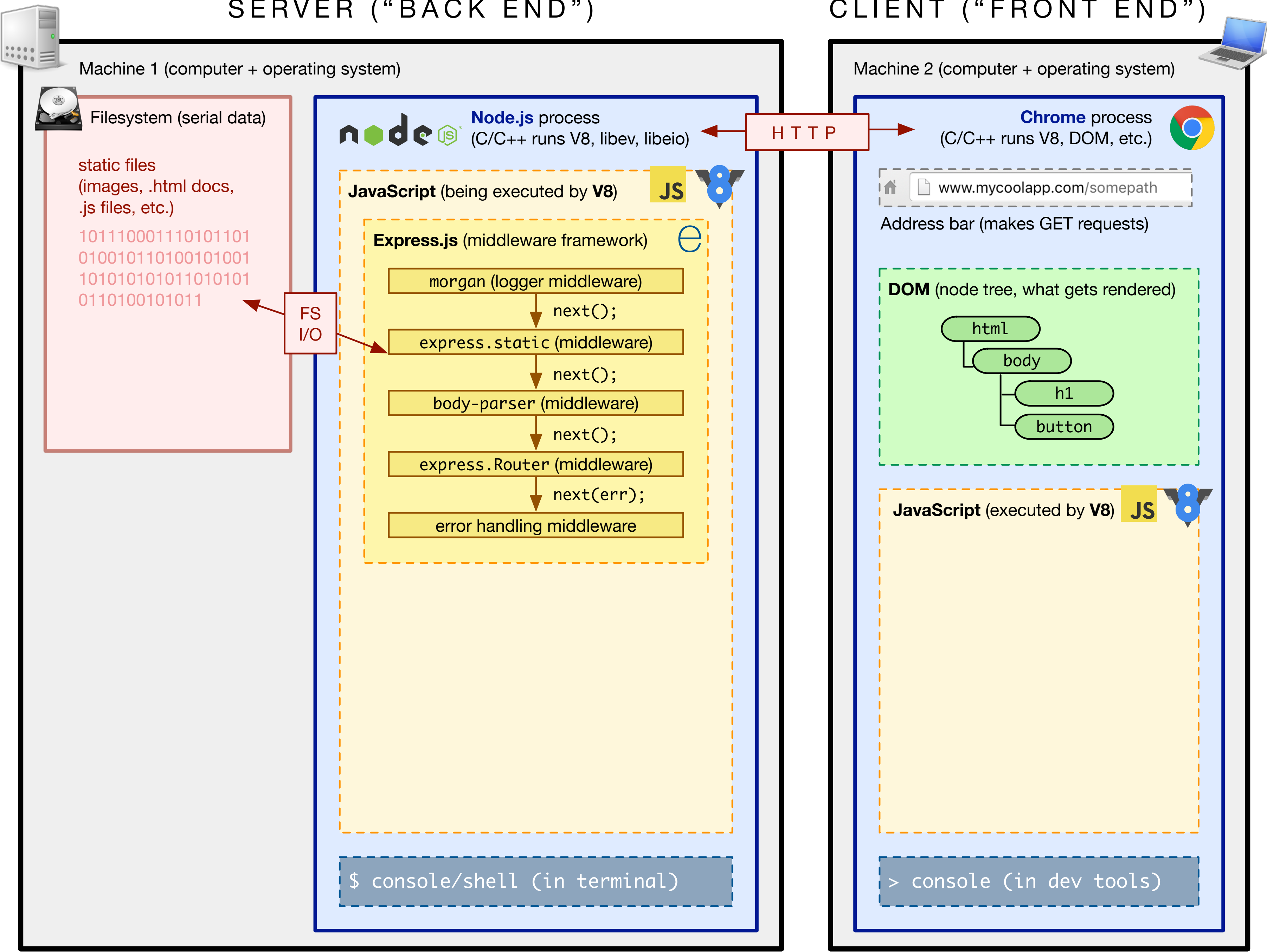
A little more context

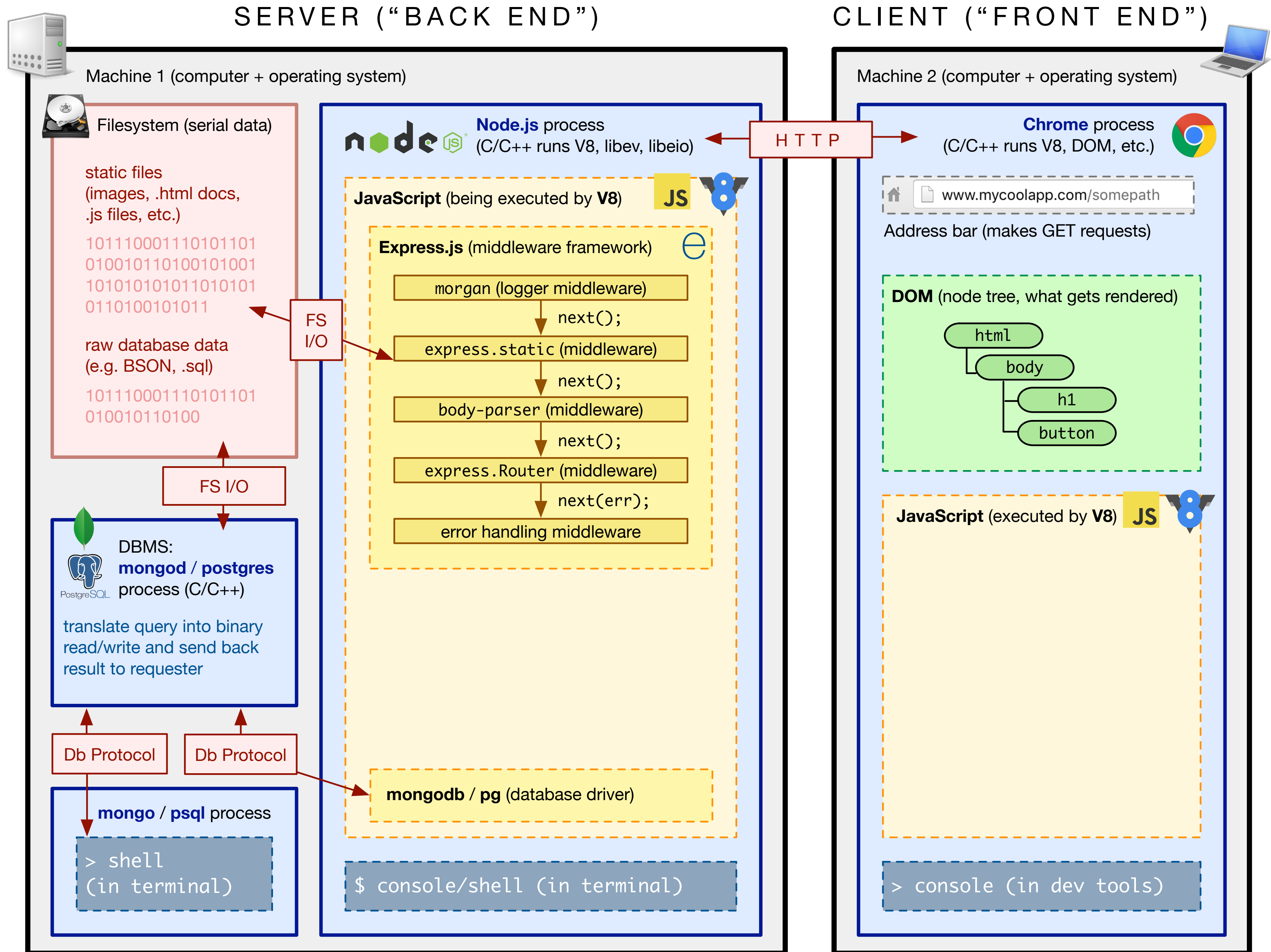
Sequelize

- Lives inside Node.js process
- Knows how to communicate to a few SQL DBMSs, including PostgreSQL and sqlite3

SERVER ("BACK END")

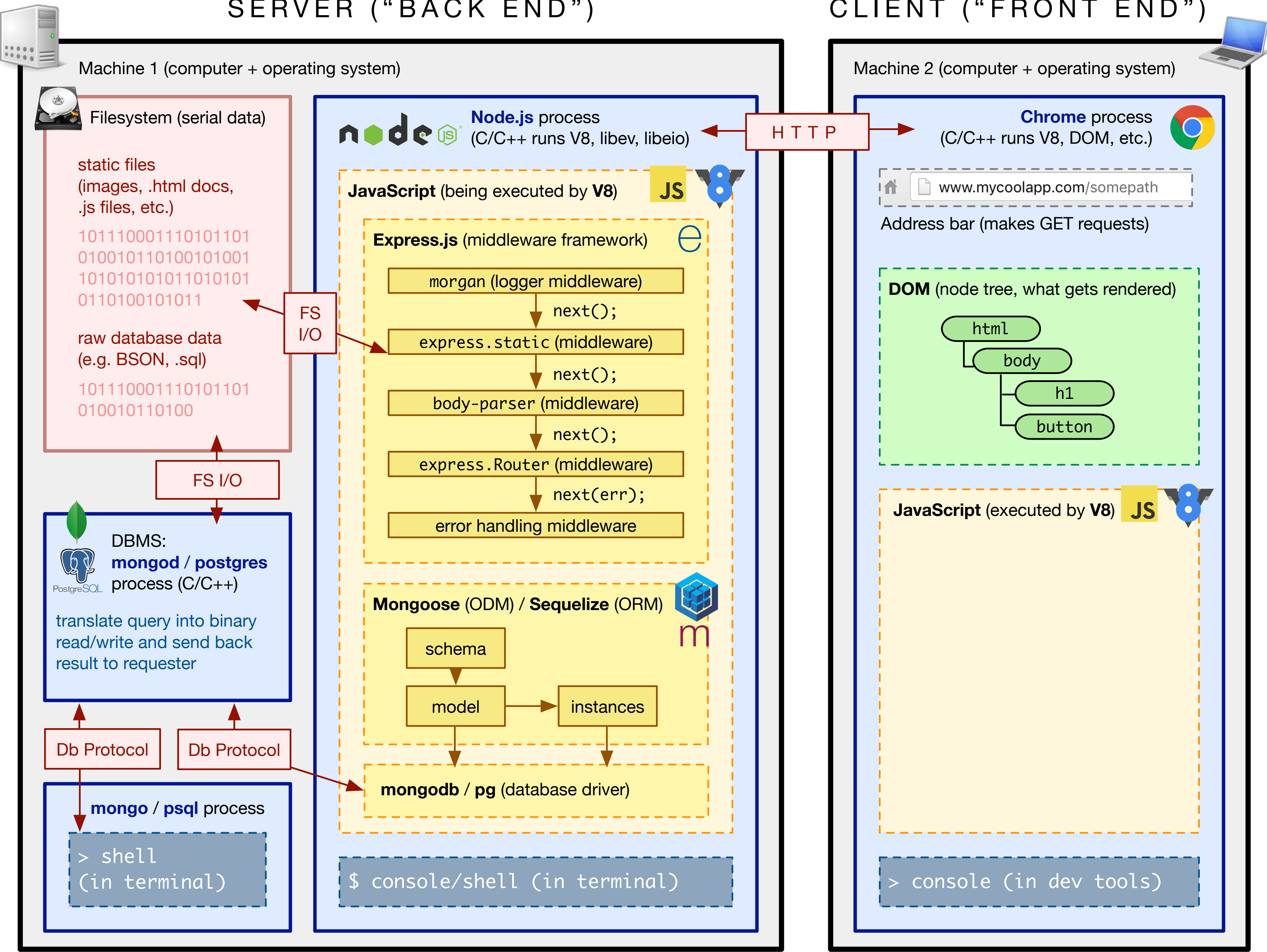
CLIENT ("FRONT END")





SERVER ("BACK END")

CLIENT ("FRONT END")



Wikistack

- Build a Wikipedia clone
- Walk you through installing and using sequelize
- Application of everything we've learned so far