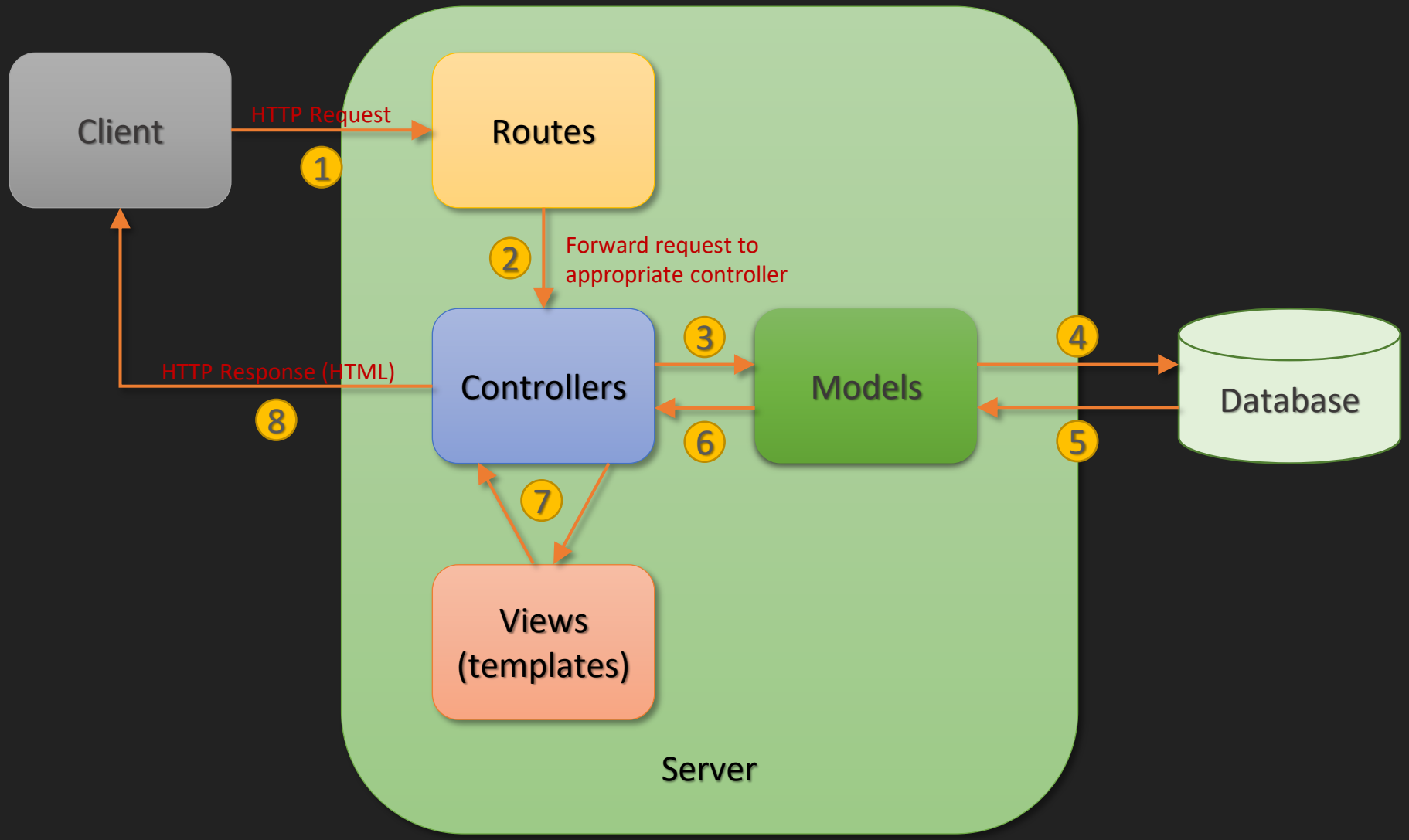
This work is licensed under CC BY 4.0

Persistent Data using a Document Database and Object Data Modeling

How to connect to MongoDB with Mongoose

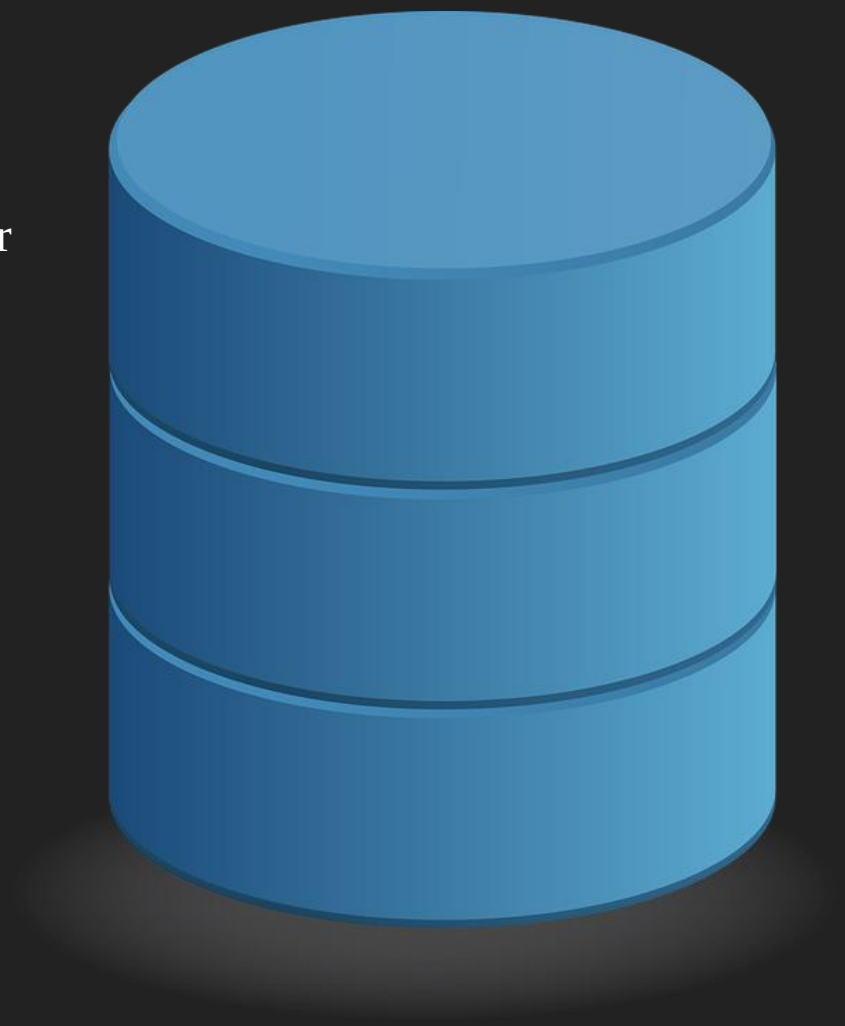


Overview of the CRUD Web Application



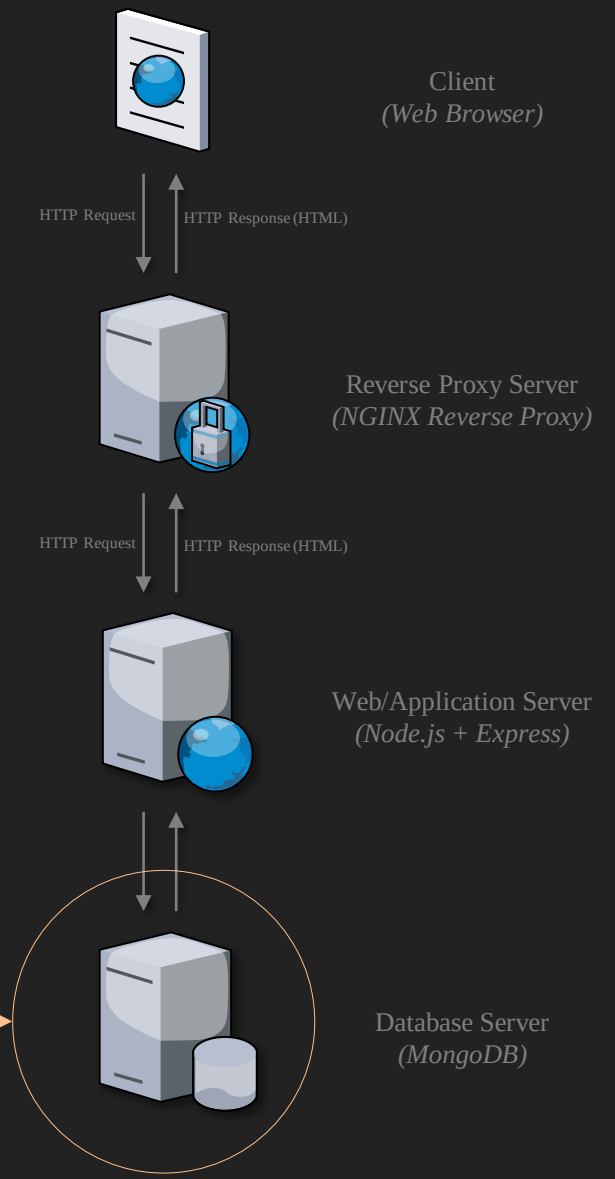
Persistent Data

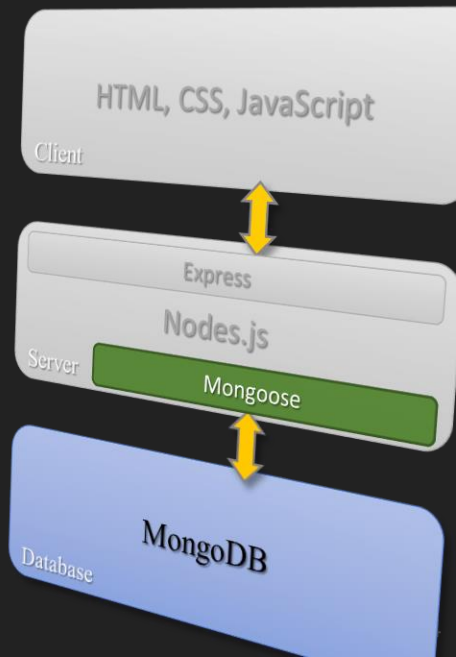
- ✓ Create, read, update and delete (CRUD) are the four basic function of persistent storage.
- ✓ How to store application data at the server side?
  - Data stored in the memory is non-persistent (data is erased when the power is turned off).
  - Filesystem persistence?
    - Hard to work with, performance, doesn't scale well.
  - Database persistence?
    - Oracle, SQL Server, MySQL, MariaDB, PostgreSQL
    - MongoDB
  - Cloud persistence?
    - mongodb.com (DBaaS, Database-as-a-Service), AWS, Azure, Google Cloud



You Will Use a Database

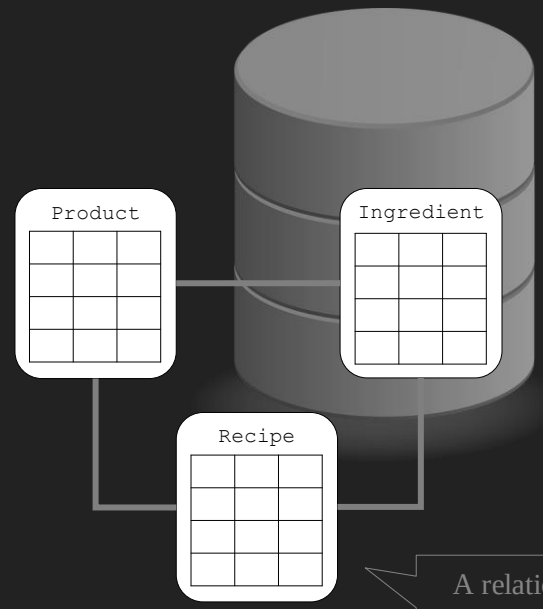
- ✓ A database is a collection of data that is organized so it can be easily accessed, manage and updated.
- ✓ Retrieve, add, modify, destroy
  - Query, filter, sort, ...
- ✓ You can host it yourself or let somebody else do it.





Different Types of Databases

- ✓ Relational database management systems (RDBMS)
  - Organizes data into tables with columns and rows.
  - Structured Query Language (SQL).
  - Oracle, SQL Server, MySQL, MariaDB, PostgreSQL, ...
- ✓ NoSQL Databases types (non-relational database)
  - Document databases
    - Organizes data into documents.
    - MongoDB, Firebase
  - Graph database
  - Key-value database



A relational database table organizes structured data fields into defined columns.



A non-relational database does not incorporate the table model. Instead, data can be stored in a single document file.

MongoDB Ecosystem



- MongoDB
  - An open-source document-based database system.
    - "MongoDB" derives from the word "humongous" because of the database's ability to scale up with ease and hold very large amounts of data.
  - Stores data in flexible, JSON-like documents.
  - The document model is simple to learn and use.
  - You can setup a MongoDB container in Docker in several ways.
    - Getting Started With MongoDB As A Docker Container Deployment
  - MongoDB Atlas
    - MongoDB Atlas is a database-as-a-service (DBaaS), a cloud computing service.
    - Easy to get started at https://www.mongodb.com/cloud/atlas.



- Mongoose
  - An Object Data Modeling (ODM) library for MongoDB and Node.js.
  - To get started, visit https://mongoosejs.com/docs/index.html.
    - `npm install mongoose`
  - Be sure to check out the documentation at https://mongoosejs.com/docs/api.html.

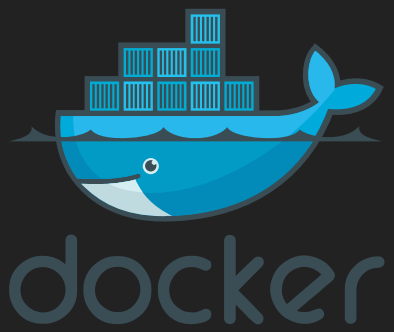
MongoDB as a Docker Container

- ✓ You can use Docker to deploy MongoDB as a container (and connect to the instance using Mongoose).
  1. Start Docker.
  2. Download the latest MongoDB Docker image.

     ```
     docker pull mongo:6.0.4
     ```
  3. Deploy an instance of MongoDB as a container.

     ```
     docker run -d -p 27017:27017 --name mongodb mongo:6.0.4
     ```

     - `docker run`, instantiates a container.
     - `-d`, run the container in the background.
     - `-p`, map the container ports with the host ports so the database can be accessed from the host computer.
     - `--name mongodb`, the container's name (if not present a the daemon generates a random string).
     - `mongo:5.0.5`, the image to derive the container from.



- ✓ You can interact with the deployed MongoDB.
  - Connect to MongoDB and start the bash shell.

    ```
    docker exec -it mongodb bash
    ```
  - Launch the MongoDB shell.

    ```
    # mongosh
    ```
    - The "mongo" shell has been superseded by "mongosh".
  - List databases.

    ```
    > show dbs
    ```



Connecting to MongoDB with Mongoose

```
// src/config/mongoose.js
import mongoose from 'mongoose'

/**
 * Establishes a connection to a database.
 *
 * @returns {Promise} Resolves to this if connection succeeded.
 */
export const connectDB = async () => {
  const { connection } = mongoose

  // Bind connection to events (to get notifications).
  connection.on('connected', () => console.log('MongoDB connection opened.'))
  connection.on('error', err => console.error(`MongoDB connection error occurred: ${err}`))
  connection.on('disconnected', () => console.log('MongoDB is disconnected.'))

  // If the Node.js process ends, close the connection.
  process.on('SIGINT', () => {
    connection.close(() => {
      console.log('MongoDB disconnected due to application termination.')
      process.exit(0)
    })
  })

  // Connect to the server.
  return mongoose.connect(process.env.DB_CONNECTION_STRING)
}
```

- ✓ You connect to MongoDB with the `mongoose.connect()` method.
  - The connection string should be stored in an environment variable (use the dotenv package while developing).

    ```
    DB_CONNECTION_STRING="mongodb://localhost:27017/<name>"
    ```

  - The connection string, enforcing access control, can be found at the DBaaS provider.

    ```
    mongodb+srv://<dbuser>:<password>@<cluster>.mongodb.net/<name>?retryWrites=true&w=majority
    ```

  - Returns a Promise object.
- ✓ Listen for connection events.
  - When successfully connected (`connected`).
  - If the connections throws an error (`error`).
  - When the connection is disconnected (`disconnected`).
- ✓ If the Node.js process ends, close the connection to MongoDB.

Establish a connection to the database

```
// src/server.js
import { connectDB } from './config/mongoose.js'

try {
  await connectDB()

  const app = express()
  const directoryFullName = dirname(fileURLToPath(import.meta.url))

  ...
} catch (err) {
  console.error(err)
  process.exitCode = 1
}
```

- ✓ If it is not possible to connect to the database, there is no point in starting the application.
- ✓ By waiting for the asynchronous function `connectDB()` to resolve, it is possible to determine if there is any point in creating, configuring, and running the Express application.



Everything Starts With a Schema

- A schema defines the shape (key and SchemaType) of the documents to be stored in the associated MongoDB collection.
  - Read more at https://mongoosejs.com/docs/guide.html.

```
const schema = new mongoose.Schema({
  name: { type: String, required: true },
  size: { type: Number, required: true, min: 15, max: 47 }
})
```

- A model is compiled from a Schema definition.
  - The first argument is the singular name of the collection the model is for.
  - The lowercase name of the collection will automatically be the plural version of the model's name.

```
const Shoe = new mongoose.model('Shoe', schema)
const myShoe = new Shoe()
```

Converting a Schema Into a Model

```
// src/models/task.js
import mongoose from 'mongoose'

// Create a schema.
const schema = new mongoose.Schema({
  description: {
    type: String,
    required: true,
    trim: true,
    minlength: 1
  },
  done: {
    type: Boolean,
    required: true,
    default: false
  }
}, {
  timestamps: true,
  toObject: {
    virtuals: true, // ensure virtual fields are serialized
    transform: function (doc, ret) {
      delete ret._id
      delete ret.__v
    }
  }
})

schema.virtual('id').get(function () {
  return this._id.toHexString()
})


// Create a model using the schema.
export const Task = mongoose.model('Task', schema)
```

Mongoose adds `createdAt` and `apdatedAt` properties to the schema and manages those properties.

- ✓ Convert a schema into a model by passing the schema to `mongoose.model(modelName, schema)`.
- ✓ An instance of a model is called a document.
- ✓ The module (`task.js`) resides in the `models` directory.
- ✓ You can add the `toObject()` method to the schema to control how a document transforms a document into a plain-old JavaScript object (POJO).
- ✓ You can add the `toJSON()` method to the schema to control how a document transforms a document into JSON.

